Шнейвайс А.Б.

ПРАКТИКА ПРОГРАММИРОВАНИЯ

(ФОРТРАН, СИ)
Второй семестр (часть 1)

для студентов астрономического отделения
математико-механического факультета СПбГУ

2018

Шнейвайс А.Б.

Учебное пособие «Практика программирования на ФОРТРАНе и СИ» (второй семестр) содержит решения наборов задач, которые предлагались в прошлые годы во втором семестре студентам первого курса астрономического отделения математико-механического факультета СПбГУ. Рассматриваемые темы задач близки по содержанию материалу, излагаемому в соответствующем лекционном курсе. Перед решением приведённых задач студенту полезно выполнить некоторые программы, приводимые на лекциях. Основные темы затрагиваемые во втором семестре: работа с несложными производными типами данных, рекурсивные процедуры, статические и размещаемые массивы, элементарная работа с ФОРТРАН- и СИ-указателями, ФОРТРАН-операции над массивами, встроенные ФОРТРАН-процедуры по работе с массивами, модульное программирование, использование в ФОРТРАНе механизма перегрузки функций и подпрограмм.

В некоторых случаях приводится несколько решений одной и той же задачи с объяснением ошибок, часто встречающихся, при выполнении заданий. Иногда решение задачи по какой-то конкретной теме может содержать (в незначительном объёме) некоторый новый материал, который служит ответом на вопросы, возникающие у добросовестного старательного студента.

Содержание

1 Рекурсивные процедуры (второй семестр).

Каждая тестирующая программа должна оценивать время работы соответственно обоих вариантов расчета, обеспечивая ввод исходных из файла и вывод результата в файл.
make-файл должен инициировать выполнение программы, используя утилиту **time**.

1.1 Задача 1

Разработать на ФОРТРАНе и СИ нерекурсивную и рекурсивную процедуры целочисленного деления одного целого числа на другое, моделируя операцию деления операциями вычитания.

1.2 Задача 2

Разработать на ФОРТРАНе и СИ нерекурсивную и рекурсивную процедуры нахождения остатка от целочисленного деления одного целого на другое, моделируя операцию нахождения остатка операциями вычитания;

1.3 Задача 3

Разработать на ФОРТРАНе и СИ нерекурсивную и рекурсивную процедуры перевода введенного целого в двоичную систему счисления.

1.4 Задача 4

Разработать на ФОРТРАНе и СИ нерекурсивную и рекурсивную процедуры возведения основания в неотрицательную целочисленную степень, моделируя операцию возведения в степень операциями умножения.

1.5 Задача 5

Разработать на ФОРТРАНе и СИ нерекурсивную и рекурсивную процедуры перевода введенного неотрицательного вещественного числа меньшего единицы в двоичную систему счисления

1.6 Задача 6

Разработать на ФОРТРАНе и СИ нерекурсивную и рекурсивную процедуры которые выясняют, является ли заданное натуральное число простым.

1.7 Задача 7

Разработать на ФОРТРАНе и СИ нерекурсивную и рекурсивную процедуры поиска ближайшего простого числа большего заданного натурального.

1.8 Задача 8

Построить дерево вызовов рекурсивной процедуры **Ханойская башня** (см. лекцию).

1.9 Задача 9

Продемонстрировать работоспособность всех лекционных алгоритмов, реализующих расчет чисел Фибоначчи.

1.10 Задача 10

Продемонстрировать работу нерекурсивной и рекурсивной процедур реализующих поиск изолированного корня уравнения f(x)=0 методом половинного деления (формулу для функции f(x) выбрать самостоятельно).

1.11 Задача 1 (ФОРТРАН-решение)

Главная программа демонстрирует вызов нерекурсивной **divn(n,m)** и рекурсивной **divr(n,m)**, которые моделируют операцию целочисленного деления одного целого на другое посредством операции вычитания.

В первой части программы каждая из функций вызывается один раз для вводимых значений **n** и **m**. Результат **divn(n,m)** помещается в переменную **kn**; результат **divr(n,m)** — в **kr**.

Во второй части — каждая из функций вызывается **kmax=10000** раз для **n=99999** и **m=2**. Перед **kmax**-кратным вызовом каждой и после него в переменные **t0**, **t1** и **t2** посредством встроенной ФОРТРАН-функции **second** (синоним **cpu_time**) заносятся моменты процессорного времени, фиксирующие начало и завершение **kmax**-кратного вызова и **divn(n,m)**, и **divr(n,m)** с выводом соответствующих временных затрат. Ожидается, что временные затраты нерекурсивной **divn** существенно меньше временных затрат рекурсивной **divr**.

Напомним, что в ФОРТРАНе (в отличие от СИ)

- аргументы процедур передаются по *ссылке*, т.е. формальному аргументу всегда передаётся **адрес** фактического. Тем самым, работа процедуры идёт непосредственно по адресу фактического аргумента. Если внутри процедуры меняется значение какого-то формального аргумента, то меняется и значение соответствующего фактического. Последнее не всегда удобно. Так, после вызова **divn(n,m)**, когда **n=6** и **m=2** значение **n** окажется равным **нулю**. А тогда последующий вызов **divn(n,m)** (или **divr(n,m)**) будет работать со значениями **n=0** и **m=2**, что приведёт к результату отличному от её первоначального вызова (мы хотим видеть совпадение результатов).

- Обеспечить неизменность фактического аргумента **n** (при изменении соответствующего формального) можно, например, копированием формального в дополнительную рабочую переменную, имя которой и использовать в дальнейшем при записи тела процедуры.

- Однако, в ФОРТРАНе (в случае простых переменных) существует и другой способ, не требующий явного описания рабочей переменной. Именно, в операторе вызова процедуры имя фактического аргумента, (значение которого может измениться) следует заключить

в скобки (т.е. не **divn(n,m)**, а **divn((n), m)**). В этом случае первый аргумент рассматривается компилятором как выражение, которое предварительно должно быть вычислено. Результат расчёта будет храниться не по адресу фактического аргумента и, тем самым изменение формального не затронет фактического.

- Программист сам вправе решать, какой из способов предпочесть. Ниже для демонстрации использован последний способ.

- В случае рекурсивной **divr(n,m)** нет нужды заботиться о сохранении первоначального значения **n**, поскольку при каждом рекурсивном вызове под аргументы **n** и **m** будут фрахтоваться новые ячейки памяти, так что исходное значение **n** не изменится.

1.11.1 Тестирующая программа

```
program test_my_div; use my_div; implicit none
integer, parameter :: kmax=10000
integer n, m, kn, kr, k
real t0, t1, t2
read (*,*) n, m; write(*,*) ' n=', n; write(*,*) ' m=', m
kn=divn( (n), m)
write(*,'(a,i7,a,i7)') ' после вызова divn(n,m):  n=',n,' kn=',kn
kr= divr(  n, m)
write(*,'(a,i7,a,i7)') ' после вызова divr(n,m):  n=',n,' kr=',kr
n=299999; m=2;  write(*,'(a,i7,a,i7)') '  n=', n,'   m=', m
call cpu_time(t0); do k=1,kmax; kn=divn( (n), m); enddo
call cpu_time(t1); do k=1,kmax; kr=divr(  n , m); enddo
call cpu_time(t2)
write(*,'(a,i7,a,i7,a,i7,a,i7,a,i7)')   ' kn=',kn,'  kr=',kr
write(*,*) ' Время на ',kmax,' вызовов  divn  равно',t1-t0
write(*,*) ' Время на ',kmax,' вызовов  divr  равно',t2-t1
end
```

1.11.2 Исходный текст модуля my_div

```
module my_div; implicit none
contains
function divn(n,m); integer divn, n, m, k
k=0
do while (n.ge.m)
  n=n-m
  k=k+1
enddo
divn=k
end function divn
recursive function divr(n,m) result(k)
integer n, m, k
if (n<m) then; k=0
         else; k=divr(n-m,m)+1
endif
end function divr
end module my_div
```

1.11.3 Вводимые данные n и m

```
6   2
```

```
 n=           6         ! Вывод test_my_div по опции -O0:
 m=           2
после вызова divn(n,m):  n=      6 kn=      3
после вызова divr(n,m):  n=      6 kr=      3
 n= 299999   m=      2
kn= 149999  kr= 149999
 Время на        10000  вызовов  divn  равно   4.60799980
 Время на        10000  вызовов  divr  равно   34.9400024

 n=           6         ! Вывод test_my_div по опции -O1:
 m=           2
после вызова divn(n,m):  n=      6 kn=      3
после вызова divr(n,m):  n=      6 kr=      3
 n= 299999   m=      2
kn= 149999  kr= 149999
 Время на        10000  вызовов  divn  равно   1.05999994
 Время на        10000  вызовов  divr  равно   15.9160004

 n=           6         ! Вывод test_my_div по опции -O2:
 m=           2
после вызова divn(n,m):  n=      6 kn=      3
после вызова divr(n,m):  n=      6 kr=      3
 n= 299999   m=      2
kn= 149999  kr= 149999
 Время на        10000  вызовов  divn  равно  0.995999992
 Время на        10000  вызовов  divr  равно   15.6199989

 n=           6         ! Вывод test_my_div по опции -O3:
 m=           2
после вызова divn(n,m):  n=      6 kn=      3
после вызова divr(n,m):  n=      6 kr=      3
 n= 299999   m=      2
kn= 149999  kr= 149999
 Время на        10000  вызовов  divn  равно  0.995999992
 Время на        10000  вызовов  divr  равно   4.46000004
```

Видно, что

- gfortran-время (версия 4.8.5) работы нерекурсивной при **n=299999** и **m=2** вчетверо меньше временных затрат рекурсивной даже при **-O3**.
- При **-O1** и **-O2** оно меньше в **шестнадцать раз**.
- При **-O0** — в 7 раз

1.11.4 "Элементарная работа с профилировщиком

Профилировщик позволяет, в частности, оценивать временны́е затраты процедур пользователя, не используя в исходном тексте программы вызов процедур типа **cpu_time**, **etime** и подобных им, нацеленных на замер времени (см. все пункты раздела **12** кроме последнего **12.7**: *«Чуть-чуть о профилировании»*

Компилятор **gfortran** имеет опцию **-pg**, активировав которую компиляцию можно провести в режиме, настроенном на использование специльной утилиты **gprof**, позволяющей на профессиональном уровне получить информацию о затратах времени. Итак, наша исходная программа теперь не пользуется вызовами подпрограмм временны́х замеров.

```
program test_my_div_gprof; use my_div; implicit none
integer, parameter :: kmax=10000
integer n, m, kn, kr, k
read (*,*) n, m; write(*,*) ' n=', n; write(*,*) ' m=', m
kn=divn( (n), m)
write(*,'(a,i7,a,i7)') ' после вызова divn(n,m):  n=',n,' kn=',kn
kr= divr(  n, m)
write(*,'(a,i7,a,i7)') ' после вызова divr(n,m):  n=',n,' kr=',kr
n=299999; m=2;  write(*,'(a,i7,a,i7)') '  n=', n,'   m=', m
do k=1,kmax; kn=divn( (n), m); enddo
do k=1,kmax; kr=divr(  n , m); enddo
write(*,'(a,i7,a,i7,a,i7,a,i7,a,i7)')   ' kn=',kn,'  kr=',kr
end
```

1. Проведём компиляцию, активировав опцию **-pg**:

   ```
   gfortran -pg my_div.f90 main.f90 -o ./main
   ```

2. Активируем исполнимый файл:

   ```
   ./main < input > respro
   ```

3. В текущей директории получаем файл **respro** с результатом:

   ```
    n=           6
    m=           2
   после вызова divn(n,m):  n=      6 kn=      3
   после вызова divr(n,m):  n=      6 kr=      3
    n= 299999   m=      2
   kn= 149999  kr= 149999
   ```

и файл с именем **gmon.out**, который своим возникновением обязан опции **-pg**. Смотреть его содержимое (как и исполнимого файла **./main**) посредством редактора не имеет смысла. **gmon.out** вспомогательный файл для дальнейшего вызова профилировщика, которому на вход следует подать наш исполнимый файл **./main** и файл **gmon.out**.

4. Вызов профилировщика:

   ```
   gprof ./main gmon.out > flat.res
   ```

   Здесь посредством операции перенаправления стандартного вывода результат направляется в файл **flat.res**. Его содержимое состоит из двух частей **flat profile** (простой профиль) и **call graph** (граф вызовов). Помимо главной содержательной информации каждая из частей содержит поясняющий комментарий к своей таблице. Остановимся пока на **flat profile**

   ```
   Flat profile:

   Each sample counts as 0.01 seconds.
     %   cumulative   self              self     total
    time   seconds   seconds    calls   s/call   s/call  name
    88.72     47.22    47.22    10001     0.00     0.00  __my_div_MOD_divr
    10.62     52.87     5.65    10001     0.00     0.00  __my_div_MOD_divn
     1.05     53.42     0.56                             frame_dummy
     0.00     53.42     0.00        1     0.00    52.87  MAIN__
   ```

   В самом правом столбце видим имя вызываемой функции, причём той самой, временные затраты которой наибольшие. Отношение временных затрат этой функции ко всему времени работы исполнимого кода приведено в самом левом столбце **88.72%**: (в процентах) Так что по сравнению с нерекурсивной функцией **divn** рекурсивная **divr** работает, грубо говоря, в восемь раз медленнее, что, вполне согласуется с оценкой, полученной без применения **gprof**. Однако, достоинство **gprof** в том, что программист не должен менять исходный текст программы, внедряя в неё дополнительные вызовы процедур оценок временных затрат.

5. Пояснения содержимого остальных столбцов **flat profile** можно найти в разделе **12.7**.

1.12 Задача 1 (СИ-решение; gcc version 4.8.5 (SUSE Linux)

1.12.1 Тестирующая программа

```
#include <stdio.h>
#include <time.h>
int divn(int, int);
int divr(int, int);
int main()
{  int const kmax=10000;  int n, m, kn, kr, k;
   long int t0, t1, t2, t3;
 scanf ("%i %i",&n, &m);   printf("n=%i  m=%i\n", n, m);
 kn=divn(n,m);             printf(" после вызова   divn(n,m):");
                           printf(" n=%i", n); printf("  kn=%i\n",kn);
 kr=divr(n,m);             printf(" после вызова    divr(n,m):");
                           printf(" n=%i", n); printf("  kr=%i\n",kr);
 n=299999; m=2;  printf(" n=%i", n); printf("   m=%i\n",m);
 t0=clock(); for (k=1;k<=kmax; k++) kn=divn(n,m);
 t1=clock(); for (k=1;k<=kmax; k++) kr=divr(n,m);
 t2=clock();
 printf("kn=%i",kn); printf("  kr=%i\n",kr);
 printf("Время на %i вызовов divn равно %le\n",kmax,(double)(t1-t0)/
                                                  CLOCKS_PER_SEC);
 printf("Время на %i вызовов divr равно %le\n",kmax,(double)(t2-t1)/
                                                  CLOCKS_PER_SEC);
 return 0;
}
```

1.12.2 Исходные тексты divn и divr

```
int divn(int n, int m)
{ int k;
  k=0;
  while (n>=m)
    {
      n=n-m; k++;
    }
  return k;
}
int divr(int n, int m)
{
  int k; if (n<m) k=0; else k=divr(n-m,m)+1; return k;
}
```

1.12.3 Вводимые данные (файл input)

```
6   2
```

1.12.4 Результат тестирования

1. **Опция -O0**

```
n=6  m=2   после вызова    divn(n,m): n=6  kn=3
           после вызова    divr(n,m): n=6  kr=3
 n=299999   m=2
kn=149999  kr=149999
Время на 10000 вызовов divn равно 3.455751e+00
Время на 10000 вызовов divr равно 3.046884e+01
```

2. **Опция -O1**

```
n=6  m=2   после вызова    divn(n,m): n=6  kn=3
           после вызова    divr(n,m): n=6  kr=3
 n=299999   m=2
kn=149999  kr=149999
Время на 10000 вызовов divn равно 9.996090e-01
Время на 10000 вызовов divr равно 7.008662e+00
```

3. **Опция -O2**

```
n=6  m=2   после вызова    divn(n,m): n=6  kn=3
           после вызова    divr(n,m): n=6  kr=3
 n=299999   m=2
kn=149999  kr=149999
Время на 10000 вызовов divn равно 9.978250e-01
Время на 10000 вызовов divr равно 9.966270e-01
```

4. **Опция -O3**

```
n=6  m=2   после вызова   divn(n,m): n=6  kn=3
           после вызова    divr(n,m): n=6  kr=3
 n=299999   m=2
kn=149999  kr=149999
Время на 10000 вызовов divn равно 3.257040e+00
Время на 10000 вызовов divr равно 3.000299e+01
```

1.12.5 Сравнение времён divn и divr на ФОРТРАНе и СИ.

	divn				**divr**			
	-O0	**-O1**	**-O2**	**-O3**	**-O0**	**-O1**	**-O2**	**-O3**
gfortran	4.6	1.1	1.0	1.0	35	16	4.16	4.5
gcc	3.5	1.0	1.0	1.0	31	7	1	1.0

Как видно, **divn**-gfortran незначительно уступает **divn**-gcc при уровнях оптимизации **-O0** и **-O1**. Однако, в случае рекурсивной **divr** компилятор **gfortran** существенно уступает **gcc**. **gcc** смог заменить рекурсию циклом даже без явного оформления хвостовой рекурсии.

1.13 Задача 2 (ФОРТРАН-решение)

Разработать нерекурсивную и рекурсивную процедуры поиска остатка от целочисленного деления одного целого на другое, моделируя операцию нахождения остатка операциями вычитания;

1.13.1 Тестирующая программа

```
program test_my_mod; use my_div; implicit none
integer, parameter :: kmax=10000
integer n, m, kn, kr, k
real t0, t1, t2
read (*,*) n, m; write(*,*) ' n=', n; write(*,*) ' m=', m
kn=modn( (n), m)
write(*,'(a,i7,a,i7)') ' после вызова modn(n,m):  n=',n,' kn=',kn
kr= modr(  n, m)
write(*,'(a,i7,a,i7)') ' после вызова modr(n,m):  n=',n,' kr=',kr
n=299999; m=2;  write(*,'(a,i7,a,i7)') '  n=', n,'   m=', m
call cpu_time(t0); do k=1,kmax; kn=modn( (n), m); enddo
call cpu_time(t1); do k=1,kmax; kr=modr(  n , m); enddo
call cpu_time(t2)
write(*,'(a,i7,a,i7,a,i7,a,i7,a,i7)')   ' kn=',kn,'  kr=',kr
write(*,*) ' Время на ',kmax,' вызовов  modn  равно',t1-t0
write(*,*) ' Время на ',kmax,' вызовов  modr  равно',t2-t1
end
```

1.13.2 Исходный текст modn и modr в кониексте модуля my_div

```
      module my_div; implicit none; contains
! ...   ...   ...   ...   ...   ...   ...   ...   ...
      function modn(n,m)
      integer modn, n, m
      do while (n.ge.m)
        n=n-m
      enddo
      modn=n
      end function modn
      recursive function modr(n,m) result(k)
      integer n, m, k
      if (n<m) then
                    k=n
               else
                    k=modr(n-m,m)
      endif
      end function modr
! ...   ...   ...   ...   ...   ...   ...   ...   ...
      end module my_div
```

1.13.3 Вводимые данные n и m

```
80  16
```

1.13.4 Вывод тестирующей программы при опции -00

```
 n=          80
 m=          16
после вызова modn(n,m):  n=     80 kn=      0
после вызова modr(n,m):  n=     80 kr=      0
 n= 299999   m=      2
kn=      1  kr=      1
 Время на        10000  вызовов  modn  равно   4.48400021
 Время на        10000  вызовов  modr  равно   34.7799988
```

1.13.5 Вывод тестирующей программы при опции -01

```
 n=          80
 m=          16
после вызова modn(n,m):  n=     80 kn=      0
после вызова modr(n,m):  n=     80 kr=      0
 n= 299999   m=      2
kn=      1  kr=      1
 Время на        10000  вызовов  modn  равно   1.49199998
 Время на        10000  вызовов  modr  равно   15.1560011
```

1.13.6 Вывод тестирующей программы при опции -02

```
 n=          80
 m=          16
после вызова modn(n,m):  n=     80 kn=      0
после вызова modr(n,m):  n=     80 kr=      0
 n= 299999   m=      2
kn=      1  kr=      1
 Время на        10000  вызовов  modn  равно  0.995999992
 Время на        10000  вызовов  modr  равно   15.7880001
```

1.13.7 Вывод тестирующей программы при опции -03

```
 n=          80
 m=          16
после вызова modn(n,m):  n=     80 kn=      0
после вызова modr(n,m):  n=     80 kr=      0
 n= 299999   m=      2
kn=      1  kr=      1
 Время на        10000  вызовов  modn  равно  0.995999992
 Время на        10000  вызовов  modr  равно   4.76400042
```

1.14 Задача 2 (СИ-решение)

1.14.1 Тестирующая программа

```
#include <stdio.h>
#include <time.h>
int modn(int, int); int modr(int, int);
int main()
{  int const kmax=10000;  int n, m, kn, kr, k;
   long int t0, t1, t2, t3;
 scanf ("%i %i",&n, &m);   printf("n=%i  m=%i\n", n, m);
 kn=modn(n,m);             printf(" после вызова   modn(n,m):");
                           printf(" n=%i", n); printf("  kn=%i\n",kn);
 kr=modr(n,m);             printf(" после вызова    modr(n,m):");
                           printf(" n=%i", n); printf("  kr=%i\n",kr);
 n=299999; m=2;  printf(" n=%i", n); printf("   m=%i\n",m);
 t0=clock(); for (k=1;k<=kmax; k++) kn=modn(n,m);
 t1=clock(); for (k=1;k<=kmax; k++) kr=modr(n,m); t2=clock();
 printf("kn=%i",kn); printf("  kr=%i\n",kr);
 printf("Время на %i вызовов modn равно %le\n",kmax,(double)(t1-t0)/
                                                  CLOCKS_PER_SEC);
 printf("Время на %i вызовов modr равно %le\n",kmax,(double)(t2-t1)/
                                                  CLOCKS_PER_SEC);
 return 0;
}
```

1.14.2 Исходный текст modn и modr

```
int modn(int n, int m) {
                         while (n>=m) n=n-m; return n;
                       }
int modr(int n, int m) {
                         if (n<m) return n; else return modr(n-m,m);
                       }
```

1.14.3 Вводимые данные n и m

```
6   2
```

1.14.4 Вывод тестирующей программы при опции -00

```
n=6  m=2
 после вызова   modn(n,m): n=6  kn=0
 после вызова    modr(n,m): n=6  kr=0
 n=299999   m=2
kn=1  kr=1
Время на 10000 вызовов modn равно 3.976954e+00
Время на 10000 вызовов modr равно 1.972354e+01
```

1.14.5 Вывод тестирующей программы при опции -01

```
n=6  m=2
 после вызова   modn(n,m): n=6  kn=0
 после вызова    modr(n,m): n=6  kr=0
 n=299999   m=2
kn=1  kr=1
Время на 10000 вызовов modn равно 9.965920e-01
Время на 10000 вызовов modr равно 7.556358e+00
```

1.14.6 Вывод тестирующей программы при опции -02

```
n=6  m=2
 после вызова   modn(n,m): n=6  kn=0
 после вызова    modr(n,m): n=6  kr=0
 n=299999   m=2
kn=1  kr=1
Время на 10000 вызовов modn равно 9.965950e-01
Время на 10000 вызовов modr равно 9.965590e-01
```

1.14.7 Вывод тестирующей программы при опции -03

```
n=6  m=2
 после вызова   modn(n,m): n=6  kn=0
 после вызова    modr(n,m): n=6  kr=0
 n=299999   m=2
kn=1  kr=1
Время на 10000 вызовов modn равно 9.965140e-01
Время на 10000 вызовов modr равно 9.964880e-01
```

1.14.8 Сравнение времён modn и modr на ФОРТРАНе и СИ.

	modr				**modn**			
	-O0	**-O1**	**-O2**	**-O3**	**-O0**	**-O1**	**-O2**	**-O3**
gfortran	4.5	1.5	1.0	1.0	35	15	16	4.8
gcc	4.0	1.0	1.0	1.0	20	1	1	1.0

Как видно, **modn**-gfortran незначительно уступает **modn**-gcc при уровнях оптимизации **-O0** и **-O1**. Однако, в случае рекурсивной **divr** компилятор **gfortran** существенно уступает **gcc**. **gcc** смог заменить рекурсию циклом даже без явного оформления хвостовой рекурсии.

1.15 Задача 3 (ФОРТРАН-решение)

Разработать нерекурсивную и рекурсивную процедуры перевода введенного целого в двоичную систему счисления.

1.15.1 Некоторое уяснение ситуации

1. Под результат отведём вектор с индексацией элементов от 0 до 31 (*Почему?*).

2. Опишем две нерекурсивные процедуры **decbinn1** и **decbinn2** с одинаковым интерфейсом:

```
subroutine decbinn1(n,nb,ii); integer  num; integer nb(0:31);
                                    integer ii;
end subroutine decbinn1
subroutine decbinn2(n,nb,ii); integer  num; integer nb(0:31);
                                    integer ii;
end subroutine decbinn2
```

   Здесь **num** — исходное неотрицательное целое; **nb** — вектор из 32 элементов. Предполагается, что элемент с нулевым индексом после работы подпрограммы должен будет хранить разряд двоичных единиц. **ii** — индекс элемента (определяется в процессе перевода), хранящего старший двоичный разряд числа **num**.

3. Различие между **decbinn1** и **decbinn2** в организации способа передачи аргумента **num**.

   Процедура **decbinn1** (в отличие от **decbinn2**) всегда сохраняет значение своего первого аргумента за счёт копирования его в дополнительную рабочую переменную, значение которой последующая часть алгоритма перевода и меняет (имя первого аргумента попросту не упоминается в алгоритме перевода).

   Алгоритм **decbin2** при вызове **call decbinn2(n,nb,ii)** изменяет значение переменной **n** и, в конечном итоге, обнулит его (*Почему?*). При вызове же **call decbinn2((n) ,nb,ii)** обнуления не произойдёт, поскольку заключение аргумента в скобки трактуется как выражение, которое надо вычислить, и результат расчёта помещается не по адресу фактического аргумента, так что значение последнего не меняется.

1.15.2 Тестирующая программа

```
program test_dec_bin; use my_div; implicit none
integer, parameter :: kmax=10000000
integer n, nn1(0:31), nn2(0:31), nr(0:31), i, ii1, ii2, iir, k;
real t0, t1, t2, t3
read (*,*) n; write(*,*) ' n=', n
call decbinn1( n ,nn1,ii1); write(*,'(15x," decbinn1: nn1=",32i1)')&
     &                                          (nn1(i),i=31,0,-1)
call decbinn2((n),nn2,ii2); write(*,'(15x," decbinn2: nn2=",32i1)')&
     &                                          (nn2(i),i=31,0,-1)
iir=0; call  decbinr( n , nr,iir); write(*,'(15x,"  decbinr:  nr=",32i1)')&
     &                                          (nr(i),i=31,0,-1)
write(*,'("  Вывод n по формату b32:   n=",b32)') n
write(*,'("  ii1=",i5,"  ii2=",i5,"  iir=",i5)') ii1, ii2, iir
call cpu_time(t0); do k=1,kmax; call decbinn1(  n ,nn1,ii1); enddo
call cpu_time(t1); do k=1,kmax; call decbinn2( (n) ,nn1,ii2); enddo
call cpu_time(t2); do k=1,kmax; iir=0; call decbinr( n ,nr,iir); enddo
call cpu_time(t3)
write(*,*) ' Время на ',kmax,' вызовов  decbinn1 равно',t1-t0
write(*,*) ' Время на ',kmax,' вызовов  decbinn2 равно',t2-t1
write(*,*) ' Время на ',kmax,' вызовов   decbinr равно',t3-t2
end
```

1.15.3 Тексты decbinn1, decbinn2 и decbinr в контексте модуля my_div

```
module my_div; implicit none;  contains ! ...   ...   ...
subroutine decbinn1(num,nb,k)
integer num, n, nb(0:31), i, k
nb=0;   n=num;  i=0
do while (n/=0); nb(i)=mod(n,2); n=n/2; i=i+1; enddo; k=i-1
end subroutine decbinn1
subroutine decbinn2(n,nb,k)
integer n, nb(0:31), i, k
nb=0; i=0;
do while (n/=0); nb(i)=mod(n,2); n=n/2; i=i+1; enddo; k=i-1
end subroutine decbinn2
recursive subroutine decbinr(n,nb,k)
integer n, nb(0:31),  k
if (n==0) then; nb(k)=0; if (k>0) k=k-1
          else; nb(k)=mod(n,2); k=k+1
                call decbinr(n/2,nb,k)
endif
end subroutine decbinr
end module my_div
```

1.15.4 Вводимые данные n и m

```
2147483647
```

1.15.5 Вывод тестирующей программы при опции -00

```
n=  2147483647
              decbinn1: nn1=01111111111111111111111111111111
              decbinn2: nn2=01111111111111111111111111111111
               decbinr:  nr=01111111111111111111111111111111
Вывод n по формату b32:   n= 11111111111111111111111111111111
ii1=   30  ii2=   30  iir=   30
Время на     10000000  вызовов  decbinn1 равно   1.59599996
Время на     10000000  вызовов  decbinn2 равно   2.08400011
Время на     10000000  вызовов   decbinr равно   3.22399974
```

1.15.6 Вывод тестирующей программы при опции -01

```
n=  2147483647
              decbinn1: nn1=01111111111111111111111111111111
              decbinn2: nn2=01111111111111111111111111111111
               decbinr:  nr=01111111111111111111111111111111
Вывод n по формату b32:   n= 11111111111111111111111111111111
ii1=   30  ii2=   30  iir=   30
Время на     10000000  вызовов  decbinn1 равно  0.828000009
Время на     10000000  вызовов  decbinn2 равно  0.859999955
Время на     10000000  вызовов   decbinr равно   1.18799996
```

1.15.7 Вывод тестирующей программы при опции -02

```
n=  2147483647
              decbinn1: nn1=01111111111111111111111111111111
              decbinn2: nn2=01111111111111111111111111111111
               decbinr:  nr=01111111111111111111111111111111
Вывод n по формату b32:   n= 11111111111111111111111111111111
ii1=   30  ii2=   30  iir=   30
Время на     10000000  вызовов  decbinn1 равно  0.736000001
Время на     10000000  вызовов  decbinn2 равно  0.716000021
Время на     10000000  вызовов   decbinr равно   1.07199991
```

1.16 Задача 3 (СИ-решение)

1.16.1 Тестирующая программа

```
#include <stdio.h>
#include <time.h>
void decbinn(int, int*, int*); void decbinr(int, int*,int*);
int main()
{  int const kmax=10000000;  int n, nn[32], nr[32], iin, iir, k, i;
   long int t0, t1, t2;
 scanf ("%i",&n);    printf("n=%i\n", n);
 decbinn(n,nn,&iin); printf(" decbinn:"); printf("  iin=%i",iin);
 printf("  nn="); for (k=31;k>=0;k--) printf("%1i",nn[k]); printf("\n");
 iir=0; for (k=0;k<=31;k++) nr[k]=0;
 decbinr(n,nr,&iir); printf(" decbinr:"); printf("  iir=%i",iir);
 printf("  nr="); for (k=31;k>=0;k--) printf("%1i",nr[k]); printf("\n");
 t0=clock(); for (k=1;k<=kmax; k++) decbinn(n,nn,&iin);
 t1=clock(); for (k=1;k<=kmax; k++)
               {  iir=0; for (i=0;i<=31;i++) nr[i]=0;
                  decbinr(n,nr,&iir);
               }
 t2=clock();
 printf("Время на %i вызовов decbinn равно %le\n",kmax,(double)(t1-t0)/
                                                    CLOCKS_PER_SEC);
 printf("Время на %i вызовов decbinr равно %le\n",kmax,(double)(t2-t1)/
                                                    CLOCKS_PER_SEC);
 return 0;
}
```

1.16.2 Исходный текст modn и modr

```
void decbinn(int n, int* nb, int* k)
{
  int i; for (i=0;i<=31;i++) nb[i]=0;
  i=0;   for (;n!=0;) {nb[i]=n%2; n=n/2; i++;} *k=i-1;
}
void decbinr(int n, int* nb, int *k)
{
  if (n==0) { nb[*k]=0; if (*k>0) *k=*k-1;}
     else   { nb[*k]=n%2; (*k)=*k+1; decbinr(n/2,nb,k); }
}
```

1.16.3 Вводимые данные n и m

```
2147483647
```

1.16.4 Вывод тестирующей программы при опции -00

```
n=2147483647
 decbinn:  iin=30  nn=01111111111111111111111111111111
 decbinr:  iir=30  nr=01111111111111111111111111111111
Время на 10000000 вызовов decbinn равно 3.261134e+00
Время на 10000000 вызовов decbinr равно 4.295999e+00
```

1.16.5 Вывод тестирующей программы при опции -01

```
n=2147483647
 decbinn:  iin=30  nn=01111111111111111111111111111111
 decbinr:  iir=30  nr=01111111111111111111111111111111
Время на 10000000 вызовов decbinn равно 1.062754e+00
Время на 10000000 вызовов decbinr равно 1.813498e+00
```

1.16.6 Вывод тестирующей программы при опции -02

```
n=2147483647
 decbinn:  iin=30  nn=01111111111111111111111111111111
 decbinr:  iir=30  nr=01111111111111111111111111111111
Время на 10000000 вызовов decbinn равно 9.499500e-01
Время на 10000000 вызовов decbinr равно 9.507380e-01
```

1.16.7 Вывод тестирующей программы при опции -03

```
n=2147483647
 decbinn:  iin=30  nn=01111111111111111111111111111111
 decbinr:  iir=30  nr=01111111111111111111111111111111
Время на 10000000 вызовов decbinn равно 7.971660e-01
Время на 10000000 вызовов decbinr равно 7.904890e-01
```

1.16.8 Сравнение времён decbinn и decbinr на gfortran и gcc.

	decbinn				**decbinr**			
	-O0	**-O1**	**-O2**	**-O3**	**-O0**	**-O1**	**-O2**	**-O3**
gfortran	2.0	0.8	0.7	0.7	3.2	1.3	1.1	1.1
gcc	3.3	1.1	1.0	0.8	4.3	1.8	0.95	0.8

Как видно, в случае нерекурсивной функции **gfortran** быстрее **gcc**. При уровнях оптимизации **-O0** и **-O1** и в рекурсивной. Однако, при **-O2** и **-O3** в случае рекурсии **gfortran** уступает **gcc**.

1.17 Задача 4 (ФОРТРАН-решение)

Разработать нерекурсивную и рекурсивную процедуры возведения основания в неотрицательную целочисленную степень, моделируя операцию возведения в степень операциями умножения.

Рассмотрены несколько вариантов реализации процедур:

1. **fpow0n** — непосредственное использование ФОРТРАН-операции ** с целью оценки её временных затрат.
2. **fpow1n** — моделирование операции $\mathbf{a^n}$ через **n** умножений.
3. **fpow2n** — моделирование операции $\mathbf{a^n}$ на основе формулы
$$\mathbf{a^{2n+1} = a \cdot a^{2n}}; \qquad \mathbf{a^{2n} = (a^2)^n}$$
4. **fpow1r** — рекурсивная модель функции **fpow1n**
5. **fpow2r** — вариант рекурсии **fpow2n** (рекурсивное домножение на основание при проверке четности показателя по **if**).
6. **fpow3r** — вариант рекурсии **fpow2n** (то же, что и **fpow2r**, но проверка чётности показателя по **do while**).
7. **fpow4r** — вариант рекурсии **fpow2n**
8. **fpowrx** — возможно хвостовая рекурсия **fpow1r**.
9. **spowr1** — рекурсия **fpow1r** через подпрограмму.

1.17.1 Makefile

```
 comp:=gfortran
  opt:=-c -O2
  pattern:=*.f90
  source :=$(wildcard $(pattern))
  obj    :=$(patsubst %.f90, %.o, $(source))
  main:=main
  $(main) : $(obj)
$(comp) $^ -o $@
 %.o %.mod : %.f90
           $(comp) $(opt) $<
 $(main).o : my_div.mod my_prec.mod
    result : $(main) input
       time ./$<  < input > result
clear :
rm -f *.o *.mod
```

1.17.2 Тестирующая программа

```
program test_4
use my_prec
use my_div
implicit none
integer, parameter :: kmax=10000000
character(6), parameter :: text(0:8)=(/'fpow0n','fpow1n','fpow2n',&
&                                      'fpow1r','fpow2r','fpow3r',&
&                                      'fpow4r','fpowrx','spowr1' /)
real(mp) a, r(0:8)
real        t(0:9)
integer n, k
read (*,*) a, n
write(*,*) ' a=', a, ' n=',n;
call second(t(0)); do k=1,kmax; r(0)=fpow0n(a,n); enddo ! 0)  a**n.
                                                        !    НЕРЕКУРСИВНЫЕ:
call second(t(1)); do k=1,kmax; r(1)=fpow1n(a,n); enddo ! 1) медленная
call second(t(2)); do k=1,kmax; r(2)=fpow2n(a,n); enddo ! 2) быстрая
                                                        !     РЕКУРСИВНЫЕ:
call second(t(3)); do k=1,kmax; r(3)=fpow1r(a,n); enddo ! 1)
call second(t(4)); do k=1,kmax; r(4)=fpow2r(a,n); enddo ! 2) if
call second(t(5)); do k=1,kmax; r(5)=fpow3r(a,n); enddo ! 2) do while
call second(t(6)); do k=1,kmax; r(6)=fpow4r(a,n); enddo ! 2)  полностью.
call second(t(7)); do k=1,kmax; r(7)=fpowrx(a,n); enddo ! 1) хвостовая?
call second(t(8)); do k=1,kmax; r(8)=1.0_mp;            ! 1) рекурсивная
                                call spowr1(a,n,r(8))   !    подпрограмма
                   enddo
call second(t(9))
write(*,'(e10.3,"**",i5,"=",e25.16)') a,n,r(0)
write(*,'(5x,"Число вызовов kmax=",i10)') kmax
write(*,'("Функция",6x,"Отн.погр.",3x,"Время")')
write(*,'(a,e16.3,f8.3)') (text(k),abs((r(k)-r(0))/r(0)),t(k+1)-t(k),k=0,8)
end
```

В четвёртой задаче главная программа (как и модуль **my_div**) использует модуль **my_prec**, в котором задана константа **mp**, указывающая число байт для представления данного вещественного типа. Таким данным в этой задаче является основание. Представляется интересным сравнить временные затраты алогритмов возведения в степень на одинарной, удвоенной и четверной точности. Использование модуля **my_prec** обеспечивает простоту смены разновидности вещественного типа, не требуя знания соответствующих опций компилятора.

1.17.3 Исходные текст процедур возведения в степень (модуль my_div)

```
module my_div; use my_prec; implicit none; contains !   ...   ...
function fpow0n(aa,nn)
real(mp) fpow0n, aa
integer nn
fpow0n=aa**nn
end function fpow0n
function fpow1n(aa,nn)                          ! Нерекурсивное
real(mp) fpow1n, aa, a, p; integer nn, n, i     !     долгое
a=aa; n=nn; p=1.0_mp                            ! возведение  aa**nn
do i=1,n; p=p*a; enddo                          ! за  nn  умножений.
fpow1n=p
end function fpow1n
function fpow2n(aa,nn)                          ! Нерекурсивное
real(mp) fpow2n, aa, a, p; integer nn, n        !     быстрое
a=aa; n=nn; p=1.0_mp                            ! возведение aa**nn
do while (n/=0)
  do while(mod(n,2)==0)
    a=a*a
    n=n/2
  enddo
  p=p*a; n=n-1
enddo
fpow2n=p
end function fpow2n
recursive function fpow1r(a,n) result(p)     ! Рекурсивное
real(mp) a, p                                !  (медленное)
integer n, i                                 ! возведение aa**nn
if (n==0) then; p=1.0_mp                     ! за nn умножений
          else; p=fpow1r(a,n-1)*a
endif
end function fpow1r
recursive function fpowrt(a,n,res) result(p) ! Хвостовая рекурсия
real(mp) a, p, res                           !     aa**nn
integer n, i                                 ! за nn умножений
if (n==0) then; p=res                        !  Ф У Н К Ц И Я:
          else; p=fpowrt(a,n-1,res*a)
endif
end function fpowrt

real(mp) function fpowrx(aa,nn) result (w)
real(mp) aa, a
integer nn, n
a=aa; n=nn
w=fpowrt(a,n,1.0_mp)
end function fpowrx
```

```
      recursive subroutine spowr1(a,n,res)
      real(mp) a, res
      integer  n
      if (n/=0) then; res=res*a
                      call spowr1(a,n-1,res)
      endif
      end subroutine spowr1
      recursive function fpow2r(aa,nn) result(p)
      real(mp) aa, a, p
      integer nn, n
      a=aa; n=nn
      if (n==0) then; p=1.0_mp
                else; if (mod(n,2)==0) then
                           a=a*a
                           n=n/2
                      endif
                p=fpow2r(a,n-1)*a
      endif
      end function fpow2r
      recursive function fpow3r(aa,nn) result(p)
      real(mp) aa, a, p
      integer nn, n
      a=aa; n=nn
      if (n==0) then; p=1.0_mp
                else; do while (mod(n,2)==0)
                        a=a*a
                        n=n/2
                      enddo
                      p=fpow3r(a,n-1)*a
      endif
      end function fpow3r
!
      recursive function fpow4r(a,n) result(p)
      real(mp) aa, a, p
      integer nn, n
      if (n==0) then; p=1.0_mp
                else
                  if (mod(n,2)==0) then
                     p=fpow4r(a*a,n/2)
                   else
                     p=fpow4r(a,n-1)*a
                  endif
      endif
      end function fpow4r
      end module my_div
```

1.17.4 Вводимые данные n и m

```
2.0 33
```

1.17.5 Вывод тестирующей программы при опции -00

```
  a=   2.0000000000000000       n=          33
 0.200E+01**   33=   0.8589934592000000E+10
     Число вызовов kmax=  10000000
Функция      Отн.погр.   Время
fpow0n       0.000E+00   0.152
fpow1n       0.000E+00   1.556
fpow2n       0.000E+00   0.336
fpow1r       0.000E+00   4.048
fpow2r       0.000E+00   0.772
fpow3r       0.000E+00   0.436
fpow4r       0.000E+00   0.860
fpowrx       0.000E+00   4.232
spowr1       0.000E+00   2.576
```

1.17.6 Сравнение временных затрат real(4), real(8), real(10) и real(16)

Оптимизация:	**-O0**				**-O1**			
Функция	**4**	**8**	**10**	**16**	**4**	**8**	**10**	**16**
fpow0n	0.156	0.152	0.224	2.308	0.080	0.080	0.168	2.268
fpow1n	1.576	1.556	3.108	10.116	0.508	0.508	0.572	9.832
fpow2n	0.340	0.336	0.736	2.264	0.104	0.112	0.128	2.264
fpow1r	4.016	4.048	6.492	12.980	1.288	1.144	1.820	10.840
fpow2r	0.768	0.772	1.448	3.508	0.284	0.288	0.704	3.148
fpow3r	0.432	0.436	0.836	2.328	0.164	0.152	0.340	2.144
fpow4r	0.884	0.860	1.488	2.684	0.324	0.332	0.576	2.288
fpowrx	3.864	4.232	7.496	11.912	2.080	1.836	3.580	11.696
spow1r	2.380	2.576	3.564	11.788	1.868	1.832	3.436	11.776

Оптимизация:	**-O2**				**-O3**			
Функция	**4**	**8**	**10**	**16**	**4**	**8**	**10**	**16**
fpow0n	0.080	0.080	0.164	2.260	0.080	0.084	0.164	2.244
fpow1n	0.500	0.500	0.552	9.828	0.504	0.504	0.552	9.832
fpow2n	0.112	0.104	0.124	2.160	0.096	0.100	0.124	2.160
fpow1r	1.476	1.476	1.752	11.080	0.864	0.860	0.768	10.240
fpow2r	0.308	0.312	0.752	3.120	0.172	0.180	0.344	3.072
fpow3r	0.156	0.160	0.324	2.156	0.100	0.116	0.120	2.152
fpow4r	0.304	0.304	0.556	2.268	0.196	0.204	0.400	2.196
fpowrx	1.968	1.912	3.548	10.384	0.748	0.796	1.396	9.748
spow1r	1.984	1.884	3.508	11.056	1.068	1.068	1.544	10.268

1.18 Задача 4 (СИ-решение)

1.18.1 Тестирующая программа

```
#include <stdio.h>
#include <math.h>
#include <time.h>
double cpow0n(double, int); double cpow1n(double, int);
double cpow2n(double, int);
double cpow1r(double, int); double cpow2r(double, int);
double cpow3r(double, int); double cpow4r(double, int);
double cpowrx(double, int); double cpowrt(double, int, double);
void vpow1r(double, int, double*);
int main()
{ int const kmax=10000000;
 char text[9][7]={"cpow0n\0","cpow1n","cpow2n",
     "cpow1r","cpow2r","cpow3n",
     "cpow4r","cpowrx","vpowr1"};
  double r[9];
  long int t[10];
  double a;
  int n, k;
  scanf ("%le %i", &a, &n);
  printf(" a=%15.7le   n=%4i\n", a, n);
 t[0]=clock(); for (k=1;k<=kmax; k++) r[0]=cpow0n(a,n);   // НЕРЕКУРСИВНЫЕ
 t[1]=clock(); for (k=1;k<=kmax; k++) r[1]=cpow1n(a,n);
 t[2]=clock(); for (k=1;k<=kmax; k++) r[2]=cpow2n(a,n);
 t[3]=clock(); for (k=1;k<=kmax; k++) r[3]=cpow1r(a,n);
 t[4]=clock(); for (k=1;k<=kmax; k++) r[4]=cpow2r(a,n);
 t[5]=clock(); for (k=1;k<=kmax; k++) r[5]=cpow3r(a,n);
 t[6]=clock(); for (k=1;k<=kmax; k++) r[6]=cpow4r(a,n);
 t[7]=clock(); for (k=1;k<=kmax; k++) r[7]=cpowrx(a,n);
 printf(" a=%15.7le   n=%4i\n", a, n);
 t[8]=clock(); for (k=1;k<=kmax; k++)
                  {r[8]=1.0;
                   vpow1r(a,n,&r[8]);}
 t[9]=clock();
  printf("pppr[8]=%le\n",r[8]);
  printf("pow(%le , %5i) = %25.16le\n",a,n,r[0]);
  printf("%5sЧисло вызовов kmax=%10i\n"," ", kmax);
  printf("Функция %1s Отн.погр. %2s Время\n"," "," ");
  for (k=0;k<=8;k++)
    {
      printf("%s %12.3le %8.3lf\n",text[k],fabs((r[k]-r[0])/r[0]),
      (double)(t[k+1]-t[k])/CLOCKS_PER_SEC);
    }
 return 0;
}
```

1.18.2 Исходные тексты процедур a^n

```
#include <math.h>
double cpow0n(double a, int n)   // pow (from math.h)
{ return (pow(a,n)); }

double cpow1n( double a, int n)  // Нерекурсивное за n умножений:
{ double p; int i;; p=1.0;
  for (i=1; i<=n; i++) p*=a; return p;
}
double cpow2n(double a, int n)   // Нерекурсивное быстрое
{ double p; p=1.0;
  for (; n!=0;) { for (; n%2==0;) { a*=a; n/=2; }
                  p*=a; n-=1;
                }
  return p;
}
double cpow1r( double a, int n)  // Рекурсивное за n умножений:
{
  double p; int i;  return (n==0 ? 1.0 : cpow1r(a,n-1)*a);
}
double cpow2r(double a, int n)   //  Рекурсивное быстрое
{
  if (n==0) return 1.0;  else { if (n%2==0) { a*=a; n/=2;}
                                return cpow2r(a,--n)*a;
                              }
}
double cpow3r(double a, int n)
{
  if (n==0) return 1.0; else { if (n%2==0) { a*=a; n/=2;}
                                 return cpow3r(a,--n)*a;
                               }
}
double cpow4r(double a, int n)
{
  if (n==0) return 1.0; else { if (n%2==0)  return cpow4r(a*a,n/2);
                           else       return cpow4r(a,(n-1))*a;
                         }
}
double cpowrt(double a, int n, double res)     // Хвостовая рекурсия
{ if (n==0) return res;                        //  Ф У Н К Ц И Я:
    else; return cpowrt(a,n-1,res*a);
}
double cpowrx(double a,int n)
{ return cpowrt(a,n,1.0); }

void vpow1r(double a, int n, double *res)
{ if (n!=0)  {*res=*res*a; vpow1r(a,--n,res);} }
```

1.18.3 Вводимые данные n и m

```
2.0 33
```

1.18.4 Вывод тестирующей программы при опции -00

```
 a=  2.0000000e+00   n=  33
 a=  2.0000000e+00   n=  33
pppr[8]=8.589935e+09
pow(2.000000e+00 ,    33) =    8.5899345920000000e+09
     Число вызовов kmax=  10000000
Функция   Отн.погр.    Время
cpow0n    0.000e+00    1.034
cpow1n    0.000e+00    1.622
cpow2n    0.000e+00    0.315
cpow1r    0.000e+00    3.575
cpow2r    0.000e+00    0.764
cpow3n    0.000e+00    0.774
cpow4r    0.000e+00    0.660
cpowrx    0.000e+00    3.223
vpowr1    0.000e+00    2.662
```

1.18.5 Сравнение затрат времени gfortran и gcc.

	-O0		**-O1**		**-O2**		**-O3**	
Функция	**gfortran**	**gcc**	**gfortran**	**gcc**	**gfortran**	**gcc**	**gfortran**	**gcc**
pow0n	0.152	1.034	0.080	1.019	0.080	0.983	0.084	0.973
pow1n	1.556	1.622	0.508	0.502	0.500	0.514	0.504	0.495
pow2n	0.336	0.315	0.112	0.117	0.104	0.099	0.100	0.107
pow1r	4.048	3.575	1.144	1.243	1.476	1.060	0.860	0.757
pow2r	0.772	0.764	0.288	0.293	0.312	0.207	0.180	0.143
pow3r	0.436	0.774	0.152	0.355	0.160	0.208	0.116	0.141
pow4r	0.860	0.660	0.332	0.334	0.304	0.167	0.204	0.103
powrx	4.232	3.223	1.836	1.132	1.912	0.490	0.796	0.488
spow1r	2.576	2.662	1.832	1.953	1.884	0.513	1.068	0.515

- Время вызова и работы СИ-функции **pow** больше затрат операции **, возведения в степень, ФОРТРАНа в десять раз.
- В случае нерекурсивных функций **gfortran** и **gcc** примерно одинаковы по затратам времени.
- Хвостовая рекурсия **gcc** существенно сокращает затраты времени по сравнению с **gfortran**-ом.

1.19 Задача 5 (ФОРТРАН-решение)

Разработать нерекурсивную и рекурсивную процедуры перевода введенного неотрицательного вещественного числа меньшего единицы в двоичную систему счисления.

```
program test_5; use my_prec;  use my_rebin; implicit none
integer kbin(130)
integer i, nbit; character(3) sn; character(30) sf
real(mp) x
write(*,*) ' mp=',mp
read(*,*) x
write(*,*) ' x=',x
call rebinn(x, kbin, nbit)
write(*,*) ' rebinN: Реальное число двоичных цифр... (nbit)=',nbit
write(sn,'(i3)') nbit
write(*,*) ' sn=',sn
sf='(a,'//trim(sn)//'(i1))'
write(*,*) ' sf=',trim(sf)
write(*,trim(sf))    '  rebinN: kbin= ',(kbin(i),i=1,nbit)
!write(*,'(a,130i1)') '  rebinN: kbin= ',(kbin(i),i=1,nbit)
!write(*,'(a$)') '  rebinN: kbin= '
!do i=1,nbit
!  write(*,'(i1$)') kbin(i)
!enddo
! write(*,*)
kbin=0; nbit=1
write(*,*) ' x=',x
call rebinr(x, kbin, nbit)
nbit=nbit-1
write(*,*) ' rebinR: Реальное число двоичных цифр... (nbit)=',nbit
write(*,trim(sf)) '  rebinR: kbin= ',(kbin(i),i=1,nbit)
end program test_5
```

1. Программа **test_5** использует модули **my_prec** и **my_rebin**. Первый реализует возможность быстрого перехода на иную допустимую разновидность типа **real**; второй содержит описания требуемых процедур: **rebinn** (нерекурсивная) и **rebinr** (рекурсивная).
2. Вектор **kbin(130)** используется программой в качестве фактического аргумента обеих процедур и предназначен для приёмки и хранения результата работы обеих процедур. Для простоты предполагается, что результат нужно получить в форме с фиксированной запятой, т.е. элемент **kbin(k)** после работы любой из процедур должен хранить двоичную цифру, соответствующую весу $\mathbf{2^{-k}}$.

3. Число **130** в качестве наибольшего индекса вектора **kbin** выбрано с запасом из расчёта работы со значениями типа **real(16)**. для которого запись двоичной мантиссы числа в форме с плавающей запятой предоставляет 113 бит.

4. Переменные **sn** и **sf** (типа **character(3)** и **character(30)** соответственно) используются программой для организации динамического повторителя в форматной строке вывода содержимого вектора **kbin**. Если **mp=4**, то максимальное количество двоичных цифр в мантиссе числа **23**; если **mp=8**, то — 53; при **mp=10, 16** — 113. Сначала переменная **sn** используется в качестве внутреннего файла, в который выводится значение **nbit**, найденное процедурами, а затем в переменной **sf** формируется формат вывода ровно **nbit** цифр, которые хранятся в первых **nbit** элементах вектора **kbin**. В принципе, можно было бы и не использовать строковые переменные, обойдясь, например, форматом:

```
write(*,'(a,130i1)') '  rebinN: kbin= ',(kbin(i),i=1,nbit)   !  или

write(*,'(a$)') '  rebinN: kbin= '                            !  так:
do i=1,nbit;  write(*,'(i1$)') kbin(i); enddo
```

   Посредством **sn** и **sf** просто продемонстрирована возможность организации динамического повторителя. ФОРТРАН-функция **trim** исключает хвостовые пробелы строковой переменной.

5. После вывода результата работы **rebinn** обнуляется содержимое вектора **kbin**, подготавливая его к приёмке результата **rebinr**, и инициализируется индекс **nbit** элемента, в который должна быть занесена двоичная цифра, соответствующая разряду с весом $\mathbf{2^{-1}}$. Далее следует вызов рекурсивной подпрограммы и вывод её результата.

1.19.1 Модуль my_prec

```
      module my_prec; implicit none; integer, parameter ::  mp=4
      end module my_prec
```

1.19.2 Модуль my_rebin

```
module my_rebin; use my_prec; implicit none; contains
subroutine rebinn(xx,kbin,n)
integer n, i; real(mp) xx, x; integer kbin(130)
x=xx; kbin=0; n=1
do while (x/=0.0_mp);  x=2*x; kbin(n)=int(x); x=x-kbin(n); n=n+1; enddo
n=n-1
end subroutine rebinn
recursive subroutine rebinr(x,kbin,n)
integer n, kbin(130); real(mp) x
if (x==0.0_mp) then; kbin(n)=0
               else; x=2*x; kbin(n)=int(x); x=x-kbin(n); n=n+1;
                     call rebinr(x,kbin,n)
endif
end subroutine rebinr
end module my_rebin
```

1.19.3 Результаты работы my_test5

```
 x=  0.123456791
 rebinN: Реальное число двоичных цифр... (nbit)=          26
 sn= 26
 sf=(a, 26(i1))
 rebinN: kbin= 00011111100110101101110101
 x=  0.123456791
 rebinR: Реальное число двоичных цифр... (nbit)=          26
 rebinR: kbin= 00011111100110101101110101

 mp=           8
 x=  0.12345678901234568
 rebinN: Реальное число двоичных цифр... (nbit)=          56
 sn= 56
 sf=(a, 56(i1))
 rebinN: kbin= 00011111100110101101110100110111010001101111011001011111
 x=  0.12345678901234568
 rebinR: Реальное число двоичных цифр... (nbit)=          56
 rebinR: kbin= 00011111100110101101110100110111010001101111011001011111
```

```
mp=          10
x=  0.12345678901234567890
rebinN: Реальное число двоичных цифр... (nbit)=          66
sn= 66
sf=(a, 66(i1))
rebinN: kbin= 000111111001101011011101001101110100011011110110010111110001110001
x=  0.12345678901234567890
rebinR: Реальное число двоичных цифр... (nbit)=          66
rebinR: kbin= 000111111001101011011101001101110100011011110110010111110001110001

mp=          16
x=  0.12345678901234567890123456789012345678901234567890123462
rebinN: Реальное число двоичных цифр... (nbit)=         116
sn=116
sf=(a,116(i1))
rebinN: kbin= 000111111001101011011101001101110100011011110110010111110001110001
          000111000011111110010110100010101011110111110001010101011111
x=  0.12345678901234567890123456789012345678901234567890123462
rebinR: Реальное число двоичных цифр... (nbit)=         116
rebinR: kbin= 000111111001101011011101001101110100011011110110010111110001110001
          000111000011111110010110100010101011110111110001010101011111
```

1.20 Задача 6 (ФОРТРАН-решение)

Разработать нерекурсивную и рекурсивную процедуры, которые выясняют является ли заданное натуральное число простым. Вспомним задачу **№6** из домашнего задания **№4** первого семестра, в которой разрабатывался этот же алгоритм в виде главной программы:

```
      program h4_6; implicit none
      character(9), parameter :: text(0:1)=(/'not prime','  prime  '/)
      integer n, k, j
      logical prime
      do; read(*,*,end=77) n
          k=3
          prime=(n==2).or.(mod(n,2)==1)
          do while (k*k<=n.and.prime)
            prime=(mod(n,k)/=0)
            k=k+2
          enddo
          j=0; if (prime) j=1
          write(*,'(i10,a10)') n,text(j)
      enddo
   77 continue
      end
```

Оформим алгоритм анализа числа нерекурсивной и рекурсивной процедурами, причём (для тренировки) каждую и в виде функции (**fprimen**, **fprimer**), и в виде подпрограммы (**sprimen**, **sprimer**). Поместим описания этих процедур в модуль **my_prime**. Каждая из них будет возвращать в вызывающую программу значение типа **logical** (функции — через своё имя; подпрограммы через дополнительный аргумент для результата). Для вывода в качестве результата не булевых значений, а слов **простое** и **составное**, можно воспользоваться условным оператором:

```
if (fprimen(n)) then; write(*,*) '  простое  '
                else; write(*,*) ' составное '
endif
```

Более элегантное оформление требуемого вывода — размещение слов **простое** и **составное** в элементах двухэлементного массива, например, **txt(0:1)** типа **character(11)**. Остаётся лишь уточнить, как значению **.false.** сопоставить нулевой элемент, а значению **.true.** — первый (ведь ФОРТРАН в качестве индекса не допускает значения булева типа. Конечно, можно опять-таки использовать условный оператор:

```
character(11) :: txt(0:1)=(/'  простое  ',' составное '/)
...   ...   ...   ...   ...   ...   ...   ...   ...   ...
if (fprimen(n)) then; write(*,*) m=1; else; write(*,*) m=0; endif;
write(*,*) txt(m)
```

Однако, всё можно устроить гораздо проще (причём двумя способами): или через оператор **equivalence** (им не рекомендуется пользоваться), или через встроенную функцию **transfer** современного ФОРТРАНа.

```
use my_prime
implicit none            ! вариант с equivalence - совмещение по памяти
equivalence (l,m)        ! переменных разных типов
logical l
integer m, n
character(11) :: txt(0:1)=(/'  простое  ',' составное '/)
...   ...   ...   ...   ...   ...   ...   ...   ...   ...
l=primen(n); write(*,*) txt(m)
end
```

```
use my_prime
implicit none            ! вариант с transfer
logical l; integer m, n
character(11) :: txt(0:1)=(/'  простое  ',' составное '/)
...   ...   ...   ...   ...   ...   ...   ...   ...   ...
l=primen(n)
m=transfer(l,m); write(*,*) txt(m)
end
```

В самом начале выполняемой части тестирующей программы, приведённой в следующем пункте, продемонстрирована работа и **equivalence**, и **transfer**. Фрагменты вывода этой программы:

```
  Работа equivalence:
 l= T  i=           1
 l= F  i=           0
 i=         123  l= T
 i=           0  l= F
 i=        -123  l= T
```

```
  Работа transfer:
 l= T  k=           1
 l= F  k=           0
 k=         123  l= T
 k=           0  l= F
 k=        -123  l= T
```

О функции **transfer** см. **Приложение I. Функция передачи типа transfer.** «Программирование на ФОРТРАНе И СИ (второй семестр)». Напомним лишь, что **transfer(аргумент_1,аргумент_2)** позволяет на значение типа **аргумент_1** *взглянуть через очки* значения типа **аргумент_2** при сохранении в неприкосновенности битового представления **аргумента_1**, т.е. без осуществления приведения типов.

1.20.1 Тестирующая программа

```
program test_6; use my_prime; implicit none
equivalence (l,i)
integer, parameter :: mmax=10000000
character(11), parameter :: txt(0:1)=(/' составное ',' простое  '/)
integer n, k, m, i; logical l, l0, l1, l2, l3; real t0, t1, t2, t3, t4
l=.true.;  write(*,*) ' l=',l,' i=',i  ! Демонстрируется работоспособность
l=.false.; write(*,*) ' l=',l,' i=',i  ! equivalence (l,i)
i=123;     write(*,*) ' i=',i,' l=',l
i=0;       write(*,*) ' i=',i,' l=',l
i=-123;    write(*,*) ' i=',i,' l=',l
l=.true.;  k=transfer(l,k); write(*,*) ' l=',l,' k=',k  ! Демонстрируется
l=.false.; k=transfer(l,k); write(*,*) ' l=',l,' k=',k  ! работоспособность
i=123;     k=transfer(l,k); write(*,*) ' k=',k,' l=',l  !    transfer
i=0;       k=transfer(l,k); write(*,*) ' k=',k,' l=',l
i=-123;    k=transfer(l,k); write(*,*) ' k=',k,' l=',l
write(*,'(30x,"Время, затраченное на ",i10," вызовов")') mmax ! Тестирование
write(*,'(3x,"Число",3x,"Bool",1x,"m",3x,"Свойство",6x,&
&    "primen",3x,"primer", 2x,"sprimen",2x,"sprimer")')       !   процедур:
do; read (*,*,end=77) n
    write(*,'(i10$)') n
    call cpu_time(t0); do m=1,mmax;      l0=primen( n ); enddo
    call cpu_time(t1); do m=1,mmax; k=3; l1=primer(n,k); enddo
    call cpu_time(t2); do m=1,mmax;      call sprimen(n,l2); enddo
    call cpu_time(t3); do m=1,mmax; k=3; call sprimer(n,k,l3); enddo
    call cpu_time(t4);
    l=l0.and.l1.and.l2.and.l3
    i=transfer(l,i)
    write(*,'(l4,i3,2x,a,4(2x,f7.3))')&
&                                  l0,i,txt(i),t1-t0,t2-t1,t3-t2,t4-t3
enddo
77 stop 0
end
```

- До сих пор мы пользовались лишь двумя первыми аргументами оператора **read**: программным номером устройства ввода и индикатором формата ввода. В данной программе они указаны звёздочками. Первая означает, что по умолчанию предполагается ввод с экрана. Вторая — что нас устраивает форма ввода, которую устанавливает тип данных, включённых в список ввода.

- Однако, оператор **read** может иметь (если нужно) и другие аргументы (или, как говорят, спецификаторы ввода). Так спецификатор с ключевым именем **end=** — **спецификатор перехода по признаку окончания файла** при записи **read(*,*,end=77)** означает,

что как только читающее устройство встретит признак окончания файла, управление будет передано на оператор программы, помеченный меткой **77**. В контексте нашей программы это — оператор **77 stop 0**.

- Напомним, что, хоть наша программа и написана для ввода с экрана, но при запуске исполнимого файла посредством, например,

  ./a.out < input > result

  можно переназначить стандартные устройства ввода-вывода

- Спецификатор **end=77**, хотя и выглядит наглядно и просто, тем не менее требует явного указания метки, что сопряжено с возможностью совершения ошибки (например, ошибочно пометим не тот оператор). Оператор **read** предоставляет более структурное (в смысле безметочное) и всеобъемлющее средство решения сходных проблем посредством спецификатора **iostat=ier — спецификатора типа ошибки**, который вырабатывает код завершения работы оператора **read**.

- Так, если оператор **read(*,*,iostat=ier) n** отработал без ошибки, т.е. вводимые данные формально соответствовали указанной спецификации формата, и при этом не был встречен признак окончания файла, то в переменную **ier**, соответствующую ключевому слову **iostat**, запишется **нуль**; если же признак окончания файла встретился, то — **целое отрицательное**, так что поместив после **read(*,*,iostat=ier) n** оператор **if (ier<0) exit** мы выйдем из цикла на ближайший следующий за ним оператор, который в данном случае не нужно будет помечать меткой.

- Если же при чтении поступил сигнал об ошибке (например, вместо числа во входном потоке из-за сбоя записывающей аппаратуры оказалась буква кириллицы), то в **ier** запишется **целое положительное**, так что посредством оператора **if (ier>0) cycle** получим возможность перейти ко вводу следующего данного, не проводя обработки ошибочного.

1.20.2 Исходные текст модуля my_prime

```
module my_prime; implicit none; contains
function primen(n)
logical primen, p          ! Индикатор подозрительности на простое.
integer, intent(in) :: n
integer k                  ! Хранилище очередного кандидата на делитель.
p=(n==2).or.(mod(n,2)==1)  ! Подозрительно (n) на простое или нет?
k=3                        ! k=3 - первый кандидат на делитель.
do while (p.and.k*k<n+1)   ! Пока n подозрительно и не проверены все
  p=mod(n,k)/=0; k=k+2     ! нечётные кандидаты на делитель уточняем
enddo                      ! подозрительность.
primen=p
end function primen
subroutine sprimen(n,p)       ! Тот же алгоритм, что и у fprimen, но
integer n, k; logical p       ! оформлен подпрограммой. p включён в
p=(n==2).or.(mod(n,2)==1)     ! список аргументов для возврата
k=3; do while (p.and.k*k<n+1) ! результата анализа.
  p=mod(n,k)/=0; k=k+2
enddo
end subroutine sprimen
recursive function primer(n, m) result(p)
logical p; integer n, m; p=(n==2).or.(mod(n,2)==1)
if (m*m<n+1) then; p=mod(n,m)/=0; if (p) p=primer(n,m+2); endif
end function primer
recursive subroutine sprimer(n, m, p)
logical p; integer n, m; p=(n==2).or.(mod(n,2)==1)
if (m*m<n+1) then; p=mod(n,m)/=0; if (p) call sprimer(n,m+2,p);
endif
end subroutine sprimer
end module my_prime
```

1.20.3 Вводимые данные n и m

```
1
2
3
4
5
6
11
31
63
67
101
1111
31313107
2147483647
```

1.20.4 Вывод тестирующей программы при опции -00

```
  l= T  i=           1
  l= F  i=           0
  i=         123  l= T
  i=           0  l= F
  i=        -123  l= T
  l= T  k=           1
  l= F  k=           0
  k=         123  l= T
  k=           0  l= F
  k=        -123  l= T
```

				Время, затраченное на 10000000 вызовов			
Число	Bool	m	Свойство	primen	primer	sprimen	sprimer
1	T	1	простое	0.098	0.096	0.079	0.093
2	T	1	простое	0.061	0.063	0.063	0.065
3	T	1	простое	0.072	0.089	0.080	0.091
4	F	0	составное	0.059	0.089	0.067	0.094
5	T	1	простое	0.072	0.089	0.079	0.093
6	F	0	составное	0.059	0.089	0.067	0.093
11	T	1	простое	0.175	0.223	0.199	0.271
31	T	1	простое	0.198	0.330	0.204	0.349
63	F	0	составное	0.285	0.122	0.275	0.134
67	T	1	простое	0.255	0.451	0.265	0.479
101	T	1	простое	0.314	0.574	0.325	0.620
1111	F	0	составное	0.387	0.639	0.418	0.647
31313107	F	0	составное	0.265	0.377	0.296	0.398
2147483647	T	1	простое	0.072	0.089	0.080	0.093

1.20.5 Фрагмент вывода затрат времени при опции -03

				Время, затраченное на 10000000 вызовов			
Число	Bool	m	Свойство	primen	primer	sprimen	sprimer
1	T	1	простое	0.032	0.100	0.032	0.052
2	T	1	простое	0.028	0.092	0.024	0.040
3	T	1	простое	0.032	0.100	0.036	0.044
4	F	0	составное	0.028	0.112	0.036	0.044
5	T	1	простое	0.036	0.108	0.036	0.052
6	F	0	составное	0.032	0.108	0.036	0.044
11	T	1	простое	0.140	0.272	0.136	0.160
31	T	1	простое	0.248	0.384	0.248	0.304
63	F	0	составное	0.140	0.152	0.144	0.140
67	T	1	простое	0.372	0.560	0.376	0.428
101	T	1	простое	0.484	0.692	0.488	0.592
1111	F	0	составное	0.660	0.928	0.664	0.748
31313107	F	0	составное	0.548	0.772	0.552	0.592

1.21 Задача 6 (СИ-решение)

1.21.1 Тестирующая программа

```
#include <stdio.h>
#include <math.h>
#include <time.h>
 int  primen(int);       int  primer(int,int);
void sprimen(int,int*); void sprimer(int,int,int*);
int main()
{ int const mmax=10000000;
 char txt[2][10]={"composite\0","  prime  \0"};
 long int t0, t1, t2, t3, t4;
  double a;
  int n, k, m, i, l, l0, l1, l2, l3;
  printf("%23s Время, затраченное на %10i вызовов\n"," ",mmax);
  printf("%3s Число   m  Свойство "," ");
  printf("   primen   primer  sprimen   sprimer\n");
  do { i=scanf("%i\n",&n);
       if (i<0) break;
       t0=clock(); for (m=1;m<=mmax; m++) l0=primen(n);
       t1=clock(); for (m=1;m<=mmax; m++) {k=3; l1=primer(n,k);}
       t2=clock(); for (m=1;m<=mmax; m++) sprimen(n,&l2);
       t3=clock(); for (m=1;m<=mmax; m++) {k=3; sprimer(n,k,&l3);}
       t4=clock();
       l=l0 && l1 && l2 && l3;
       printf("%10i %2i %10s  %7.2f  %7.2f  %7.2f  %7.2f\n", n,l0,txt[l],
       (double)(t1-t0)/CLOCKS_PER_SEC, (double)(t2-t1)/CLOCKS_PER_SEC,
       (double)(t3-t2)/CLOCKS_PER_SEC, (double)(t4-t3)/CLOCKS_PER_SEC);
     }
  while (k>0); return 0;
}
```

- Данная программа и тестируемые функции трактуют число 1 как простое, хотя оно не относится ни к простым, ни к составным. Поэтому в список чисел, подаваемых на вход данной программе для проверки на простоту, не следует включать **единицу**.
- Функция **scanf** (помимо сканирования входного потока и ввода данных по адресам указанных переменных) ещё и возвращает через своё имя число почитанных символов. Если оно отрицательно, то встречен признак окончания файла.
- Оператор **break** осуществляет выход из цикла (в частности) на следующий за циклом оператор.

1.21.2 Исходные тексты процедур

```
int primen(int n)
{ int p;                      // Индикатор подозрительности на простое.
  int k;                      // Хранилище очередного кандидата на делитель.
  p=((n==2)||(n%2)==1);       // Подозрительно (n) на простое или нет?
  k=3;                        // k=3 - первый кандидат на делитель.
  while (p && (k*k<n+1))      // Пока n подозрительно и не проверены все
  { p=n%k; k=k+2; }           // нечётные кандидаты на делитель уточняем
  if (p!=0) p=1; return p;    // подозрительность.
}
void sprimen(int n, int*p)
{ int k;                       // Хранилище очередного кандидата на делитель.
  *p=((n==2)||(n%2)==1);       // Подозрительно (n) на простое или нет?
  k=3;                         // k=3 - первый кандидат на делитель.
  while (*p && (k*k<n+1))      // Пока n подозрительно и не проверены все
  { *p=n%k; k=k+2; }           // нечётные кандидаты на делитель уточняем
  if (*p!=0) *p=1;             // подозрительность.
}
int primer(int n, int m)
{ int p;
  p=( (n==2) || (n%2==1) );
  if (m*m<n+1)
    { p=( (n%m)!=0 ); if (p) p=primer(n,m+2); }  return p;
}
void sprimer(int n, int m, int*p)
{
  *p=( (n==2) || (n%2==1) );
if (m*m<n+1) { *p=( (n%m)!=0 ); if (*p) *p=primer(n,m+2); }
}
```

1.21.3 Числа,вводимые для проверки на простоту

```
2
3
4
5
11
31
63
67
101
1111
31313107
```

1.21.4 Вывод тестирующей программы при опции -00

			Время, затраченное на 10000000 вызовов			
Число	m	Свойство	primen	primer	sprimen	sprimer
2	1	prime	0.07	0.05	0.09	0.07
3	1	prime	0.07	0.06	0.08	0.06
4	0	composite	0.06	0.06	0.07	0.06
5	1	prime	0.07	0.06	0.08	0.06
11	1	prime	0.18	0.23	0.20	0.24
31	1	prime	0.30	0.41	0.34	0.41
63	0	composite	0.18	0.17	0.20	0.19
67	1	prime	0.44	0.60	0.49	0.61
101	1	prime	0.56	0.78	0.63	0.84
1111	0	composite	0.74	0.97	0.82	1.05
31313107	0	composite	0.60	0.72	0.65	0.73

1.21.5 Вывод тестирующей программы при опции -01

			Время, затраченное на 10000000 вызовов			
Число	m	Свойство	primen	primer	sprimen	sprimer
2	1	prime	0.03	0.05	0.03	0.04
3	1	prime	0.03	0.04	0.03	0.04
4	0	composite	0.03	0.04	0.03	0.04
5	1	prime	0.04	0.04	0.04	0.04
11	1	prime	0.14	0.16	0.14	0.17
31	1	prime	0.25	0.30	0.25	0.30
63	0	composite	0.14	0.14	0.14	0.15
67	1	prime	0.37	0.46	0.38	0.47
101	1	prime	0.48	0.60	0.49	0.60
1111	0	composite	0.64	0.77	0.67	0.77
31313107	0	composite	0.54	0.60	0.55	0.60

1.21.6 Вывод тестирующей программы при опции -02

			Время, затраченное на 10000000 вызовов			
Число	m	Свойство	primen	primer	sprimen	sprimer
1	1	prime	0.03	0.03	0.03	0.03
2	1	prime	0.03	0.03	0.03	0.04
3	1	prime	0.03	0.03	0.03	0.03
4	0	composite	0.03	0.03	0.03	0.03
5	1	prime	0.03	0.03	0.04	0.03
6	0	composite	0.03	0.03	0.03	0.03
11	1	prime	0.13	0.14	0.14	0.15
31	1	prime	0.24	0.25	0.25	0.26
63	0	composite	0.13	0.14	0.14	0.14
67	1	prime	0.37	0.38	0.38	0.39
101	1	prime	0.48	0.49	0.49	0.50
1111	0	composite	0.65	0.70	0.66	0.67

```
31313107  0  composite     0.53     0.57     0.54     0.56
```

1.21.7 Вывод тестирующей программы при опции -03

```
                    Время, затраченное на   10000000 вызовов
  Число   m  Свойство    primen   primer  sprimen   sprimer
      1   1    prime       0.03     0.03     0.03     0.03
      2   1    prime       0.03     0.03     0.03     0.04
      3   1    prime       0.03     0.03     0.03     0.03
      4   0  composite     0.03     0.03     0.03     0.03
      5   1    prime       0.03     0.03     0.04     0.03
      6   0  composite     0.03     0.03     0.03     0.03
     11   1    prime       0.13     0.14     0.14     0.15
     31   1    prime       0.24     0.25     0.25     0.26
     63   0  composite     0.13     0.14     0.14     0.14
     67   1    prime       0.37     0.38     0.38     0.39
    101   1    prime       0.48     0.49     0.49     0.50
   1111   0  composite     0.65     0.70     0.66     0.67
31313107  0  composite     0.53     0.57     0.54     0.56
```

1.21.8 Сравнение затрат времени gfortran и gcc на 10000000 вызовов.

	-O0		**-O1**		**-O2**		**-O3**	
Функция	**fortran**	**gcc**	**fortran**	**gcc**	**fortran**	**gcc**	**fortran**	**gcc**
primen 101	0.66	0.56	0.53	0.48	0.49	0.48	0.48	0.48
primen 1111	0.86	0.74	0.71	0.64	0.66	0.65	0.66	0.65
primer 101	0.93	0.78	0.63	0.60	0.74	0.49	0.69	0.49
primer 1111	1.12	0.97	0.80	0.77	0.96	0.70	0.93	0.70
sprimen 101	0.71	0.63	0.51	0.49	0.49	0.49	0.49	0.49
sprimen 1111	0.92	0.82	0.68	0.67	0.66	0.66	0.66	0.66
sprimer 101	0.98	0.84	0.64	0.60	0.65	0.50	0.59	0.50
sprimer 1111	1.24	1.05	0.81	0.77	0.91	0.67	0.75	0.67

2 Производные типы данных (второй семестр).

(знакомство с массивами, стуктурами и указателями):

1. **Задача 1.**

   a) Написать подпрограмму **swap_arint**, которая обменивает содержимым массивы (в каждом по 10 элементов типа **integer**) через посредство простой переменной соответствующего типа.

   b) Написать подпрограмму **swap_stint**, обменивающую содержимым переменные производного типа (структуры)

   ```
   type stint
       integer, dimension (10) :: p
   end type stint
   ```

   через посредство простой переменной соответствующего типа

   c) Описание типа **stint** и подпрограмм обмена разместить в модуле **mhome2**.

   d) Тестирующая программа должна замерять временные затраты на **kmax** обращений к каждой из подпрограмм соответственно.

   e) Обеспечить пропуск программы, используя соответствующий **makefile**, в котором цель генерации исполнимого файла зависит от наличия объектных файлов главной программы и модуля **mhome2**, цель вызова исполнимого файла реализуется через посредство команды **time**, а файлы ввода/вывода (**input** и **result**) из текущей директории задействованы через операции переназначения стандартных потоков ввода/вывода.

2. Задача та же, но тип элементов **real** (float).

3. Задача та же, но тип элементов **real(8)** (double).

4. Задача та же, но тип элементов **real(10)** (long double).

5. Задача та же, но тип элементов **character(1302)**. (заполнить элементы оператором присваивания, а не ввода).

6. Задача та же, но процедуру необходимо оформить так, чтобы её вызов по имени **swap** был в состоянии выполнить работу любой

из выше указанных процедур. Главная программа должна вывести табличку вида:

Тип данных	Время **swap** при работе с массивами из 10 элементов	Время **swap** при работе со структурами из 10 полей
integer real(4) real(8) real(16) character(1333)		

7. Задача. Модифицировать СИ- и ФОРТРАН-решения задачи о считалке так, чтобы оперативная память, отведённая изначально для кандидатов на водящего, передавалась по мере их выбывания операционной системе.

8. Написать две СИ-функции расчёта значения полинома **n**-ой степени по заданным значениям его коэффициентов, хранящихся в массиве, и аргументу. В первой формальный параметр-массив описать в терминах конструкции **[]**; во второй — через указатель.

   Продемонстрировать работу тестирующей программы.

2.1 Уяснение ситуации

Решение любой из ФОРТРАН-задач должно использовать исходные тексты двух файлов: главной программы **main.f** и модуля **mhome2.f** (возможны расширения **.f90** или **.f95**) и файла ввода **input**.

Без модуля пришлось бы, например, включать описания типов в файлы главной программы и процедур, используя директиву **include**. Кроме того, в задаче **5** для описания родового имени **swap** потребовались бы имена типов и процедур из всех предыдущих задач этого домашнего задания. Последнее легко реализуется, если упомянутые имена описаны в одном модуле. Поэтому в задачах **1**–**5** описания всех производных типов и подпрограмм включаем в модуль **mhome2**.

В приводимых далее решениях задач **1**–**4** исходный текст модуля **mhome2**, содержит описания только типов и процедур, касающихся конкретной рассматриваемой задачи. Невостребованные в ней типы и процедуры опущены, чтобы не мешать восприятию решения, однако при решении задачи **5** они естественно потребуются.

2.2 Задача 1 (решение)

2.2.1 Тестирующая программа

```
 program test_1; use mhome2; implicit none
 type (sti) :: c, d          ! c и d - переменные типа sti
                             ! сам тип sti описан в mhome2.
 integer  :: k, a(n), b(n)   ! a и b - целочисленные массивы
                             ! n=10 - константа, задана в mhome2.
 real t0, t1                 ! для хранения отсчётов времени.
 read (*,*) a, b             ! ввод массивов
 write(*,'(" a=",1000i4)') a ! их контрольный
 write(*,'(" b=",1000i4)') b !                вывод до обмена
 call second(t0)             ! фиксация начала отсчёта времени.
 do k=1, kmax                ! kmax-кратный (kmax задано в mhome2)
   call swap_ari(a,b)        ! обмен содержимым массивов
 enddo                       !
 call second(t1)             ! фиксация момента завершения обмена.
 write(*,*) 'Время на ',kmax,'-кратный вызов swap_ari равно',
>                                                    t1-t0
 write(*,'(" a=",10i4)') a   ! контрольный вывод массивов
 write(*,'(" b=",10i4)') b   ! после обмена
 write(*,'(50("="))')        ! для наглядности отделяем результат
                             ! обмена массивов пограничной чертой
 call rdr_sti(c)           ! ввод значений производного типа
 call rdr_sti(d)           ! (описание процедур ввода rdr_sti
 write(*,'(" c="$)'); call prt_sti(c)      ! и вывода prt_sti
 write(*,'(" d="$)'); call prt_sti(d)      ! в модуле mhome2)
 call second(t0)
 do k=1, kmax; call swap_sti(c,d); enddo
 call second(t1)
 write(*,*) 'Время на ',kmax,'-кратный вызов swap_sti равно',
>                                                   t1-t0
 write(*,'(" c="$)'); call prt_sti(c)
 write(*,'(" d="$)'); call prt_sti(d)
 end
```

1. Описание производного типа **sti** помещено в модуль **mhome2**.

2. Описание процедур **swap_ari**, **swap_sti**, **rdr_sti** и **prt_sti** помещено в модуль **mhome2**.

3. Обратите внимание на символ **$** — последний символ форматной строки, например, в операторе **write(*,'("c:"$)')**. Он обеспечит вывод очередного оператора **write** (из **prt_sti**) непосредственно вслед за **c:**, т.е. в ту же строку (иногда это бывает удобно).

2.2.2 Исходный текст модуля mhome2

```
      module mhome2; implicit none
      integer,parameter :: kmax=100000001 ! Число вызовов процедур обмена
      integer,parameter :: n=10           ! Число элементов в массиве и
                                          ! в p-поле типа stint.
      type  stint                         ! Определение производного типа
        integer, dimension(n) :: p        ! stint - структуры с одним
      end type stint                      ! полем p из n элементов
      contains
! ============================================================
! Подпрограмма swap_arint(a,b) обменивает содержимым массивы
! a(n) и b(n), используя одну рабочую переменную типа integer.
! ............................................................
      subroutine swap_arint(a,b); integer  a(n), b(n) ! константа n
      integer i, w                                    ! описанная в
      do i=1,n                                        ! модуле доступна
        w=a(i); a(i)=b(i); b(i)=w                     ! всем процедурам
      enddo                                           ! модуля.
      end subroutine swap_arint
! ============================================================
! Подпрограмма swap_stint(a,b) обменивает содержимым структуры
! a и b типа stint, используя одну рабочую переменную.
! stint - структура данных с полем p типа integer, dimension(n).
! ............................................................
      subroutine swap_stint(a,b); type (stint) a, b
      integer i, j
      do i=1,n
        j=a%p(i)
        a%p(i)=b%p(i)
        b%p(i)=j
      enddo
      end subroutine swap_stint
! ============================================================
! Подпрограмма rdr_stint(x) предназначена для вывода значений типа
! stint - структура данных с полем p типа integer, dimension(n).
! ............................................................
      subroutine rdr_stint(x); type (stint) x
      read(*,*) x%p
      end subroutine rdr_stint
! ============================================================
! Подпрограмма prt_stint(x) предназначена для вывода значений типа
! stint - структура данных с полем p типа integer, dimension(n).
! ............................................................
      subroutine prt_stint(x); type (stint) x
      write(*,'(1000i4)') x%p
      end subroutine prt_stint
      end module mhome2
```

2.2.3 Результат работы программы test_1 (опция -O0)

```
 a=   0   1   2   3   4   5   6   7   8   9
 b=   9   8   7   6   5   4   3   2   1   0
 Время на    100000001 -кратный вызов swap_arint равно   8.6736803
 a=   9   8   7   6   5   4   3   2   1   0
 b=   0   1   2   3   4   5   6   7   8   9
================================================
 c=   0   1   2   3   4   5   6   7   8   9
 d=   9   8   7   6   5   4   3   2   1   0
 Время на    100000001 -кратный вызов swap_stint равно   8.4277182
 c=   9   8   7   6   5   4   3   2   1   0
 d=   0   1   2   3   4   5   6   7   8   9
```

1. Оператор **write(*,'("a=",10i4)') a** выводит значения всех десяти элементов вектора **a** в ту же самую строку, что и сочетание **a=**, размещая очередное значение в очередных **четырёх** позициях.

   10 (число перед дескриптором формата **i4**) — повторитель, который указывает, сколько значений надо вывести в одну строку. Так оператор **write(*,'(i4)') a** вывел бы каждый элемент в новую строку.

   i4 — один из возможных дескрипторов *описателей* формата. Литера **i** (**integer**) означает, что при вводе число следует перевести в машинное представление для **целых** чисел, а при выводе — из машинного представления целого в обычное десятичное.

2. В тексте программы **test_1** оператор **write(*,'("a= 1000i4)') a** лишь демонстрирует, что когда повторитель (в данном случае **1000**) превышает число элементов списка вывода (в данном случае **10**), то ФОРТРАН игнорирует инструкции форматной строки, для которых не нашлось элементов списка вывода. Задание некоторого наибольшего значения повторителя, превышающего число выводимых элементов, бывает удобно, когда заранее неизвестно число последних.

3. Старые версии ФОРТРАНа не разрешали в качестве повторителя формата использовать имена переменных. Некоторые современные компиляторы допускают, например, конструкцию <**n**>**i4**, где **n** — имя константы или переменной. **gfortran** подобную конструкцию не допускает, но предоставляет возможность смоделировать ситуацию динамического повторителя формата, используя понятие так называемого внутреннего файла, в качестве которого могут использоваться переменные типа **character** (как — узнаем позже).

1. При чётном **kmax** возможна иллюзия о неверной работе процедур. **Почему?**
2. О чём свидетельствуют выведенные временные затраты?
3. Почему **swap_arint** работает медленнее **swap_stint**?
4. Работает ли в процедурах **стандарт правила умолчания**?
5. Результаты пропусков при разных ключах оптимизации:

```
a=   0   1   2   3   4   5   6   7   8   9     Ключ оптимизации -O1
b=   9   8   7   6   5   4   3   2   1   0     ====================
Время на    100000001 -кратный вызов swap_arint равно   1.8157229
a=   9   8   7   6   5   4   3   2   1   0
b=   0   1   2   3   4   5   6   7   8   9
.......................................
c=   0   1   2   3   4   5   6   7   8   9
d=   9   8   7   6   5   4   3   2   1   0
Время на    100000001 -кратный вызов swap_stint равно   1.8147241
c=   9   8   7   6   5   4   3   2   1   0
d=   0   1   2   3   4   5   6   7   8   9
================================================================

a=   0   1   2   3   4   5   6   7   8   9     Ключ оптимизации -O2
b=   9   8   7   6   5   4   3   2   1   0     ====================
Время на    100000001 -кратный вызов swap_arint равно   1.7147400
a=   9   8   7   6   5   4   3   2   1   0
b=   0   1   2   3   4   5   6   7   8   9
.......................................
c=   0   1   2   3   4   5   6   7   8   9
d=   9   8   7   6   5   4   3   2   1   0
Время на    100000001 -кратный вызов swap_stint равно   1.7137389
c=   9   8   7   6   5   4   3   2   1   0
d=   0   1   2   3   4   5   6   7   8   9
================================================================

a=   0   1   2   3   4   5   6   7   8   9     Ключ оптимизации -O3
b=   9   8   7   6   5   4   3   2   1   0     ====================
Время на    100000001 -кратный вызов swap_arint равно   1.5127701
a=   9   8   7   6   5   4   3   2   1   0
b=   0   1   2   3   4   5   6   7   8   9
.......................................
c=   0   1   2   3   4   5   6   7   8   9
d=   9   8   7   6   5   4   3   2   1   0
Время на    100000001 -кратный вызов swap_stint равно   1.4637769
c=   9   8   7   6   5   4   3   2   1   0
d=   0   1   2   3   4   5   6   7   8   9
```

2.3 Задача 2. Тип элементов real(4)

2.3.1 Тестирующая программа и вывод результата её работы

```
 program test_2; use mhome2; implicit none
 type (st4) :: c, d; real a(n), b(n); integer  :: k; real t0, t1
 read (*,*) a, b
 write(*,'(" a=",10f6.1)') a; write(*,'(" b=",10f6.1)') b
 call second(t0); do k=1, kmax; call swap_ar4(a,b); enddo
 call second(t1)
 write(*,'(/a,i10,a,f10.7/)')
> 'Время на ',kmax,'-кратный вызов swap_ar4 равно ', t1-t0
 write(*,'(" a=",10f6.1)') a; write(*,'(" b=",10f6.1)') b
 write(*,'(70("="))')
 call rdr_st4(c);     call rdr_st4(d)
 write(*,'(" c="$)'); call prt_st4(c)
 write(*,'(" d="$)'); call prt_st4(d)
 call second(t0); do k=1, kmax; call swap_st4(c,d); enddo
 call second(t1)
 write(*,'(/a,i10,a,f10.3/)')
> 'Время на ',kmax,'-кратный вызов swap_st4 равно ', t1-t0
 write(*,'(" c="$)'); call prt_st4(c)
 write(*,'(" d="$)'); call prt_st4(d)
 end
```

1. В настоящей программе выводятся значения типа **real**. Например, **write(*,'(" a=",10f6.1)') a**. Здесь **10** — повторитель формата **f6.1**.

   f6.1 — ещё один из многих дескрипторов (описателей) формата. Литера **f** означает, что при выводе необходимо преобразовать машинное представление типа **real** в привычную форму записи десятичного числа с **фиксированной** запятой: на всю запись **шесть** позиций, из которых **одна** отводится для цифры после запятой.

2. Дескриптор **a** (см., например,

   ```
    write(*,'(/a,i10,a,f10.7/)')
   >      'Время на ',kmax,'-кратный вызов swap_ar4 равно', t1-t0
   ```

   служит для вывода строки. Для вывода трёх первых символов используем форму **a3**.

3. Дескриптор / — признак завершения передачи данных в текущую запись (попросту — перевод строки). Здесь он использован для обрамления сообщения о временных затратах двумя пробельными строками (повышаем наглядность результата).

2.3.2 Фрагмент текста модуля mhome2 для задачи 2

```
      module mhome2; implicit none
      integer,parameter :: kmax=500000001
      integer,parameter :: n=10
      type  st4                     ! Определение производного типа
        real(4), dimension(n) :: p  ! st4 - структуры с одним полем
      end type st4                  ! p из n элементов типа real(4)
      contains
! ================================================================
! Подпрограмма swap_ar4(a,b,n) обменивает содержимым массивы
! a(n) и b(n), используя одну рабочую переменную типа real(4).
! ................................................................
      subroutine swap_ar4(a,b); real(4) a(n), b(n), w
      integer i
      do i=1,n
        w=a(i); a(i)=b(i); b(i)=w
      enddo
      end subroutine swap_ar4
! ================================================================
! Подпрограмма swap_st4(a,b) обменивает содержимым структуры a и b
! типа st4, используя одну рабочую переменную типа real(4).
! st4 - структурный тип данных с полем p типа real(4), dimension(n).
! ................................................................
      subroutine swap_st4(a,b); type (st4) a, b
      integer i; real(4) w
      do i=1,n
        w=a%p(i)
        a%p(i)=b%p(i)
        b%p(i)=w
      enddo
      end subroutine swap_st4
! ================================================================
! Подпрограмма rdr_st4(x) предназначена для вывода значений типа
! st4 - структурный тип данных с полем p типа real(4), dimension(n).
! ................................................................
      subroutine rdr_st4(x); type (st4) x
      read(*,*) x%p
      end subroutine rdr_st4
! ================================================================
! Подпрограмма prt_st4(x) предназначена для вывода значений типа
! st4 - структурный тип данных с полем p типа real(4), dimension(n).
! ................................................................
      subroutine prt_st4(x); type (st4) x
      write(*,'(1000f6.2)') x%p
      end subroutine prt_st4
      end module mhome2
```

На первый взгляд кажется, что процедуры **rdr_st4** и **prt_st4** особо не нужны. Действительно, в главной программе вместо вызова процедуры можно написать

```
read(*,*) c%p; read(*,*) d%p
```

Однако, если в процессе разработки программы выяснено, что нужны объекты производного типа, то значит — нам выгодно оперировать объектами этого типа, как единым целым. Поэтому инкрустация в такую программу операторов явной работы с его частями нежелательна.

Наше кредо: при наличии структур стараемся пользоваться только их именами (т.е. предпочтительнее **rdr_st4(c)**, а не **read(*,*) c%p**, допуская последнее в качестве временного тактического средства).

2.3.3 Результат работы программы test_2 (опция -O0)

```
 a=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
 b=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0

Время на  100000001-кратный вызов swap_ar4 равно  8.7696657

 a=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0
 b=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
================================================================
 c=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
 d=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0

Время на  100000001-кратный вызов swap_st4 равно      8.411

 c=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0
 d=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
```

Имя структуры	**dimension(n)**				**st4**			
Опция	**-O0**	**-O1**	**-O2**	**-O3**	**-O0**	**-O1**	**-O2**	**-O3**
Базовый тип								
integer(4)	8.674	1.816	1.715	1.513	8.427	1.815	1.714	1.463
real(4)	8.770	1.769	1.715	1.513	8.411	1.859	1.715	1.467

1) временные затраты на вызов процедуры обмена значениями двух структур с полями типа **real, dimension (n)** меньше времени обмена двух массивов того же типа.

2) работа с типом **real(4)** идёт медленнее чем с типом **integer**.

2.4 Задача 3. Тип элементов real(8)

2.4.1 Тестирующая программа test_3

```
      program test_3; use mhome2; implicit none
      type (st8) :: c, d; real(8) a(n), b(n); integer  :: k
      real t0, t1
      read (*,*) a, b
      write(*,'(" a=",10f6.1)') a
      write(*,'(" b=",10f6.1)') b
      call second(t0); do k=1, kmax; call swap_ar8(a,b); enddo
      call second(t1)
      write(*,'(/a,i10,a,f10.3/)')
     > 'Время на ',kmax,'-кратный вызов swap_ar8 равно ', t1-t0
      write(*,'(" a=",10f6.1)') a
      write(*,'(" b=",10f6.1)') b
      write(*,'(70("="))')
      call rdr_st8(c)
      call rdr_st8(d)
      write(*,'(" c="$)'); call prt_st8(c)
      write(*,'(" d="$)'); call prt_st8(d)
      call second(t0); do k=1, kmax; call swap_st8(c,d); enddo
      call second(t1)
      write(*,'(/a,i10,a,f10.3/)')
     > 'Время на ',kmax,'-кратный вызов swap_st8 равно ', t1-t0
      write(*,'(" c="$)'); call prt_st8(c)
      write(*,'(" d="$)'); call prt_st8(d)
      end
```

2.4.2 Результат работы программы test_3 (опция -O0)

```
 a=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
 b=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0

Время на  100000001-кратный вызов swap_ar8 равно      8.060

 a=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0
 b=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
======================================================================
 c=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
 d=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0

Время на  100000001-кратный вызов swap_st8 равно      8.068

 c=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0
 d=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
```

2.4.3 Фрагмент текста модуля mhome2 для задачи 3

```
      module mhome2; implicit none
      integer,parameter :: kmax=100000001
      integer,parameter :: n=10
      type  st8                     ! Определение производного типа
        real(8), dimension(n) :: p  ! st8 - структуры с одним полем
      end type st8                  ! p из n элементов типа real(8)
      contains
! ================================================================
! Подпрограмма swap_ar8(a,b,n) обменивает содержимым массивы
! a(n) и b(n), используя одну рабочую переменную типа real(8).
! ................................................................
      subroutine swap_ar8(a,b); real(8) a(n), b(n), w
      integer i
      do i=1,n
        w=a(i); a(i)=b(i); b(i)=w
      enddo
      end subroutine swap_ar8
! ================================================================
! Подпрограмма swap_st8(a,b) обменивает содержимым структуры a и b
! типа st8, используя одну рабочую переменную типа real(8).
! st8 - структурный тип данных с полем p типа real(8), dimension(n).
! ................................................................
      subroutine swap_st8(a,b); type (st8) a, b
      integer i; real(8) w
      do i=1,n
        w=a%p(i)
        a%p(i)=b%p(i)
        b%p(i)=w
      enddo
      end subroutine swap_st8
! ================================================================
! Подпрограмма rdr_st8(x) предназначена для ввода значений типа
! st8 - структурный тип данных с полем p типа real(8), dimension(n).
! ................................................................
      subroutine rdr_st8(x); type (st8) x
      read(*,*) x%p
      end subroutine rdr_st8
! ================================================================
! Подпрограмма prt_st8(x) предназначена для вывода значений типа
! st8 - структурный тип данных с полем p типа real(8), dimension(n).
! ................................................................
      subroutine prt_st8(x); type (st8) x
      write(*,'(10f6.1)') x%p
      end subroutine prt_st8
      end module mhome2
```

2.5 Задача 4. Тип элементов real(10)

2.5.1 Тестирующая программа test_4

```
      program test_4; use mhome2; implicit none
      type (st10) :: c, d; real(10) a(n), b(n); integer  :: k
      real t0, t1
      read (*,*) a, b
      write(*,'(" a=",10f6.1)') a
      write(*,'(" b=",10f6.1)') b
      call second(t0); do k=1, kmax; call swap_ar10(a,b); enddo
      call second(t1)
      write(*,'(/a,i10,a,f10.3/)')
     > 'Время на ',kmax,'-кратный вызов swap_ar10 равно ', t1-t0
      write(*,'(" a=",10f6.1)') a
      write(*,'(" b=",10f6.1)') b
      write(*,'(70("="))')
      call rdr_st10(c)
      call rdr_st10(d)
      write(*,'(" c="$)'); call prt_st10(c)
      write(*,'(" d="$)'); call prt_st10(d)
      call second(t0); do k=1, kmax; call swap_st10(c,d); enddo
      call second(t1)
      write(*,'(/a,i10,a,f10.3/)')
     > 'Время на ',kmax,'-кратный вызов swap_st10 равно ', t1-t0
      write(*,'(" c="$)'); call prt_st10(c)
      write(*,'(" d="$)'); call prt_st10(d)
      end
```

2.5.2 Результат работы программы test_4 (опция -O0)

2.5.3 Вывод тестирующей программы

```
 a=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
 b=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0

Время на  100000001-кратный вызов swap_ar10 равно     19.955

 a=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0
 b=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
======================================================================
 c=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
 d=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0

Время на  100000001-кратный вызов swap_st10 равно     20.235

 c=   9.0   8.0   7.0   6.0   5.0   4.0   3.0   2.0   1.0   0.0
 d=   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
```

2.5.4 Фрагмент модуля mhome2 для задачи 4

```
      module mhome2; implicit none
      integer,parameter :: kmax=100000001
      integer,parameter :: n=10
      type  st10                      ! Определение производного типа
        real(10), dimension(n) :: p   ! st8 - структуры с одним полем
      end type st10                   ! p из n элементов типа real(8)
      contains
! ================================================================
! Подпрограмма swap_ar10(a,b,n) обменивает содержимым массивы
! a(n) и b(n), используя одну рабочую переменную типа real(10).
! ................................................................
      subroutine swap_ar10(a,b); real(10) a(n), b(n), w
      integer i
      do i=1,n; w=a(i); a(i)=b(i); b(i)=w; enddo
      end subroutine swap_ar10
! ================================================================
! Подпрограмма swap_st10(a,b) обменивает содержимым структуры a и b
! типа st10, используя одну рабочую переменную типа real(10).
! st10 - структурный тип данных с полем p типа real(10), dimension(n).
! ................................................................
      subroutine swap_st10(a,b); type (st10) a, b
      integer i; real(10) w
      do i=1,n; w=a%p(i); a%p(i)=b%p(i); b%p(i)=w; enddo
      end subroutine swap_st10
! ================================================================
! Подпрограмма rdr_st10(x) предназначена для ввода значений типа
! st10 - структурный тип данных с полем p типа real(10), dimension(n).
! ................................................................
      subroutine rdr_st10(x); type (st10) x; read(*,*) x%p
      end subroutine rdr_st10
! ================================================================
! Подпрограмма prt_st8(10) предназначена для вывода значений типа
! st10 - структурный тип данных с полем p типа real(10), dimension(n).
      subroutine prt_st10(x); type (st10) x; write(*,'(10f6.1)') x%p
      end subroutine prt_st10
      end module mhome2
```

2.5.5 Сравнение временных затрат

Имя структуры	**dimension(n)**				**st4**			
Опция	**-O0**	**-O1**	**-O2**	**-O3**	**-O0**	**-O1**	**-O2**	**-O3**
Базовый тип								
integer(4)	8.674	1.816	1.715	1.513	8.427	1.815	1.714	1.463
real(4)	8.770	1.769	1.715	1.513	8.411	1.859	1.715	1.467
real(8)	8.060	1.816	1.715	1.414	8.068	1.614	1.766	1.729
real(10)	19.96	4.338	4.638	4.882	20.24	5.344	5.205	4.444

2.6 Задача 5. Тип элементов character(1302), dimension (n)

2.6.1 Тестирующая программа test_5

```
program test_5; use mhome2; implicit none
type (stchar) :: c, d
integer  :: k
character(1302) a(n), b(n)
real t0, t1
call init_char(a,b); write(*,'(" a:")'); call prt_char(a)
                     write(*,'(" b:")'); call prt_char(b)
call second(t0); do k=1, kmax; call swap_char(a,b); enddo
call second(t1)
write(*,*) 'Время на ',kmax,'-кратный вызов swap_char равно',
>                                                          t1-t0
write(*,'(" a:")')  ; call prt_char(a)
write(*,'(" b:")')  ; call prt_char(b)
write(*,'(50("="))'); call init_stchar(c,d,a,b)
write(*,'(" c:")'  ); call prt_stchar(c)
write(*,'(" d:")'  ); call prt_stchar(d)
call second(t0); do k=1, kmax; call swap_stchar(c,d); enddo
call second(t1)
write(*,*) 'Время на ',kmax,'-кратный вызов swap_stchar равно',
>                                                         t1-t0
write(*,'(" c:")'); call prt_stchar(c)
write(*,'(" d:")'); call prt_stchar(d)
end
```

1. Структура программы аналогична предыдущим, НО значения элементов векторов и производных типов не вводятся из файла ввода, а генерируются программно процедурами **init_char** и **init_stchar**.

2. В **init_char** используем встроенные функции **achar** и **iachar**, и операцию **конкатенация** (сцепка строковых значений), обозначаемую двумя значками // , следующими друг за другом без пробела.

3. **achar(номер)** находит по **ASCII**-номеру символа сам символ.
   iachar(символ) — по **ASCII**-символу его номер.

4. Так как в наборе **ASCII**-символов буквы латиницы следуют непосредственно друг за другом в алфавитном порядке, а цифры — в порядке возрастания обозначаемых ими чисел, то:

```
achar(iachar('A'))='A'     (похоже exp(ln(x))=x)
achar(iachar('B')-1)='A'
iachar('7')-iachar('0')=7  (слева '7' - цифра, а 7 справа - целое)
```

2.6.2 Фрагмент текста модуля mhome2 для задачи 5

```
      module mhome2; implicit none
      integer,parameter :: kmax=1000001
      integer,parameter :: n=10
      type  stchar                          ! Определение производного типа
        character(1302),dimension(n) :: p ! stchar - структуры с одним
      end type stchar                       ! полем p из n элементов
                                            ! (каждый 1302 байта)
      contains
! ============================================================
! Подпрограмма init_char(x,y) заполняет каждый из элементов
! вектора x(n) типа character(1302) пятьюдесятью копиями
! ASCII-латиницы в алфавитном порядке (50*26=1300), обрамляя этот
! набор копий цифрой, обозначающей число меньшее номера элемента
! вектора x на единицу.  Вектор y(n) заполняется аналогично, но в
! порядке обратном алфавитному.
! Цель процедуры - программное моделирование содержимого исходных
! векторов, чтобы не загромождать файл ввода.
! ............................................................
      subroutine init_char(x, y)
      character(1302) x(n), y(n)
      character(1300) w
      character(26) s; character(1) f; integer i, j, j13
      do i=1,n; w=''
        do j=1,26; s(j:j)=achar(iachar('A')+j-1); enddo
        do j=1,50; w=trim(w)//s; enddo
        f=achar(iachar('0')+i-1); x(i)=f//trim(w)//f
        do j=1,650
          j13=1300-j+1; f=w(j:j); w(j:j)=w(j13:j13); w(j13:j13)=f
        enddo
        f=achar(iachar('0')+i-1); y(i)=f//trim(w)//f
      enddo
      end subroutine init_char
! ============================================================
! Инициализация элементов производного типа stchar осуществляется
! копированием значений векторов x и y в соответствующее поле
! структур c и d. Копирование оформлено процедурой, чтобы не
! загромождать главную программу подробностями иерархической
! структуры производного типа данных
! ............................................................
      subroutine init_stchar(c,d,x,y)
      character(1302) x(n), y(n)
      type (stchar) c, d
      c%p=x
      d%p=y
      end subroutine init_stchar
```

```
! ================================================================
! Подпрограмма swap_char(a,b) обменивает содержимым массивы
! a(n) и b(n) с элементами строкового типа character(1302),
! используя одну рабочую переменную типа character(1302).
! ................................................................
      subroutine swap_char(a,b); character(1302) a(n), b(n)
      integer i
      character(1302) w
      do i=1,n
        w=a(i); a(i)=b(i); b(i)=w
      enddo
      end subroutine swap_char
! ================================================================
! Подпрограмма swap_stchar(a,b) обменивает содержимым структуры
! a и b типа stchar, используя одну рабочую переменную.
! stchar - структура данных с полем p типа
!                                  character(1300),dimension(n).
! ................................................................
      subroutine swap_stchar(a,b); type (stchar) a, b
      character(1302) j; integer i
      do i=1,n
        j=a%p(i)
        a%p(i)=b%p(i)
        b%p(i)=j
      enddo
      end subroutine swap_stchar
! ================================================================
!Подпрограмма prt_char(x) выводит первые n и последние n символы
! каждого из n типа character(1300) вектора x(n)
! ................................................................
      subroutine prt_char(x); character(1302) x(n)
      integer i
      do i=1,n/2
        write(*,'(2(a10," ... ,",a10,"   "))')
     >  x(i)(1:10),x(i)(1293:1302), x(5+i)(1:10),x(5+i)(1293:1302)
      enddo
      end subroutine prt_char
! ================================================================
! Подпрограмма prt_stchar(x) предназначена для вывода значений типа
! stchar - структура данных с полем p типа
!                                  character(1300),dimension(n).
! ................................................................
      subroutine prt_stchar(x); type (stchar) x
      integer i
      write(*,'(2(a5," ... ,",a5,"    "))') (x%p(i)(1:5),
     > x%p(i)(1298:1302), x%p(5+i)(1:5),x%p(5+i)(1298:1302),i=1,n/2)
      end subroutine prt_stchar
      end module mhome2
```

2.6.3 Результат работы программы test_5 (опция -O0)

```
 a:
0ABCDEFGHI, ... ,RSTUVWXYZ0   5ABCDEFGHI, ... ,RSTUVWXYZ5
1ABCDEFGHI, ... ,RSTUVWXYZ1   6ABCDEFGHI, ... ,RSTUVWXYZ6
2ABCDEFGHI, ... ,RSTUVWXYZ2   7ABCDEFGHI, ... ,RSTUVWXYZ7
3ABCDEFGHI, ... ,RSTUVWXYZ3   8ABCDEFGHI, ... ,RSTUVWXYZ8
4ABCDEFGHI, ... ,RSTUVWXYZ4   9ABCDEFGHI, ... ,RSTUVWXYZ9
 b:
0ZYXWVUTSR, ... ,IHGFEDCBA0   5ZYXWVUTSR, ... ,IHGFEDCBA5
1ZYXWVUTSR, ... ,IHGFEDCBA1   6ZYXWVUTSR, ... ,IHGFEDCBA6
2ZYXWVUTSR, ... ,IHGFEDCBA2   7ZYXWVUTSR, ... ,IHGFEDCBA7
3ZYXWVUTSR, ... ,IHGFEDCBA3   8ZYXWVUTSR, ... ,IHGFEDCBA8
4ZYXWVUTSR, ... ,IHGFEDCBA4   9ZYXWVUTSR, ... ,IHGFEDCBA9
 Время на      1000001 -кратный вызов swap_char равно   3.0625341
 a:
0ZYXWVUTSR, ... ,IHGFEDCBA0   5ZYXWVUTSR, ... ,IHGFEDCBA5
1ZYXWVUTSR, ... ,IHGFEDCBA1   6ZYXWVUTSR, ... ,IHGFEDCBA6
2ZYXWVUTSR, ... ,IHGFEDCBA2   7ZYXWVUTSR, ... ,IHGFEDCBA7
3ZYXWVUTSR, ... ,IHGFEDCBA3   8ZYXWVUTSR, ... ,IHGFEDCBA8
4ZYXWVUTSR, ... ,IHGFEDCBA4   9ZYXWVUTSR, ... ,IHGFEDCBA9
 b:
0ABCDEFGHI, ... ,RSTUVWXYZ0   5ABCDEFGHI, ... ,RSTUVWXYZ5
1ABCDEFGHI, ... ,RSTUVWXYZ1   6ABCDEFGHI, ... ,RSTUVWXYZ6
2ABCDEFGHI, ... ,RSTUVWXYZ2   7ABCDEFGHI, ... ,RSTUVWXYZ7
3ABCDEFGHI, ... ,RSTUVWXYZ3   8ABCDEFGHI, ... ,RSTUVWXYZ8
4ABCDEFGHI, ... ,RSTUVWXYZ4   9ABCDEFGHI, ... ,RSTUVWXYZ9
================================================
 c:
0ZYXW, ... ,DCBA0    5ZYXW, ... ,DCBA5
1ZYXW, ... ,DCBA1    6ZYXW, ... ,DCBA6
2ZYXW, ... ,DCBA2    7ZYXW, ... ,DCBA7
3ZYXW, ... ,DCBA3    8ZYXW, ... ,DCBA8
4ZYXW, ... ,DCBA4    9ZYXW, ... ,DCBA9
 d:
0ABCD, ... ,WXYZ0    5ABCD, ... ,WXYZ5
1ABCD, ... ,WXYZ1    6ABCD, ... ,WXYZ6
2ABCD, ... ,WXYZ2    7ABCD, ... ,WXYZ7
3ABCD, ... ,WXYZ3    8ABCD, ... ,WXYZ8
4ABCD, ... ,WXYZ4    9ABCD, ... ,WXYZ9
 Время на      1000001 -кратный вызов swap_stchar равно   3.2515061
 c:
0ABCD, ... ,WXYZ0    5ABCD, ... ,WXYZ5
1ABCD, ... ,WXYZ1    6ABCD, ... ,WXYZ6
2ABCD, ... ,WXYZ2    7ABCD, ... ,WXYZ7
3ABCD, ... ,WXYZ3    8ABCD, ... ,WXYZ8
4ABCD, ... ,WXYZ4    9ABCD, ... ,WXYZ9
 d:
```

```
0ZYXW, ... ,DCBA0    5ZYXW, ... ,DCBA5
1ZYXW, ... ,DCBA1    6ZYXW, ... ,DCBA6
2ZYXW, ... ,DCBA2    7ZYXW, ... ,DCBA7
3ZYXW, ... ,DCBA3    8ZYXW, ... ,DCBA8
4ZYXW, ... ,DCBA4    9ZYXW, ... ,DCBA9
```

2.6.4 Сравнение временных затрат для -O0, -O1, -O2, -O3

Имя структуры	**dimension(n)**				**type st...**			
Опция	**-O0**	**-O1**	**-O2**	**-O3**	**-O0**	**-O1**	**-O2**	**-O3**
kmax=100000001								
Базовый тип								
integer(4)	8.674	1.816	1.715	1.513	8.427	1.815	1.714	1.463
real(4)	8.770	1.769	1.715	1.513	8.411	1.859	1.715	1.467
real(8)	8.060	1.816	1.715	1.414	8.068	1.614	1.766	1.729
real(10)	19.96	4.338	4.638	4.882	20.24	5.344	5.205	4.444
kmax=1000001								
character(1302)	3.062	3.068	3.286	3.310	3.252	3.014	2.996	3.039

2.7 Задача 6. Демонстрация перегрузки функций

Решается та же задача обмена, что и в **1-5**, но процедуру обмена необходимо оформить так, чтобы её вызов по имени **swap** был в состоянии выполнить работу любой из ранее используемых процедур обмена.
В задачах **1-5** имели дело с процедурами, решающими по сути одинаковые задачи, но с переменными разных типов, так что для работы с каждым конкретным типом использовалась отдельная процедура со своим специфическим именем.

Подобное часто встречается и в самих языках программирования. Так, ФОРТРАН-функции с именами **iabs**, **abs**, **dabs**, **cabs**, **cdabs** получают абсолютную величину для аргументов типа *integer, real(4), real(8), complex(4), complex(8)* соответственно.

Наши алгоритмы обмена по сути одинаковые, хотя и работают с разными типами данных. Тем не менее, ясно, что необходимость использовать специфическое имя, соответствующее типу, вряд ли удобна. Удобно использовать **одно общее имя**, переложив на *интеллект* компилятора выяснение вопроса:

какое именно специфическое имя соответствует типу и/или количеству аргументов,
обеспечивая вызов соответствующей встроенной функции по общему родовому
имени.

Подобная возможность называется перегрузкой функций.

Современный ФОРТРАН позволяет не только вызывать встроенные процедуры по родовому имени, но и сопоставлять родовое имя набору процедур пользователя, реализуя соответствующий механизм перегрузки процедур посредством именования интерфейсного блока.

Напоминания.

1. **Неименованный интерфейсный блок** (см. гл. **9; I-семестр**) служит для объявления в программе интерфейса процедур, которые эта программа вызывает. Цель — обеспечить через посредство компилятора контроль за соответствием количества и типов фактических аргументов процедуры количеству и типам формальных.

Если процедуры пишутся без использования многочисленных новых возможностей современного ФОРТРАНа, то наличие интерфейсного блока, конечно, обеспечит упомянутый автоматический контроль, хотя можно обойтись и без него, контролируя ситуацию лично.

Однако, использование новых ФОРТРАН-возможностей (к примеру, **function**, возвращающая через своё имя массив или имеющая необязательный аргумент) с необходимостью (в той или иной форме) потребует наличия соответствующего интерфейсного блока.

Основные формы явного задания интерфейса:

использование рамочного оператора **interface ... end interface** с указанием внутри **интерфейсов** (заголовков) всех нужных процедур. При этом, когда нужные процедуры могут потребоваться в нескольких программных единицах, описание **явного интерфейсного блока** выгодно поместить в отдельный файл, подсоединяемый к этим единицам посредством инструкции **include**;

использование единиц компиляции вида **module**, подсоединяемых к нужным программным единицам оператором **use**. В этом случае, если не нужны какие-то уточнения относительно интерфейса процедур, описанных в модуле, их модульный интерфейс оказывается явным по умолчанию.

2. **Именованный интерфейсный блок** (помимо упомянутого контроля) реализует и механизм перегрузки, сопоставляя всем указанным в нём прототипам процедур родовое имя — имя интерфейсного блока. **Именованный интерфейсный блок** (как и неименованный) задаётся явно либо посредством его описания в той единице компиляции, в которой хотим по одному родовому имени вызывать различные процедуры; либо посредством описания его в модуле перед **contains**. При этом в интерфейсном блоке не требуется полностью описывать интерфейсы процедур, вызываемых по одному родовому имени — достаточно указать лишь их имена (через запятую) в операторе **module procedure**, что наиболее практично.

2.7.1 Уяснение ситуации

1. Все процедуры, вызываемые главной программой в задачах **1-5** для решения задачи **6** описаны в модуле **mhome6**.

2. Оператор **use mhome6** (перед **implicit none** главной программы **test_6**) позволяет обратиться в ней к любой из процедур модуля, не описывая отдельно их интерфейс, который доступен через посредство оператора **use** по умолчанию.

3. Поэтому, единственное, что необходимо явно определить в модуле, так это родовое имя процедур обмена.

4. Приводимая ниже программа **test_6**) не вводит значения обмениваемых структур данных, а задаёт посредством инициализации. Тем самым, предоставляется возможность описать и набор соответствующих процедур инициализации (а также и вывода данных) сопоставив родовые имена каждой родственной группе процедур. Именно **swap** — для процедур обмена, **init** — для процедур инициализации, **prt** — для процедур вывода (последние не вызываются в главной программе, поскольку наша главная задача — оценка временны́х затрат при работе с разными структурами).

2.7.2 Make-файл

```
 comp:=gfortran
  opt:=-c -O0                          #-fdefault-real-8
  pattern:=*.f
  source :=$(wildcard $(pattern))
  obj    :=$(patsubst %.f, %.o, $(source))
  main:=main
  $(main) : $(obj)
$(comp) $^ -o $@
 %.o %.mod : %.f
           $(comp) $(opt) $<
 $(main).o : mhome6.mod
    result : $(main) input
       time ./$<   > result
clear :
rm -f *.o  *.mod $(main)
```

2.7.3 Главная программа

```
! ====================================================================
! Главная программа нацелена на оценку временных затрат процедур
! обмена при работе с разными структурами данных:
!  integer, dimension(10)        и type(sti)
!  real(4), dimension(10)        и type(st4)
!  real(8), dimension(10)        и type(st8)
! real(10), dimension(10)        и type(st10)
! character(1302), dimension(10) и type(stchar)
! Результат работы программы --- табличка
! #     Тип           Массивы        Структуры
!     integer       0.00   0.81      0.81   1.56
!     real(4)       1.56   2.34      2.34   3.06
!     real(8)       3.06   3.87      3.87   4.66
!    real(10)       4.66   5.80      5.80   6.90
! character(1302)   6.90  49.28     49.28  91.52
! Программа использует модуль mhome6, в котором описаны все
! используемые типы данных, все процедуры и задана константа
! kmax, определяющая кратность их вызова.
! Для оценки временных затрат используется встроенная
! подпрограмма ФОРТРАНа cpu_time (или что то же second)
! ....................................................................
      program test_6; use mhome6; implicit none
      integer        ai(n),  bi(n) ! описание массивов типа: integer
      real(4)        a4(n),  b4(n) !                         real(4)
      real(8)        a8(n),  b8(n) !                         real(8)
      real(10)      a10(n), b10(n) !                        real(10)
      character(1302) a(n),   b(n) !                 character(1302)
      type (sti)        ci, di     ! описание структур типа:     sti
      type (st4)        c4, d4     !                              st4
      type (st8)        c8, d8     !                              st8
      type (st10)      c10, d10    !                             st10
      type (stchar) ::   c, d      !                           stchar
      character(15), parameter :: tp(5)=(/'    integer    ',
     >       '    real(4)    ','    real(8)    ','   real(10)    ',
     >                                          'character(1302)'/)
      integer  :: i, k
      real t(5,4)   !                     i=1(1)5;i - номер задачи.
                    ! Отсчёты времени: t(i,1),t(i,2):  для массивов
                    !                  t(i,3),t(i,4):  для структур
! Задача 1:
      call init(ai,bi)
      call second(t(1,1)); do k=1, kmax; call swap(ai,bi); enddo
      call second(t(1,2))
      call init(ci,di)
      call second(t(1,3)); do k=1, kmax; call swap(ci,di); enddo
      call second(t(1,4))
```

```
! Задача 2:
      call init(a4,b4)
      call second(t(2,1)); do k=1, kmax; call swap(a4,b4); enddo
      call second(t(2,2))
      call init(c4,d4)
      call second(t(2,3)); do k=1, kmax; call swap(c4,d4); enddo
      call second(t(2,4))
! Задача 3:
      call init(a8,b8)
      call second(t(3,1)); do k=1, kmax; call swap(a8,b8); enddo
      call second(t(3,2))
      call init(c8,d8)
      call second(t(3,3)); do k=1, kmax; call swap(c8,d8); enddo
      call second(t(3,4))
! Задача 4:
      call init(a10,b10)
      call second(t(4,1)); do k=1, kmax; call swap(a10,b10); enddo
      call second(t(4,2))
      call init(c10,d10)
      call second(t(4,3)); do k=1, kmax; call swap(c10,d10); enddo
      call second(t(4,4))
! Задача 5:
      call init(a,b)
      call second(t(5,1)); do k=1, kmax; call swap(a,b); enddo
      call second(t(5,2))
      call init(c,d,a,b)
      call second(t(5,3)); do k=1, kmax; call swap(c,d); enddo
      call second(t(5,4))
      write(*,'(" #    Тип",12x,"Массивы",8x,"Структуры")')
      do i=1,5
        write(*,'(a,2(f7.2),"   ",2(f7.2))')
     >                          tp(i),(t(i,k),k=1,4)
      enddo
      end
```

2.7.4 Результат работы главной программы

```
 #    Тип            Массивы        Структуры
    integer       0.00   0.81      0.81   1.56
    real(4)       1.56   2.34      2.34   3.06
    real(8)       3.06   3.87      3.87   4.66
   real(10)       4.66   5.80      5.80   6.90
character(1302)   6.90  49.28     49.28  91.52
```

2.7.5 Модуль mhome6

```
      module mhome6; implicit none
      integer,parameter :: kmax=10000001! Число вызовов процедур обмена
      integer,parameter :: n=10       ! Число эл. в массиве и p-поле
      type  sti                      ! Определение производного типа
        integer, dimension(n) :: p   ! sti - структуры с одним
      end type sti                   ! полем p из n элементов
      type  st4                      ! Определение производного типа
        real(4), dimension(n) :: p   ! st4 - структуры с одним полем
      end type st4                   ! p из n элементов типа real(4)
      type  st8                      ! Определение производного типа
        real(8), dimension(n) :: p   ! st8 - структуры с одним полем
      end type st8                   ! p из n элементов типа real(8)
      type  st10                     ! Определение производного типа
        real(10), dimension(n) :: p ! st8 - структуры с одним полем
      end type st10                  ! p из n элементов типа real(8)
      type  stchar                   ! Определение производного типа
      character(1302) :: p(n)        ! stchar - структуры с одним
      end type stchar                ! полем p из n элементов
!                  Определение родового имени процедур инициализации:
      interface init
      module procedure init_ari, init_ar4, init_ar8, init_ar10,
     >                  init_sti, init_st4, init_st8, init_st10,
     >                 init_char, init_stchar
      end interface init
!                         Определение родового имени процедур обмена:
      interface swap
      module procedure swap_ari, swap_sti, swap_ar4, swap_st4,
     > swap_ar8, swap_st8, swap_ar10, swap_st10, swap_char, swap_stchar
      end interface swap
!                         Определение родового имени процедур вывода:
      interface prt
      module procedure prt_sti, prt_char, prt_stchar
      end interface prt
      contains
!                                     ! Процедуры инициализации:
! ============================================================
! Подпрограмма init_ari(x,y) инициализирует n элементов
! вектора x целыми значениями 1 ( 1)10, а
! вектора y целыми значениями 10(-1)1.
! ............................................................
      subroutine init_ari(x,y); integer x(n), y(n)
      integer i
      x=(/(i,i=1,n)/)
      y=(/(i,i=n,1,-1)/)
      end subroutine init_ari
```

```
! ================================================================
! Подпрограмма init_sti(x) инициализирует структуры x и y типа
! stint - структура данных с полем p типа integer, dimension(n).
! ................................................................
      subroutine init_sti(x,y); type (sti) x, y
      integer i
      x%p=(/(i,i=1,n)/)
      y%p=(/(i,i=n,1,-1)/)
      end subroutine init_sti
! ================================================================
! Подпрограмма init_ar4(x,y) инициализирует n элементов
! вектора x полуцелыми значениями 0.5( 1)9.5, а
! вектора y полуцелыми значениями 9.5(-1)0.5.
! ................................................................
      subroutine init_ar4(x,y); real(4) x(n), y(n)
      integer i
      x=(/(i-0.5,i=1,n)/)
      y=(/(i-0.5,i=n,1,-1)/)
      end subroutine init_ar4
! ================================================================
! Подпрограмма init_st4(x,y) инициализирует структуры x и y типа
! st4 - структура данных с полем p типа real(4), dimension(n).
! ................................................................
      subroutine init_st4(x,y); type (st4) x, y
      integer i
      x%p=(/(i-0.5,i=1,n)/)
      y%p=(/(i-0.5,i=n,1,-1)/)
      end subroutine init_st4
! ================================================================
! Подпрограмма init_ar8(x,y) инициализирует n элементов
! вектора x полуцелыми значениями 0.5_8( 1)9.5_8, а
! вектора y полуцелыми значениями 9.5_8(-1)0.5_8.
! ................................................................
      subroutine init_ar8(x,y); real(8) x(n), y(n)
      integer i
      x=(/(i-0.5_8,i=1,n)/)
      y=(/(i-0.5_8,i=n,1,-1)/)
      end subroutine init_ar8
! ================================================================
! Подпрограмма init_st8(x,y) инициализирует структуры x и y типа
! st8 - структурный тип данных с полем p типа real(8), dimension(n).
! ................................................................
      subroutine init_st8(x,y); type (st8) x, y
      integer i
      x%p=(/(i-0.5_8,i=1,n)/)
      y%p=(/(i-0.5_8,i=n,1,-1)/)
      end subroutine init_st8
```

```
! ================================================================
! Подпрограмма init_ar10(x,y) инициализирует n элементов
! вектора x полуцелыми значениями 0.5_10( 1)9.5_10, а
! вектора y полуцелыми значениями 9.5_10(-1)0.5_10.
! ................................................................
      subroutine init_ar10(x,y); real(10) x(n), y(n)
      integer i
      x=(/(i-0.5_10,i=1,n)/)
      y=(/(i-0.5_10,i=n,1,-1)/)
      end subroutine init_ar10
! ================================================================
! Подпрограмма init_st10(x,y) инициализирует структуры x и y типа
! st10 - структурный тип данных с полем p типа real(10), dimension(n).
! ................................................................
      subroutine init_st10(x,y); type (st10) x, y
      integer i
      x%p=(/(i-0.5_10,i=1,n)/)
      y%p=(/(i-0.5_10,i=n,1,-1)/)
      end subroutine init_st10
! ================================================================
! Подпрограмма init_char(x,y) заполняет каждый из элементов
! типа character(1302) вектора x(n)  пятьюдесятью копиями
! ASCII-латиницы в алфавитном порядке (50*26=1300), предваряя и
! завершая этот набор цифрой, обозначающей число равное
! номеру элемента без единицы.
! Вектор y(n) заполняется аналогично x, но в порядке обратном
! алфавитному.
! Цель процедуры - программное моделирование содержимого исходных
! векторов, чтобы не загромождать файл ввода.
! ................................................................
      subroutine init_char(x, y)
      character(1302) x(n), y(n)
      character(1300) w
      character(26) s; character(1) f; integer i, j, j13
      do i=1,n; w=''
        do j=1,26; s(j:j)=achar(iachar('A')+j-1); enddo
        do j=1,50
          w=trim(w)//s
        enddo
        f=achar(iachar('0')+i-1); x(i)=f//trim(w)//f
        do j=1,650
          j13=1300-j+1; f=w(j:j)
          w(j:j)=w(j13:j13)
          w(j13:j13)=f
        enddo
        f=achar(iachar('0')+i-1); y(i)=f//trim(w)//f
      enddo
      end subroutine init_char
```

```
! ================================================================
! Инициализация элементов производного типа stchar осуществляется
! копированием значений векторов x и y в соответствующее поле
! структур c и d. Копирование офоромлено процедурой, чтобы не
! загромождать главную программу подробностями иерархической
! структуры производного типа данных
! ................................................................
      subroutine init_stchar(c,d,x,y)
      character(1302) x(n), y(n)
      type (stchar) c, d
      c%p=x
      d%p=y
      end subroutine init_stchar
                                        ! Процедуры обмена:
! ================================================================
! Подпрограмма swap_ari(a,b,n) обменивает содержимым массивы
! a(n) и b(n), используя одну рабочую пееменную типа integer.
! ................................................................
      subroutine swap_ari(a,b); integer  a(n), b(n)
      integer i, w
      do i=1,n
        w=a(i); a(i)=b(i); b(i)=w
      enddo
      end subroutine swap_ari
! ================================================================
! Подпрограмма swap_sti(a,b) обменивает содержимым структуры
! a и b типа stint, используя одну рабочую переменную.
! stint - структура данных с полем p типа integer, dimension(n).
! ................................................................
      subroutine swap_sti(a,b); type (sti) a, b
      integer i, j
      do i=1,n
        j=a%p(i)
        a%p(i)=b%p(i)
        b%p(i)=j
      enddo
      end subroutine swap_sti
! ================================================================
! Подпрограмма swap_ar4(a,b,n) обменивает содержимым массивы
! a(n) и b(n), используя одну рабочую переменную типа real(4).
! ................................................................
      subroutine swap_ar4(a,b); real(4) a(n), b(n), w
      integer i
      do i=1,n
        w=a(i); a(i)=b(i); b(i)=w
      enddo
      end subroutine swap_ar4
```

```
! ====================================================================
! Подпрограмма swap_st4(a,b) обменивает содержимым структуры a и b
! типа st4, используя одну рабочую переменную типа real(4).
! st4 - структурный тип данных с полем p типа real(4), dimension(n).
! ....................................................................
      subroutine swap_st4(a,b); type (st4) a, b
      integer i; real(4) w
      do i=1,n
        w=a%p(i)
        a%p(i)=b%p(i)
        b%p(i)=w
      enddo
      end subroutine swap_st4
! ====================================================================
! Подпрограмма swap_ar8(a,b,n) обменивает содержимым массивы
! a(n) и b(n), используя одну рабочую переменную типа real(8).
! ....................................................................
      subroutine swap_ar8(a,b); real(8) a(n), b(n), w
      integer i
      do i=1,n
        w=a(i)
        a(i)=b(i)
        b(i)=w
      enddo
      end subroutine swap_ar8
! ====================================================================
! Подпрограмма swap_st8(a,b) обменивает содержимым структуры a и b
! типа st8, используя одну рабочую переменную типа real(8).
! st8 - структурный тип данных с полем p типа real(8), dimension(n).
! ....................................................................
      subroutine swap_st8(a,b); type (st8) a, b
      integer i; real(8) w
      do i=1,n
        w=a%p(i)
        a%p(i)=b%p(i)
        b%p(i)=w
      enddo
      end subroutine swap_st8
! ====================================================================
! Подпрограмма swap_ar10(a,b,n) обменивает содержимым массивы
! a(n) и b(n), используя одну рабочую переменную типа real(10).
! ....................................................................
      subroutine swap_ar10(a,b); real(10) a(n), b(n), w
      integer i
      do i=1,n
        w=a(i); a(i)=b(i); b(i)=w
      enddo
      end subroutine swap_ar10
```

```
! ================================================================
! Подпрограмма swap_st10(a,b) обменивает содержимым структуры a и b
! типа st10, используя одну рабочую переменную типа real(10).
! st10 - структурный тип данных с полем p типа real(10), dimension(n).
! ................................................................
      subroutine swap_st10(a,b); type (st10) a, b
      integer i; real(10) w
      do i=1,n
        w=a%p(i)
        a%p(i)=b%p(i)
        b%p(i)=w
      enddo
      end subroutine swap_st10
! ================================================================
! Подпрограмма swap_char(a,b) обменивает содержимым массивы
! a(n) и b(n) с элементами строкового типа character(1302),
! используя одну рабочую переменную типа character(1302).
! ................................................................
      subroutine swap_char(a,b); character(1302) a(n), b(n)
      integer i
      character(1302) w
      do i=1,n
        w=a(i); a(i)=b(i); b(i)=w
      enddo
      end subroutine swap_char
! ================================================================
! Подпрограмма swap_stchar(a,b) обменивает содержимым структуры
! a и b типа stchar, используя одну рабочую переменную.
! stchar - структура данных с полем p типа
!                                   character(1300),dimension(n).
! ................................................................
      subroutine swap_stchar(a,b); type (stchar) a, b
      character(1302) j; integer i
      do i=1,n
        j=a%p(i)
        a%p(i)=b%p(i)
        b%p(i)=j
      enddo
      end subroutine swap_stchar
                                              ! Процедуры вывода:
! ================================================================
! Подпрограмма prt_sti(x) предназначена для вывода значений типа
! stint - структура данных с полем p типа integer, dimension(n).
! ................................................................
      subroutine prt_sti(x); type (sti) x
      write(*,'(1000i4)') x%p
      end subroutine prt_sti
```

```
! ================================================================
! Подпрограмма prt_st4(x) предназначена для вывода значений типа
! st4 - структурный тип данных с полем p типа real(4), dimension(n).
! ................................................................
      subroutine prt_st4(x); type (st4) x
      write(*,'(10f6.1)') x%p
      end subroutine prt_st4
! ================================================================
! Подпрограмма prt_st8(x) предназначена для вывода значений типа
! st8 - структурный тип данных с полем p типа real(8), dimension(n).
! ................................................................
      subroutine prt_st8(x); type (st8) x
      write(*,'(10f6.1)') x%p
      end subroutine prt_st8
! ================================================================
! Подпрограмма prt_st10(x) предназначена для вывода значений типа
! st10 - структурный тип данных с полем p типа real(10), dimension(n).
! ................................................................
      subroutine prt_st10(x); type (st10) x
      write(*,'(10f6.1)') x%p
      end subroutine prt_st10
! ================================================================
!Подпрограмма prt_char(x) выводит первые n и последние n символы
! каждого из n типа character(1300) вектора x(n)
! ................................................................
      subroutine prt_char(x); character(1302) x(n)
      integer i
      do i=1,n/2
        write(*,'(2(a10,", ... ,",a10,"   "))')
     >  x(i)(1:10),x(i)(1293:1302), x(5+i)(1:10),x(5+i)(1293:1302)
      enddo
      end subroutine prt_char
! ================================================================
! Подпрограмма prt_stchar(x) предназначена для вывода значений типа
! stchar - структура данных с полем p типа
!                                  character(1300),dimension(n).
! ................................................................
      subroutine prt_stchar(x); type (stchar) x
      integer i
      write(*,'(2(a5,", ... ,",a5,"    "))') (x%p(i)(1:5),
     > x%p(i)(1298:1302), x%p(5+i)(1:5),x%p(5+i)(1298:1302),i=1,n/2)
      end subroutine prt_stchar
      end module mhome6
```

2.7.6 Использование профилировщика gprof

1. . Проведём компиляцию, активировав опцию -pg:

   ```
   gfortran -pg mhome6.f  main.f -o ./main
   ```

2. Активируем исполнимый файл:

   ```
   ./main  > res1
   ```

3. В текущей директории получаем файл **res1**:

   ```
    #    Тип            Массивы        Структуры
       integer       0.00   0.74      0.74   1.44
       real(4)       1.44   2.14      2.14   2.84
       real(8)       2.84   3.54      3.54   4.25
      real(10)       4.25   5.54      5.54   6.35
   character(1302)   6.35  27.75     27.75  48.34
   ```

   и файл с именем **gmon.out**.

4. Вызов профилировщика:

   ```
   gprof ./main gmon.out > flat.res
   ```

   ```
   Flat profile:

   Each sample counts as 0.01 seconds.
     %   cumulative   self              self     total
    time   seconds   seconds    calls   s/call   s/call  name
    16.02      1.37     1.37 10000001     0.00     0.00  __mhome6_MOD_swap_char
    16.02      2.74     1.37 10000001     0.00     0.00  __mhome6_MOD_swap_stchar
    14.38      3.97     1.23 10000001     0.00     0.00  __mhome6_MOD_swap_ar10
     8.07      4.66     0.69 10000001     0.00     0.00  __mhome6_MOD_swap_st10
     7.83      5.33     0.67 10000001     0.00     0.00  __mhome6_MOD_swap_ari
     7.54      5.98     0.65 10000001     0.00     0.00  __mhome6_MOD_swap_ar8
     7.48      6.62     0.64 10000001     0.00     0.00  __mhome6_MOD_swap_st4
     7.37      7.25     0.63 10000001     0.00     0.00  __mhome6_MOD_swap_ar4
     7.02      7.85     0.60 10000001     0.00     0.00  __mhome6_MOD_swap_sti
     6.72      8.43     0.58 10000001     0.00     0.00  __mhome6_MOD_swap_st8
     1.29      8.54     0.11        1     0.11     8.54  MAIN__
   ```

   Столбец **self seconds** — время (в секундах) затраченное только функцией, имя которой находится на правой границе текущей строки. Таким образом, **swap_char** на 10000001 вызовов затрачено 1.37

с (так решил **gprof**). В то же время, встроенная **second** оценила время в **27.75-6.35=21.40с** — поразительное рассогласование.

В чём дело?

Аналогичный результат и для **swap_stchar**: (48.34-27.75=20.59).

Процедура **swap_ar10** по мнению **gprof** работала **1.23с**, а согласно **second** — **1.29** с (хоть какое-то согласие есть)

Процедура **swap_st10** по **gprof** — 0.69с; а по **second** — 0.81 .

5. Пояснения содержимого остальных столбцов flat profile можно найти в разделе **12.7**.

3 Статические массивы (второй семестр).

- **Решения представить на ФОРТРАНе и C (fprintf, fscanf).**
- ФОРТРАН-процедуры:

  из задач с **1** по **4** разместить в модуле **poly**;

  из задач с **5** по **6** разместить в модуле **binrad**;

  из задачи **7** (функция **rectan**) разместить в модуле **quadra**.
- Главная программа должна подсоединять модуль, вводить исходные данные из файла, вызывать нужную процедуру и выводить результат в файл вывода.
- Инициировать запуск каждой программы должен **make**-файл, используя утилиту **time**.

1. Разработать процедуру, которая по заданным **n+1** коэффициентам полинома **n**-й степени и его аргументу **x** вычисляет значение полинома **P(x)**.
2. Разработать процедуру, которая по заданным **n+1** коэффициентам полинома **n**-й степени и его аргументу вычисляет **два** значения полинома (**P(x)** и **P(-x)**) практически за время расчёта одного значения **P(x)**.
3. Первые **n** элементов одномерного массива содержат **n** целых чисел из диапазона **[-16,16]**. Разработать процедуру, которая подсчитывает количество встреч в массиве каждого из чисел соответственно.
4. Разработать процедуру, которая по заданному целому находит его двоичное представление, помещая очередную двоичную цифру в соответствующий элемент одномерного массива (для отрицательных чисел представление должно быть получено в дополнительном коде).
5. Разработать процедуру, которая набор нулей и единиц, размещённых в элементах одномерного массива переводит в значение типа **integer**

6. Разработать процедуру, преобразующую набор двоичных цифр, размещённых в элементах одномерного массива, в набор восьмеричных цифр так, чтобы целые значения, изображаемые указанными наборами, численно были равны.

7. Значение интеграла по промежутку **[a,b]** от подинтегральной функции **f(x)** приближенно вычисляется через квадратурную сумму формулы **средних прямоугольников** (подробнее см. приложение **IV**):

$$\tilde{\mathbf{S}} = \mathbf{h} \cdot \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{f}(\mathbf{x_i}) \quad \text{где} \quad \mathbf{h} = \frac{\mathbf{b-a}}{\mathbf{n}}; \qquad \mathbf{x_i} = \mathbf{a} + \mathbf{h} \cdot (\mathbf{i} - \frac{\mathbf{1}}{\mathbf{2}}); \qquad \mathbf{i} = \mathbf{1}, \mathbf{2}, \cdots \mathbf{n}.$$

n – число промежутков равномерного дробления отрезка **[a,b]**.

Разработать функцию **rectan(y,a,b,n)**, которая по набору из **n** значений подинтегральной функции, хранящихся в первых **n** элементах одномерного массива **y** вычисляет значение квадратурной суммы формулы средних прямоугольников. Описание функции **rectan** поместить в модуль **quarda**.

3.1 Задача 1

Разработать процедуру, которая по заданным **n+1** коэффициентам полинома **n**-й степени и его аргументу **x** вычисляет значение полинома **P(x)**.

3.1.1 Тестирующая программа

```
! ============================================================
! Программа test1 тестирует функцию horner0(x,p,n) расчёта
! значения полинома n-ой степени
!                   P_n(x)=p(0)x^n+p(1)x^(n-1)+...+p(n-1)x+p(n)
! по вводимым значениям x (аргумент), n (показатель степени
! полинома) и вектора коэффициентов p(0:n).
! horner0 - вариант с передачей показателя степени полинома
!           (или, что то же индекса свободного коэффициента).
! horner1 - вариант с определением показателя степени внутри
!           функции.
! ............................................................
      program test1
      use my_prec; use poly
      implicit none
      real(mp) x, p(0:nmax), r0, r1
      real(mp), allocatable :: q(:)
      integer n, i, ier
      real t0, t1, t2
      read (*,'(i10/(e10.3))') n, x, (p(i),i=0,n)
      write(*,'(" # n=",i3,5x,"  x=",e15.7)') n,x
      write(*,'(" # i",6x,"p(i)"/(i4,e15.7))') (i,p(i),i=0,n)
      allocate(q(0:n), stat=ier)
      q=p(0:n)
      if (ier/=0) stop 1
      call second(t0); do i=1, kmax; r0=horner0(x,p,n); enddo
      call second(t1); do i=1, kmax; r1=horner1(x,q);   enddo
      call second(t2)
      write(*,*) ' r0=',r0, 'kmax=',kmax,' time=',t1-t0
      write(*,*) ' r1=',r1, 'kmax=',kmax,' time=',t2-t1
      stop 0
      end
```

1. **horner0**. Три формальных аргумента. Наличие третьего (**n**) удобно описанием формального вектора коэффициентов в виде **real(mp) p(0:n)**, что позволяет функции использовать указанные размер и индексацию вектора.

   Недостаток: лишний параметр — лишняя возможность ошибиться.

Если для формального аргумента использовать вместо описания **p(0:n)**, описание **p(0:)** (что тоже возможно), то при вызове внутри **horner0** функции **n=ubound(p,1)** получим **n=nmax=100**.

2. **horner1**. Два формальных аргумента (меньше ошибок).

   Недостаток: вынуждены в программе, вызывающей **horner1**, размер фактического вектора коэффициентов, указывать точно таким, каким он должен быть на самом деле, что иногда неудобно.

3.1.2 Исходный текст модулей my_prec и poly

```
module my_prec; implicit none; integer, parameter ::  mp=4
end module my_prec

module poly; use my_prec; implicit none
integer, parameter :: nmax=100, kmax=100000000
contains
function horner0(x,p,n) result(s)
integer i, n; real(mp) x, p(0:n), s
s=p(0); do i=1,n; s=s*x+p(i); enddo
end function horner0
function horner1(x,p) result(s); real(mp) x, p(0:), s
integer n, i
n=ubound(p,1); s=p(0); do i=1,n; s=s*x+p(i); enddo
end function horner1
end module poly
```

3.1.3 Вывод тестирующей программы (-O0)

```
 # n=  2       x=  0.5000000E+00
 # i      p(i)
   0  0.1000000E+01
   1  0.2000000E+01
   2  0.3000000E+01
  r0=   4.250000     kmax=   100000000  time=   2.883561
  r1=   4.250000     kmax=   100000000  time=   5.233205
```

3.1.4 Вывод тестирующей программы (-O3)

```
 # n=  2       x=  0.5000000E+00
 # i      p(i)
   0  0.1000000E+01
   1  0.2000000E+01
   2  0.3000000E+01
  r0=   4.250000     kmax=   100000000  time=   1.009846
  r1=   4.250000     kmax=   100000000  time=   1.518769
```

3.2 Задача 2

Разработать процедуру, которая по заданным **n+1** коэффициентам полинома **n**-й степени и его аргументу вычисляет **два** значения полинома (**P(x)** и **P(-x)**) практически за время расчёта одного значения **P(x)**.

3.2.1 Уяснение ситуации

Конечно, двукратный вызов процедуры **horner0** из задачи **№1**:

```
rp=horner0( x,p,n)
rm=horner0(-x,p,n)
```

обеспечит расчёт искомых значений, но при этом количество выполняемых действий (а, значит, и временные затраты) будет вдвое бóльшим, чем требуется для одного вызова **horner0**. Условие задачи **№2** требует, чтобы и число действий, и временные затраты практически соответствовали однократному вызову процедуры **horner0**, но при этом было вычислено два значения: $\mathbf{P_n(x)}$ и $\mathbf{P_n(-x)}$.

Выполнить требуемое можно, если учесть, что $\mathbf{x^2} = (-\mathbf{x})^2$. Все слагаемые полинома распределяются на два набора: с чётными и нечётными показателями степени аргумента. Из набора слагаемых с нечётными степенями можно вынести множитель **x**. получив в скобках набор слагаемых опять-таки только с чётными прказателями. Если показатель степени полинома чётен, то число слагаемых с чётными показателями на единицу больше числа слагаемых с нечётными. Если же показатель степени полинома нечётен, то число слагаемых с чётными и нечётными степенями одинаково и равно $\mathbf{n/2}$. Обозначим через $\mathbf{S_1}$ сумму всех слагаемых с чётностью показателя степени полинома, а через $\mathbf{S_2}$ сумму слагаемых с противоположной чётностью показателя. Тогда, если **n** — чётен, то

$$\mathbf{P_n}(|\mathbf{x}|) = \mathbf{s1} + |\mathbf{x}| * \mathbf{S_2} \text{ и } \mathbf{P_n}(-|\mathbf{x}|) = \mathbf{s1} - |\mathbf{x}| * \mathbf{S_2},$$

а, если нечётен, то

$$\mathbf{P_n}(|\mathbf{x}|) = \mathbf{S_2} + |\mathbf{x}| * \mathbf{S_1} \text{ и } \mathbf{P_n}(-|\mathbf{x}|) = \mathbf{S_2} - |\mathbf{x}| * \mathbf{S_1},$$

3.2.2 Тестирующая программа

```
! ============================================================
! Программа по вводимым значениям аргумента (x), показателя
! степени полинома (n) и вектора коэффициентов p(0:n):
!           P_n(x)=p(0)x^n+p(1)x^(n-1)+...+p(n-1)x+p(n)
! тестирует функции horner0(x,p,n) и horner2(x,p,n) на предмет
! временных затрат по расчёту P_n(x) и P_n(-x) (двух значений).
! <<horner0>> вызывается дважды (с аргументами x и -x).
! <<horner2>> --- за один вызов и число сложений и умножений,
!             практически равное количеству тех же операций,
! выполняемых horner0, тем не менее, вычисляет сразу два
! значения и P_n(x), и P_n(-x).
! ............................................................
      program test2; use my_prec; use poly; implicit none
      real(mp) x, p(0:nmax), rp0, rm0, r(2)
      integer n, i, ier; real t0, t1, t2
      read (*,'(i10/(e10.3))') n, x, (p(i),i=0,n)
      write(*,'(" # n=",i3,5x,"  x=",e15.7)') n,x
      write(*,'(" # i",6x,"p(i)"/(i4,e15.7))') (i,p(i),i=0,n)
      call second(t0); do i=1, kmax; rp0=horner0( x,p,n)
                                     rm0=horner0(-x,p,n)
                     enddo
      call second(t1); do i=1, kmax; r=horner2(x,p,n); enddo
      call second(t2)
      write(*,'(a,e15.7,a,e15.7,a,i10,a,f7.3)')
     > '  rp0=', rp0, '  rm0=', rm0, ' kmax=',kmax,' time=',t1-t0
      write(*,'(a,e15.7,a,e15.7,a,i10,a,f7.3)')
     > ' r(1)=',r(1),' r(2)=',r(2), ' kmax=',kmax,' time=',t2-t1
      stop 0
      end
```

3.2.3 Результаты пропуска при n=9 (-O0)

```
 # n=  9       x=  0.1000000E+01
 # i      p(i)
   0  0.1000000E+01
   1  0.2000000E+01
   2  0.3000000E+01
   3  0.4000000E+01
   4  0.5000000E+01
   5  0.6000000E+01
   6  0.7000000E+01
   7  0.8000000E+01
   8  0.9000000E+01
   9  0.1000000E+02
  rp0=  0.5500000E+02  rm0=  0.5000000E+01 kmax= 100000000 time= 18.860
 r(1)=  0.5500000E+02 r(2)=  0.5000000E+01 kmax= 100000000 time=  9.217
```

3.2.4 Исходный текст модулей my_prec и poly

```
module my_prec; implicit none; integer, parameter ::  mp=4
end module my_prec
```

```
module poly; use my_prec; implicit none
integer, parameter :: nmax=100, kmax=100000000
contains
function horner0(x,p,n)
integer i, n; real(mp) horner0, x, p(0:n), s
s=p(0); do i=1,n; s=s*x+p(i); enddo; horner0=s
end function horner0
function horner1(x,p); real(mp) horner1, x, p(0:), s
integer n, i
n=ubound(p,1); s=p(0); do i=1,n; s=s*x+p(i); enddo
horner1=s
end function horner1
function horner2(x,p,n)
integer i, n; real(mp) horner2(2), x, p(0:n), s1, s2, ax, x2
ax=abs(x)
select case (n)
 case(0); horner2(1)=p(0); horner2(2)=p(0); return
 case(1); horner2(1)=p(1)+ax*p(0); horner2(2)=p(1)-ax*p(0);return
 case default; x2=x*x; s1=p(0); do i=2,n,2; s1=s1*x2+p(i); enddo
                       s2=p(1); do i=3,n,2; s2=s2*x2+p(i); enddo
   if (mod(n,2)==0) then; horner2(1)=s1+ax*s2;horner2(2)=s1-ax*s2
                    else; horner2(1)=s2+ax*s1;horner2(2)=s2-ax*s1
   endif
end select
end function horner2
end module poly
```

3.2.5 Вывод тестирующей программы при n=9 (-O3)

```
 # n=  9       x=  0.1000000E+01
 # i      p(i)
   0  0.1000000E+01
   1  0.2000000E+01
   2  0.3000000E+01
   3  0.4000000E+01
   4  0.5000000E+01
   5  0.6000000E+01
   6  0.7000000E+01
   7  0.8000000E+01
   8  0.9000000E+01
   9  0.1000000E+02
  rp0=  0.5500000E+02  rm0=  0.5000000E+01 kmax= 100000000 time=  7.033
 r(1)=  0.5500000E+02 r(2)=  0.5000000E+01 kmax= 100000000 time=  3.785
```

3.3 Задача 3

Первые **n** элементов одномерного массива содержат **n** целых чисел из диапазона **[-16,16]**. Разработать процедуру, которая подсчитывает количество встреч в массиве каждого из чисел соответственно.

3.3.1 Уяснение ситуации

Если по условию задачи гарантировано отсутствие в векторе **p** чисел вне диапазона **[-16:16]**, то значение любого элемента вектора **p** можно трактовать как индекс вектора-счётчика. Тогда для подсчёта количества встреч каждого из чисел вектора **p** достаточно лишь раз их перебрать:

```
do i=1,n
  k(p(i))=k(p(i))+1
enddo
```

3.3.2 Тестирующая программа

Для усвоения возможностей современного ФОРТРАНа опишем несколько процедур подсчёта:

1. **subroutine howmany0_s(p,n,k)**, т.е. оформим алгоритм подпрограммой с тремя аргументами. **p(n)** — анализируемый вектор, **n** — число его элементов; **k(-16:16)** — искомый вектор-счётчик. Старый ФОРТРАН не допускал возвращать через имя функции массив. Для возвращения последнего требовался дополнительный аргумент (**k**).
2. **function howmany0(p,n,k)** — находит **k** тем же способом, что и **howmany_s**, но через своё имя возвращает **k(0)**.
3. **function howmany1(p,n)** — функция с **двумя** аргументами, возвращающая через свое имя массив (современный ФОРТРАН).
4. **function howmany2(p)** — оформляем алгоритм функцией с **одним** аргументом (зачем подавать на вход число элементов массива **p**, если его можно узнать внутри функции).
5. **function howmany** — оформляем алгоритм функцией с именем **howmany**, которая описана так, что позволяет по одному этому имени вызвать любую из описанных выше **function** (в соответствии с формой вызова).

```
! ============================================================
! Программа test3 тестирует подпрограмму howmany0_s(a,n,k) и
! функции howmany0(a,n,k), howmany1(a,n) и howmany2(a), которые
! находят число встреч в массиве a(n) каждого числа из [-16:16].
! ............................................................
      program test3; use poly; implicit none
      integer, allocatable :: p(:)
      integer  ks(-16:16), k0 (-16:16),  k1(-16:16),  k2(-16:16)
      integer k00(-16:16), k11(-16:16), k22(-16:16)
      integer n, i, ier, kz; logical q
      read (*,'(i10)') n;      write(*,'(" # n=",i5)') n
      allocate(p(n),stat=ier); if (ier/=0) stop 1
      write(*,'(" #  N",4x,"p")')
      do i=1,n; p(i)=mod(irand(0),33)-16; enddo
      write(*,'(a,9i3/a,9i3)') ' # N:',(i, i=1,n), ' # p:',(p(i),i=1,n)

      call howmany0_s(p,n, ks);
       kz=howmany0(p,n,k0);   k1=howmany1(p,n);   k2=howmany2(p)
      k00=howmany (p,n,k00); k11=howmany (p,n);  k22=howmany (p)
      write(*,'(" # kz=",I5)') kz
      write(*,'(" # i",2x,"ks",2x,"k0",2x,"k1",2x,"k2",5x,
     >                    "k00",2x,"k11",2x,"k22")')
      do i=-16,16
        q= k0(i)/=0.and.k1(i)/=0.and.k2(i)/=0
        if (q) write(*,'(5i4,2x,3i5)') i,ks(i), k0 (i), k1 (i), k2 (i),
     >                             k00(i), k11(i), k22(i)
      enddo
      end
```

3.3.3 Вывод тестирующей программы (-O3)

```
 # n=    9
 #  N    p
 # N:  1  2  3  4  5  6  7  8  9
 # p: -6-15 -5  1  3 -2  8 -5  4
 # kz=    0
 # i  ks  k0  k1  k2     k00  k11  k22
 -15   1   1   1   1      0    1    1
  -6   1   1   1   1      0    1    1
  -5   2   2   2   2      0    2    2
  -2   1   1   1   1      0    1    1
   1   1   1   1   1      0    1    1
   3   1   1   1   1      0    1    1
   4   1   1   1   1      0    1    1
   8   1   1   1   1      0    1    1
```

3.3.4 Исходный текст модулей my_prec и poly

```
      module my_prec
      implicit none
      integer, parameter ::  mp=4
      end module my_prec
```

Ниже, в исходном тексте модуля **poly** опущены процедуры, не имеющие отношения к задаче **№3**:

```
      module poly; use my_prec; implicit none
      interface howmany
        module procedure howmany0, howmany1, howmany2
      end interface howmany
      integer, parameter :: nmax=100, kmax=100000000
      contains
!   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
      subroutine howmany0_s(a,n,k)
      integer n, a(n), k(-16:16)
      integer m, i
      k=0; do i=1,n; m=a(i); k(m)=k(m)+1; enddo
      end subroutine howmany0_s
      function howmany0(a,n,k) result(kk)
      integer n, a(n), k(-16:16), kk
      integer m, i
      k=0; do i=1,n; m=a(i); k(m)=k(m)+1; enddo
      kk=k(0)
      end function howmany0
      function howmany1(a,n) result(kk)
      integer n, a(n), kk(-16:16)
      integer m, i
      kk=0; do i=1,n; m=a(i); kk(m)=kk(m)+1
      enddo
      end function howmany1
      function howmany2(a) result(kk)
      integer a(:), kk(-16:16)
      integer n, m, i
      n=ubound(a,1)
      kk=0
      do i=1,n
         m=a(i)
         kk(m)=kk(m)+1
      enddo
      end function howmany2
      end module poly
```

3.4 Задача 4

Разработать процедуру, которая по заданному целому находит его двоичное представление, помещая очередную двоичную цифру в соответствующий элемент одномерного массива (для отрицательных чисел представление должно быть получено в дополнительном коде).

3.4.1 Уяснение ситуации

ЭВМ-представление значений целого типа зависит от знака данного:

1. Для представления неотрицательных данных используется **прямой код**, который отличается от обычной записи целого в десятичной системе счисления лишь основанием системы. Например,

$$(14)_{10} = 1 * 10^1 + 4 * 10^0$$
$$(1110)_2 = 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0 = 8 + 4 + 2 = 14$$

2. Для машинного представления отрицательных целых используется **дополнительный код** (отличный от прямого; см. Раздел **4.4**; лекции 1-го семестра»). Так в однобайтовой переменной двоичные записи чисел **14** и **-14** будут иметь вид:

```
76543210<-- номер двоичного разряда (бита)
!<------  старший бит хранит знак числа: 0 --- плюс, 1 --- минус
00001110<-- +14 (в прямом коде)
11110010<-- -14 (в дополнительном)
```

3.4.2 Исходный текст главной программы

```
program test_3_4; use my_poly; implicit none
integer n, nb(0:31), i, k
read (*,*) n; write(*,*) ' n=', n
call decbinn(n,nb,k);
write(*,'(" decbinn      : n=",32i1)') (nb(i),i=31,0,-1)
write(*,'(" format    b32: n=",b32)') n
write(*,'(" format b32.32: n=",b32.32)') n
n=-n
write(*,'(" n=",i11)') n
call decbinn(n,nb,k);
write(*,'(" decbinn   :    n=",32i1)') (nb(i),i=31,0,-1)
write(*,'(" format b32:    n=",b32)') n
write(*,'(" format b32.32: n=",b32.32)') n
end
```

Среди спецификаторов оператора **format** есть спецификатор **Bw[.d]**, который позволяет выводить целое значение в двоичном виде. Если после литеры **B** указывается только ширина поля, то незначащие старшие нули выводятся пробелами; если же указывается **b32.32**, то — цифрами **0**.

```
module my_poly; implicit none
contains
subroutine decbinn(nn,nb,k); integer nn, n, nb(0:31), i, k
nb=0                                  ! Инициализация результата нулями
if (nn.eq.-huge(0)-1) then; nb(31)=1 ! или единицами.
  else; n=abs(nn)       ! Ищем прямой код модуля отрицательного числа.
     i=0                ! Индекс элемента под самую младшую цифру.
    do while (n/=0)                ! Пока число не оказалось нулём
      nb(i)=mod(n,2); n=n/2;i=i+1 ! заполняем текущий i-ый элемент
    enddo                          ! очередной двоичной цифрой
    k=i-1                          ! Количество двоичных цифр в числе
    if (nn.lt.0) then              ! Если исходное число < 0, то
      nb=1-nb                      ! находим обратный код и младший
      i=0; nb(i)=nb(i)+1           !        элемент дополнительного.
      do while ((nb(i)==2))        ! Пока текущий элемент == 2
        nb(i)=0                    ! обнуляем его,
        i=i+1                      ! находим индекс соседнего старшего,
        nb(i)=nb(i)+1              ! и увеличиваем значение соседа
      enddo                        !   на "один в уме".
    endif
endif
end subroutine decbinn
end module my_poly
```

Результат тестирования программы test_3_4

```
 n=         -14
decbinn      : n=11111111111111111111111111110010
format    b32: n=11111111111111111111111111110010
format b32.32: n=11111111111111111111111111110010
n=         14
decbinn   :    n=00000000000000000000000000001110
format b32:    n=                            1110
format b32.32: n=00000000000000000000000000001110

 n=  2147483647
decbinn      : n=01111111111111111111111111111111
format    b32: n= 1111111111111111111111111111111
format b32.32: n=01111111111111111111111111111111
n=-2147483647
decbinn   :    n=10000000000000000000000000000001
format b32:    n=10000000000000000000000000000001
format b32.32: n=10000000000000000000000000000001
```

3.5 Задача 5

Разработать процедуру, которая набор нулей и единиц, размещённых в элементах одномерного массива, переводит в значение типа **integer**.

Легко сообразить, что в условии сформулирована задача, которая является обратной по отношению к предыдущей. Поэтому, дополнив решение последней (после вывода результата процедуры **decbinn**) вызовом процедуры **bindecn**, получаем очевидную возможность проверить правильность её работы: результат **bindecn** должен будет совпасть с числом, подаваемым на вход **decbinn**.

```
program test_3_5; use my_poly; implicit none
integer n, nb(0:31), i, k
read (*,*) n; write(*,*) '1) n=', n
call decbinn(n,nb,k);
write(*,*) ' k=',k
write(*,'(" decbinn      : n=",32i1)') (nb(i),i=31,0,-1)
write(*,'(" format    b32: n=",b32)') n
write(*,'(" format b32.32: n=",b32.32)') n
n=bindecn(nb,k)
write(*,*) 'bindecn      : n=',n
n=-n
write(*,*) '2) n=', n
call decbinn(n,nb,k);
write(*,*) ' k=',k
write(*,'(" decbinn      : n=",32i1)') (nb(i),i=31,0,-1)
write(*,'(" format b32   : n=",b32)') n
write(*,'(" format b32.32: n=",b32.32)') n
n=bindecn(nb,k)
write(*,*) 'bindecn      : n=',n
end
```

Результаты работы test_3_5:

```
1) n=         -14
 k=           3
decbinn      : n=11111111111111111111111111110010
format    b32: n=11111111111111111111111111110010
format b32.32: n=11111111111111111111111111110010
bindecn      : n=         -14
2) n=          14
 k=           3
decbinn      : n=00000000000000000000000000001110
format b32   : n=                              1110
format b32.32: n=00000000000000000000000000001110
bindecn      : n=          14
```

Содержимое модуля my_poly

```
module my_poly; implicit none;  contains
subroutine decbinn(nn,nb,k); integer nn, n, nb(0:31), i, k
nb=0            ! Начальная инициализация результата нулями
if (nn.eq.-huge(0)-1) then; nb(31)=1
  else          ! Если число не наименьшее из отрицательных, то
     n=abs(nn)  ! получаем прямой код неотрицательного числа.
     i=0        ! Индекс элемента для хранения самой младшей цифры.
    do while (n/=0)                ! Пока число не оказалось нулём
      nb(i)=mod(n,2); n=n/2;i=i+1  ! заполняем текущий i-ый элемент
    enddo                          ! очередной двоичной цифрой
    k=i-1                        ! Количество двоичных цифр в числе
    if (nn.lt.0) then            ! Если исходное число < 0, то
      nb=1-nb                    !   находим обратный код и
      i=0; nb(i)=nb(i)+1         !   младший элемент дополнительного.
      do while ((nb(i)==2))      ! Пока текущий элемент == 2
        nb(i)=0                  ! обнуляем его,
        i=i+1                    ! находим индекс соседнего старшего,
        nb(i)=nb(i)+1            ! и увеличиваем значение соседа
      enddo                      !   на "один в уме"
    endif
endif
end subroutine decbinn
function bindecn(nb,k) result (m); integer nb(0:31), m, k, s, i
if (k==-1) then; m=0; return; endif
if (nb(31)==0) then; s=1      ! Запоминаем в s  знак числа
  else; s=-1; i=0             ! В случае дополнительного кода
    do while (i<=30)          ! Если все
      if (nb(i)/=0) exit      !           элементы
      i=i+1                   !                      массива
    enddo                     ! кроме 31-го нули,
    if (i==31) then           ! то
      m=-huge(0)-1; return    !    m=-2147483648
    endif                     ! В противном случае получаем
    nb(0)=nb(0)-1             ! обратный код, и пока его  i-ый
    i=0; do while (nb(i)==-1) ! разряд должен "занимать" у
           nb(i)=1            ! соседнего старшего 1, то
           i=i+1              ! занимаем её, вычисляем индекс
           nb(i)=nb(i)-1      ! последнего и его значение.
         enddo
    nb=1-nb                   ! Получаем из обратного кода прямой.
endif
m=nb(k)                         ! Подготовка к применению схемы
do i=k-1,0,-1; m=m*2+nb(i); enddo ! Горнера и расчёт по ней числа
m=isign(m,s)                    ! Присваивание знака результату.
end function bindecn
end module my_poly
```

Некоторые частные случаи.

```
1) n=           0
 k=          -1
decbinn      : n=00000000000000000000000000000000
format    b32: n=                                0
format b32.32: n=00000000000000000000000000000000
bindecn      : n=           0
2) n=           0
 k=          -1
decbinn      : n=00000000000000000000000000000000
format b32   : n=                                0
format b32.32: n=00000000000000000000000000000000
bindecn      : n=           0
```

- Когда значение, подаваемое на обработку процедуре **decbinn** равно нулю, то значение **k** (номер двоичного разряда, заполняемого алгоритмом перевода), являющееся одним из результатов **decbinn**, оказывается равным **-1**, что соответствует простому обнулению элементов массива, хранящих значения двоичных цифр (без их дальнейшего уточнения).

  Это значение **k=-1**, поданное на вход процедуры **bindecn**, служит признаком, что результату обратного перевода можно сразу присвоить нулевое значение.

- Если массив **nb** описать как **integer nb(-1:31)**, используя элемент **nb(-1)** для хранения нуля, то обошлись бы без условного оператора **if (k==-1) ...**.

```
1) n= -2147483647
 k=          30
decbinn      : n=10000000000000000000000000000001
format    b32: n=10000000000000000000000000000001
format b32.32: n=10000000000000000000000000000001
bindecn      : n= -2147483647
2) n=  2147483647
 k=          30
decbinn      : n=01111111111111111111111111111111
format b32   : n= 11111111111111111111111111111111
format b32.32: n=01111111111111111111111111111111
bindecn      : n=  2147483647
```

На типе **integer(4)** максимальное целое равно **2147483647**, а минимальное целое равно **-2147483648** (*Почему так?* объяснялось в первом семестре). Когда ввели максимально большое (но корректное) на типе **integer(4)** число — **2147483647**, то и результат оказался корректным.

```
1) n= -2147483648
 k=           0
decbinn      : n=10000000000000000000000000000000
format    b32: n=10000000000000000000000000000000
format b32.32: n=10000000000000000000000000000000
bindecn      : n= -2147483648
2) n= -2147483648
 k=           0
decbinn      : n=10000000000000000000000000000000
format b32   : n=10000000000000000000000000000000
format b32.32: n=10000000000000000000000000000000
bindecn      : n= -2147483648
```

Но когда в качестве входного данного берём минимальное из всех отрицательных **-2147483648**, то алгоритм расчёта соответствующего положительного (корректный для диапазона **[-1:-2147483647]**) оказывается некорректным, т.к. значение **2147483648**, полученное оператором **n=-n** главной программы не может быть записано в прямом коде ввиду того, что после перевода его в двоичную систему счисления оно представляется одной единственной единицей в самом старшем (31-ом) бите четырёхбайтовой ячейки, который трактуется как знак числа, а не как значащая цифра. В результате всё двоичное представление трактуется как дополнительный код, которым записано отрицательное целое.

- Заметим, что решение задач 4 и 5 может быть дано и с привлечением побитовых операций.
- Решение, данное выше, преследует цель развития у учащихся навыков программирования по теме «Одномерный массив», которая и привлекается для решения задачи, помогая лучше уяснить изучаемый материал.

3.6 Задача 6

Разработать процедуру, преобразующую набор двоичных цифр, размещённых в элементах вектора, в набор восьмеричных цифр так, чтобы целые значения, изображаемые этими наборами, численно были равны.

3.6.1 Уяснение ситуации

1. Решение, можно оформить на базе задачи 5. Действительно, коли получили в виде вектора двоичную модель целого числа, то этот же вектор можно использовать и для получения восьмеричной модели.

2. Конечно, число, введённое в десятичной системе счисления, можно перевести в восьмеричную по известному алгоритму. Однако задача требует осуществить непосредственный перевод из двоичной записи в восьмеричную, не привлекая десятичную в качестве посредника.

3. Перевод из двоичной в восьмеричную прост, если заметить, что любая восьмеричная цифра всегда представима тремя двоичными разрядами ($\mathbf{8 = 2^3}$). Поэтому достаточно двоичное представление разбить на последовательные тройки битов, а затем найти соответствующую восьмеричную цифру по формуле

$$\mathbf{C_8(k) = C_2(3k + 2) \cdot 2^2 + C_2(3k + 1) \cdot 2^1 + C_2(3k) \cdot 2^0}$$

   - $\mathbf{k}$ — номер восьмеричной цифры (или номер очередной тройки битов). Восьмеричную цифру из разряда восьмеричных единиц получим из двоичных разрядов **2**, **1** и **0** (при условии, что вектор, моделирующий двоичное число, имеет индексы из диапазона **[0,31]**. Восьмеричную цифру из разряда восьмеричных десятков получим из **5**-го, **4**-го и **3**-го битов, и т.д.
   - $\mathbf{C_8(k)}$ целочисленный аналог **k**-ой восьмеричной цифры;
   - $\mathbf{C_2(j)}$ целочисленный аналог **j**-ой двоичной цифры;
   - **k**-ая восьмеричная цифра ($\mathbf{k \in [0, 10]}$) получается из двоичных с номерами **3k**, **3k+1** и **3k+2**, кроме случая **k=10**, для которого бита с индексом **3k+2=32** нет, поскольку самый старший бит двоичного представления имеет индекс **31**. Так что

$$\mathbf{C_8(10) = C_2(31) \cdot 2^1 + C_2(30) \cdot 2^0}$$

4. Похожий перевод возможен и в шестнадцатеричную систему счисления с той лишь разницей, что одну шестнадцатеричную цифру получаем из **тетрады** битов (из четырёх двоичных разрядов, $\mathbf{16 = 2^4}$):

$$\mathbf{C_{16}(k) = C_2(4k+3) \cdot 2^3 + C_2(4k+2) \cdot 2^2 + C_2(4k+1) \cdot 2^1 + C_2(4k)}$$

5. Правда, в случае восьмеричной системы счисления в качестве модели восьмеричной цифры всегда возможно однозначное целое число. В шестнадцатеричной системе — шестнадцать цифр. Для первых десяти тоже допустимо использовать однозначные целые от **0** до **9**. Однако для оставшихся шести потребовались бы двузначные числовые аналоги **10**, **11**, **12**, **13**, **14**, **15**. Как известно, соответствующие цифры в шестнадцатеричной системе счисления принято обозначать буквами латиницы **A**, **B**, **C**, **D**, **E** и **F**. Таким образом, если в случае восьмеричной системы ещё можно было в качестве модели восьмеричного числа использовать массив типа **integer**, то в случае шестнадцатеричной придётся использовать массив символьного типа, например, **character(1) ch(0:7)**.

6. В случае перевода в шестнадцатеричную систему счисления двоичной модели числа типа **integer(4)** не нужно выделять особо получение самой старшей цифры **ch(7)**, т.к. тип **integer(4)** сосоит из целого числа тетрад.

7. В приведённых ниже исходных текстах модуля и тестирующей программы наряду с подпрограммой **binoctn** и её вызовом помещены текст и вызов подпрограммы **binhexn**, которая получает соответствующую шестнадцатеричную модель.

8. В исходном тексте **binhexn** использованы встроенные функции с именами **achar** и **iachar**, посредством которых можно, в частности, из двузначных числовых эквивалентов шестнадцатеричных цифр получать сами цифры, т.е. значки цифр и соответствующие буквы латиницы.

9. Помним, что среди множества символов **ASCII**-таблицы цифры упорядочены по возрастанию, а буквы латиницы расположены друг за другом в алфавитном порядке. Поэтому, **achar(iachar('0')+7)** получает цифру **7**, а **achar(iachar('A')+15-10)** получает цифру **F**.

10. Функция **iachar('A')** получает целочисленный код литеры **A**. Подробнее о работе с функциями обработки символьными переменных будем знакомиться в третьем семестре.

11. В тестирующей программе наряду с выводом векторной модели числа помещены и результаты перевода этого числа в восьмеричную и шестнадцатеричную системы счисления посредством соответствующих форматных дескрипторов **o11.11** и **z8.8**.

12. Последние два дескриптора можно использовать в двух формах, например, **o11** и **z8** или **o11.11** и **z8.8**. В первом случае после литеры дескриптора указывается только ширина поля вывода, что соответствует замене всех незначащих нулей пробелами. Во втором случае незначащие нули выводятся нулями.

3.6.2 Содержимое модуля my_poly

```
module my_poly
implicit none
contains

subroutine decbinn(nn,nb,k)
integer nn, n, nb(0:31), i, k
nb=0            ! Начальная инициализация результата нулями
if (nn.eq.-huge(0)-1) then; nb(31)=1
  else          ! Если число не наименьшее из отрицательных, то
     n=abs(nn)  ! получаем прямой код неотрицательного числа.
     i=0        ! Индекс элемента для хранения самой младшей цифры.
    do while (n/=0)                ! Пока число не оказалось нулём
      nb(i)=mod(n,2); n=n/2;i=i+1  ! заполняем текущий i-ый элемент
    enddo                          ! очередной двоичной цифрой
    k=i-1                        ! Количество двоичных цифр в числе
    if (nn.lt.0) then            ! Если исходное число < 0, то
      nb=1-nb                    !   находим обратный код и
      i=0; nb(i)=nb(i)+1         !   младший элемент дополнительного.
      do while ((nb(i)==2))      ! Пока текущий элемент == 2
        nb(i)=0                  ! обнуляем его,
        i=i+1                    ! находим индекс соседнего старшего,
        nb(i)=nb(i)+1            ! и увеличиваем значение соседа
      enddo                      !   на "один в уме"
    endif
endif
end subroutine decbinn
```

```
function bindecn(nb,k) result (m)
integer nb(0:31), m, k, s, i
if (nb(31)==0) then; s=1      ! Запоминаем в s  знак числа
  else;              s=-1     ! В случае дополнительного кода
    nb(0)=nb(0)-1             ! получаем обратный код и пока
    i=0; do while (nb(i)==-1) ! его i-ый разряд должен "брать" у
           nb(i)=1            ! соседнего старшего 1, помещаем её.
           i=i+1              ! Вычисляем индекс последнего и
           nb(i)=nb(i)-1      ! его значение.
         enddo
    nb=1-nb                   ! Получаем из обратного кода прямой.
endif
m=nb(k)                       ! Подготовка к применению схемы Горнера
do i=k-1,0,-1                 ! и расчёт по ней
  m=m*2+nb(i)                 !                 значения полинома
enddo
m=isign(m,s)                  ! Присваивание знака результату.
end function bindecn

subroutine binoctn(nb,n8)           ! BINOCTN из цифр двоичной записи
integer(4) nb(0:31), n8(0:10), k, m ! целого, хранящихся в элементах
do k=0,9; m=3*k                     ! nb(0:31) (nb(0)- разряд единиц)
  n8(k)=nb(m)+2*(nb(m+1)+2*nb(m+2)) ! получает в n8(0:10) цифры его
enddo                               ! восьмеричной записи, т.е.
n8(10)=nb(30)+2*nb(31)              ! результат перевода из 2 в 8
end subroutine binoctn              !

subroutine binhexn(nb,ch)           ! BINHEXN из цифр двоичной записи
character(1) ch(0:7)                ! nb(0:31) (nb(0)- разряд единиц)
integer(4) nb(0:31), j, k, m        ! получает в ch(0:7) цифры его
do k=0,7; m=k*4                     !  шестнадцатеричной записи, т.е.
  j=nb(m)+2*(nb(m+1)+2*(nb(m+2)+2*nb(m+3)))   ! формальный результат
  if (j<10) then; ch(k)=achar(iachar('0')+j)  ! перевода из 2 в 16.
            else; ch(k)=achar(iachar('A')+j-10)
  endif
enddo
end subroutine binhexn
end module my_poly
```

3.6.3 Исходный текст главной программы test_3_6

```
program test_3_6; use my_poly; implicit none
integer n, nb(0:31), n8(0:10),i, k; character(1) ch(0:7)
read (*,*) n; write(*,*) '1) n=', n
call decbinn(n,nb,k); write(*,*) '   k=',k
write(*,'(" decbinn       :  n=",32i1)') (nb(i),i=31,0,-1)
write(*,'(" format     b32:  n=",b32)') n
write(*,'(" format  b32.32:  n=",b32.32)') n
call binoctn(nb,n8)
write(*,'(a,11i1)')   ' binoctn       : n8=',(n8(i),i=10,0,-1)
write(*,'(a,o11.11)') ' n_8           : n8=',n
call binhexn(nb,ch)
write(*,'(a,8a1)')    ' binhexn       : nh=',(ch(i),i=7,0,-1)
write(*,'(a,z8.8)')   ' c_h           : nh=',n
n=-n;         write(*,*) '2) n=', n
call decbinn(n,nb,k); write(*,*) '   k=',k
write(*,'(" decbinn       :  n=",32i1)') (nb(i),i=31,0,-1)
write(*,'(" format     b32:  n=",b32)') n
write(*,'(" format  b32.32:  n=",b32.32)') n
call binoctn(nb,n8)
write(*,'(a,11i1)')   ' binoctn       : n8=',(n8(i),i=10,0,-1)
write(*,'(a,o11.11)') ' n_8           : n8=',n
call binhexn(nb,ch)
write(*,'(a,8a1)')    ' binhexn       : nh=',(ch(i),i=7,0,-1)
write(*,'(a,z8.8)')   ' c_h           : nh=',n
end
```

Результат тестирования для $n = \pm 5$

```
1) n=          -5
   k=           2
decbinn       :  n=11111111111111111111111111111011
format     b32:  n=11111111111111111111111111111011
format  b32.32:  n=11111111111111111111111111111011
binoctn       : n8=37777777773
n_8           : n8=37777777773
binhexn       : nh=FFFFFFFB
c_h           : nh=FFFFFFFB
2) n=           5
   k=           2
decbinn       :  n=00000000000000000000000000000101
format     b32:  n=                                101
format  b32.32:  n=00000000000000000000000000000101
binoctn       : n8=00000000005
n_8           : n8=00000000005
binhexn       : nh=00000005
c_h           : nh=00000005
```

Результат тестирования для n = ±14

```
1) n=         -14
   k=           3
decbinn       :  n=11111111111111111111111111110010
format     b32:  n=11111111111111111111111111110010
format  b32.32:  n=11111111111111111111111111110010
binoctn       : n8=37777777762
n_8           : n8=37777777762
binhexn       : nh=FFFFFFF2
c_h           : nh=FFFFFFF2
2) n=          14
   k=           3
decbinn       :  n=00000000000000000000000000001110
format     b32:  n=                               1110
format  b32.32:  n=00000000000000000000000000001110
binoctn       : n8=00000000016
n_8           : n8=00000000016
binhexn       : nh=0000000E
c_h           : nh=0000000E
```

Результат тестирования для n = ±0

```
1) n=           0
   k=          -1
decbinn       :  n=00000000000000000000000000000000
format     b32:  n=                                 0
format  b32.32:  n=00000000000000000000000000000000
binoctn       : n8=00000000000
n_8           : n8=00000000000
binhexn       : nh=00000000
c_h           : nh=00000000
2) n=           0
   k=          -1
decbinn       :  n=00000000000000000000000000000000
format     b32:  n=                                 0
format  b32.32:  n=00000000000000000000000000000000
binoctn       : n8=00000000000
n_8           : n8=00000000000
binhexn       : nh=00000000
c_h           : nh=00000000
```

Результат тестирования для n = ±2147483647

```
1) n= -2147483647
   k=          30
decbinn       :  n=10000000000000000000000000000001
format     b32:  n=10000000000000000000000000000001
format  b32.32:  n=10000000000000000000000000000001
binoctn       : n8=20000000001
n_8           : n8=20000000001
binhexn       : nh=80000001
c_h           : nh=80000001
2) n=  2147483647
   k=          30
decbinn       :  n=01111111111111111111111111111111
format     b32:  n= 11111111111111111111111111111111
format  b32.32:  n=01111111111111111111111111111111
binoctn       : n8=17777777777
n_8           : n8=17777777777
binhexn       : nh=7FFFFFFF
c_h           : nh=7FFFFFFF
```

Результат тестирования для n = ±2147483648

```
1) n= -2147483648
   k=           0
decbinn       :  n=10000000000000000000000000000000
format     b32:  n=10000000000000000000000000000000
format  b32.32:  n=10000000000000000000000000000000
binoctn       : n8=20000000000
n_8           : n8=20000000000
binhexn       : nh=80000000
c_h           : nh=80000000
2) n= -2147483648
   k=           0
decbinn       :  n=10000000000000000000000000000000
format     b32:  n=10000000000000000000000000000000
format  b32.32:  n=10000000000000000000000000000000
binoctn       : n8=20000000000
n_8           : n8=20000000000
binhexn       : nh=80000000
c_h           : nh=80000000
```

- Заметим, что в восьмеричном и шестнадцатеричном выводах знак числа трактуется как часть значащей цифры числа. В случае положительных чисел это несущественно.

- Однако, в дополнительном коде восьмеричной записи **20000000001** отрицательного числа **-214748367** имеем в качестве старшей цифры **двойку**, которую, как следует из двоичной записи,

  ```
  10 000 000 000 000 000 000 000 000 000 001
  ```

  следует трактовать просто как знак минус, а для получения восьмеричной записи числового аналога в прямом коде необходимо осуществить соответствующий перевод, именно: получить обратный код путём вычитания единицы из разряда единиц

  ```
  10 000 000 000 000 000 000 000 000 000 000 = -2*8^10+1
  ```

  и посредством инверсии получить нужный прямой код

  ```
  (01 111 111 111 111 111 111 111 111 111 111)_2=-2^{31}+1=
  (-1  7   7   7   7   7   7   7   7   7   7 )_8=-8^{10}*2+1=
                                                  =-2147483647=
  -8^10-7*8^9-7*8^8-7*8^7-7*8^6-7*8^5-7*8^4-7*8^3-7*8^2-7*8-7+1
  ```

3.7 Задача 7

Значение интеграла по промежутку **[a,b]** от подинтегральной функции **f(x)** приближенно можно вычислить через квадратурную сумму формулы **средних прямоугольников**:

$$\tilde{\mathbf{S}} = \mathbf{h} \cdot \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{f}(\mathbf{x_i}) \quad \text{где} \quad \mathbf{h} = \frac{\mathbf{b-a}}{\mathbf{n}}; \qquad \mathbf{x_i} = \mathbf{a} + \mathbf{h} \cdot (\mathbf{i} - \frac{\mathbf{1}}{\mathbf{2}}); \qquad \mathbf{i} = \mathbf{1}, \mathbf{2}, \cdots \mathbf{n}.$$

n – количество промежутков равномерного дробления отрезка **[a,b]**.

Разработать функцию **rectan(y,a,b,n)**, которая по набору из **n** значений подинтегральной функции, хранящихся в первых **n** элементах вектора **y** вычисляет значение квадратурной суммы формулы средних прямоугольников. Описание функции **rectan** поместить в модуль **quarda**.

Главная программа должна вводить значения **a**, **b** и **n** из файла, заполнять первые **n** элементов вектора **y**, вызывать **rectan** и печатать результат расчета.

3.7.1 Исходный текст модуля my_prec

```
module my_prec                !                          Файл my_prec.f90
implicit none
integer, parameter :: mp=10   ! <--= тут задаём mp=4, 8, 10, 16
contains
function frmp() result(f)       ! Функция frmp() по значению mp возвращает
character(21) f                 ! строковое значение, хранящее дескриптор
character(21),parameter :: frm(4)=              ! вывода одного значения
>(/'(e14.6 )','(e24.14)','(e27.18)','(e42.35)'/) ! типа real(mp)
integer m
m=int(mp/4.0+0.6)
f=frm(m)
end function frmp
end module my_prec
```

- Исходный текст функции **frmp** включён в модуль **my_prec**, так как функция выбирает формат вывода значения типа **real(mp)**.
- **frmp** используется в главной программе, которая через операцию **конкатенация** осуществляет сцепку трёх строк **'(a,'**, **trim(fr)** и **')'**, что позволяет использовать строковую переменную, объединившую их, в качестве форматной строки для вывода любого значения типа **character(*)** и одного значения типа **real(mp)**.

3.7.2 Исходный текст модуль quadra

```
module quadra
use my_prec
implicit none
contains
function rectan(y,a,b,n) result(s)
integer n, i
real(mp) y(n), a, b, s, h
h=(b-a)/n
s=0                          !\  В современном ФОРТРАНе вместо этих трёх
do i=1, n; s=s+y(i); enddo ! > строк можно:
s=s*h                        !/                  s=sum(y)*h
end function rectan
end module quadra
```

3.7.3 Исходный текст главной программы test_3_7

```
program test_3_7; use my_prec; use quadra;  implicit none
real(mp) a, b, h, x, r
real(mp) y1(100000), y2(100000)
integer i, n
character(21) fr,frm
read(*,'(e10.3)') a, b
read(*,'(i10)') n
fr=frmp()
frm='(a,'//trim(fr)//')'
write(*, '(a,i5)') ' # mp =', mp
write(*,trim(frm)) ' #   a=', a
write(*,trim(frm)) ' #   b=', b
write(*,'(a,i10)') ' #   n=', n
h=(b-a)/n
do i=1,n
  x=a+(i-0.5_mp)*h
  y1(i)=f(x)
  y2(i)=g(x)
enddo
r=rectan(y1,a,b,n)
write(*,trim(frm)) ' # f(x)= x ; r=', r
r=rectan(y2,a,b,n);
write(*,trim(frm)) ' # f(x)=x^2; r=', r
stop 0
contains
function f(x) result(w); real(mp) x, w; w=x;   end function f
function g(x) result(w); real(mp) x, w; w=x*x; end function g
end program test_3_7
```

3.7.4 Результаты тестирования rectan

mp=4, n=1

```
# mp =    4
#   a=  0.000000E+00
#   b=  0.100000E+01
#   n=    1
# f(x)= x ; r=  0.500000E+00
# f(x)=x^2; r=  0.250000E+00
```

- Даже по одной точке формула средних прямоугольников для любой линейной функции в идеале должна дать точный результат, поскольку остаточное слагаемое квадратурной формулы пропорционально второй производной подынтегральной функции (т.е. в случае линейной функции равно нулю).
- В случае квадратичной функции остаток квадратурной формулы уже не нуль. Поэтому одной точки дробления недостаточно для получения удовлетворительного результата (0.25 вместо 0.3(3)).

mp=4, n=100

```
# mp =    4
#   a=  0.000000E+00
#   b=  0.100000E+01
#   n=  100
# f(x)= x ; r=  0.500000E+00
# f(x)=x^2; r=  0.333325E+00
```

- В случае квадратичной функции увеличение точек дробления позволяет повысить точность расчёта. Например, при **n=100** значение интеграла по промежутку **[0,1]** от функции $\mathbf{x^2}$ равно **0.333325**, что возможно и приемлемо при невысокоточных расчётах.

mp=10, n=1000

```
# mp =   10
#   a=   0.00000000000000000E+00
#   b=   0.10000000000000000E+01
#   n=      1000
# f(x)= x ; r=   0.50000000000000000E+00
# f(x)=x^2; r=   0.33333250000000000E+00
```

3.7.5 Важное замечание.

В главной программе расчёт аргумента **x** ведётся по формуле

$$x = a + (i - 0.5) * h,$$

а не посредством **x=x+h**, т.е. не прибавлением к текущему значению **x** шага **h** при начальном **x=a+h/2**. И это не случайно. Безусловно, формула $\mathbf{x} = \mathbf{x} + \mathbf{h}$, безусловно, привлекает своей простотой, но, тем не менее, она гораздо опаснее используемой.

Дело в том, что при $\mathbf{x} = \mathbf{x} + \mathbf{h}$ после каждой операции сложения в переменной **x** накапливается погрешность округления от всех предыдущих сложений, в то время как в первом случае только от одной. Подчеркнём, что речь идёт не о накоплении погрешности шага **h**, а о накоплении погрешности операции округления при сложении.

Рассмотрим, пример. Разобьём отрезок **[0,1]** на **n** равных частей, и посмотрим на результаты работы программы

```
program error; implicit none; real a, x, h, s; integer n, i
read(*,*) n
a=0; x=0
h=1.0/n
do i=1, n
  x=x+h
  s=a+i*h
enddo
write(*,*) n, x, s, h, epsilon(1.0)/2
end
```

которая складывает все эти части двумя упомянутыми способами. Инициировать запуск программы после её компиляции можно посредством **./a.out ≫ res**, вводя разные **n**, и накапливая результат в файле **res**.

```
      n           x             s             h
          1   1.0000000       1.0000000       1.0000000
         10   1.0000001       1.0000000      0.10000000
        100  0.99999934       1.0000000      9.99999978E-03
       1000  0.99999070       1.0000000      1.00000005E-03
      10000   1.0000535       1.0000000      9.99999975E-05
     100000   1.0009902       1.0000000      9.99999975E-06
    1000000   1.0090389       1.0000000      9.99999997E-07
   10000000   1.0647675       1.0000000      1.00000001E-07
   11000000   1.1454368       1.0000000      9.09090900E-08
```

Вряд ли нас устроит значение **x**, полученное при $\mathbf{n} \gg \mathbf{1}$, если ведутся высокоточные расчёты. Однако, при дальнейшем увеличении **n** значение **x** начинает убывать и при **n=16777216** оказывается в точности равным единице, причём этот верный результат держится вплоть до **n=30000000**:

n	x	s	h	epsilon/2
12000000	0.87475812	1.0000000	8.33333331E-08	
13000000	0.91010654	1.0000000	7.69230795E-08	
14000000	0.88197190	1.0000000	7.14285733E-08	
15000000	0.90791243	1.0000000	6.66666651E-08	
16000000	0.95713460	1.0000000	6.24999998E-08	
16700000	0.99544382	1.0000000	5.98802430E-08	
16775000	0.99986798	1.0000000	5.96125176E-08	
16777000	0.99998719	1.0000000	5.96054122E-08	5.96046448E-08
16777216	1.0000000	1.0000000	5.96046448E-08	5.96046448E-08
16777500	1.0000000	1.0000000	5.96036358E-08	5.96046448E-08
30000000	1.0000000	1.0000000	3.33333325E-08	5.96046448E-08

Согласитесь, что случайно задав **n=20000000** или **n=30000000**, и убедившись в правильности ответа, запросто можно подумать, что, как говорил дока Лейстред: *« Всякие теории нам ни к чему, и можно опираться на очевидные факты»*.

В последней таблице наряду с **n**, **x** и **s** выведено и значение шага **h**, а также и значение наибольшего для типа **real(4)** денормализованного числа. Они при правильном анализе ситуации помогут уяснить причину наблюдаемого эффекта.

Истинная причина получения правильного результата в том, что значение шага приближается к числу, которое до последнего бита может быть точно записано в поле двоичной мантиссы, т.е. с погрешностью округления равной нулю. В этом случае и погрешность округления суммы равна нулю, так как значения операндов абсолютно точны.

Однако, не следует думать, что дальнейшее увеличение **n** не обнаружит ничего нового:

n	x	s	h	epsilon/2
40000000	0.50000000	1.0000000	2.50000003E-08	5.96046448E-08
50000000	0.50000000	1.0000000	1.99999999E-08	5.96046448E-08
1000000000	3.12500000E-02	1.0000000	9.99999972E-10	5.96046448E-08
2000000000	1.56250000E-02	1.0000000	4.99999986E-10	5.96046448E-08
2147000000	1.56250000E-02	1.0000000	4.65766203E-10	5.96046448E-08
2147483647	7.81250000E-03	1.0000000	4.65661287E-10	5.96046448E-08

Причины происходящего в том, что

1. значения **h** настолько малы, что даже часть накопленной суммы из-за конечности разрядной сетки не может увеличиться за счёт прибавления **h**;

2. значения **h** настолько малы, что денормализованы, т.е. записываются в памяти меньшим числом бит нежели предоставляет тип **real(4)**.

3. далеко не каждое **h** имеет нулевую погрешность округления

Помочь уяснить сказанное может пропуск программы:

```
program error2; implicit none; real a, x, h, s; integer n, i, k
n=1           ! Почему результаты, полученные программой error имеют
do k=1,30     ! ненулевую погрешность округления, а результаты
  a=0; x=0    ! программы error2 --- НУЛЕВУЮ (вплоть до n=16777216)?
  h=1.0/n     !
  do i=1, n   ! Почему, работая в режиме нулевой погрешности,
    x=x+h     ! программа error2, пытаясь сложить n=33554432 чисел
    s=a+i*h   ! 1.0/33554432, получает результат меньший, чем при
  enddo       ! сложении n=16777216 чисел 1.0/16777216?
  write(*,*) n, x, s, h, epsilon(1.0)/2
  n=n*2
enddo
end
```

n	x	s	h	epsilon/2
512	1.0000000	1.0000000	1.95312500E-03	5.96046448E-08
1024	1.0000000	1.0000000	9.76562500E-04	5.96046448E-08
2048	1.0000000	1.0000000	4.88281250E-04	5.96046448E-08
4096	1.0000000	1.0000000	2.44140625E-04	5.96046448E-08
8192	1.0000000	1.0000000	1.22070313E-04	5.96046448E-08
16384	1.0000000	1.0000000	6.10351563E-05	5.96046448E-08
32768	1.0000000	1.0000000	3.05175781E-05	5.96046448E-08
65536	1.0000000	1.0000000	1.52587891E-05	5.96046448E-08
131072	1.0000000	1.0000000	7.62939453E-06	5.96046448E-08
262144	1.0000000	1.0000000	3.81469727E-06	5.96046448E-08
524288	1.0000000	1.0000000	1.90734863E-06	5.96046448E-08
1048576	1.0000000	1.0000000	9.53674316E-07	5.96046448E-08
2097152	1.0000000	1.0000000	4.76837158E-07	5.96046448E-08
4194304	1.0000000	1.0000000	2.38418579E-07	5.96046448E-08
8388608	1.0000000	1.0000000	1.19209290E-07	5.96046448E-08
16777216	1.0000000	1.0000000	5.96046448E-08	5.96046448E-08
33554432	0.50000000	1.0000000	2.98023224E-08	5.96046448E-08
67108864	0.25000000	1.0000000	1.49011612E-08	5.96046448E-08
134217728	0.12500000	1.0000000	7.45058060E-09	5.96046448E-08
268435456	6.25000000E-02	1.0000000	3.72529030E-09	5.96046448E-08
536870912	3.12500000E-02	1.0000000	1.86264515E-09	5.96046448E-08

4 Размещаемые массивы (второй семестр).

Решения задач оформить с использованием **динамического массива**.

- **Решения представить на ФОРТРАНе-95 и С**.
- ФОРТРАН-решения задач с **1** по **7** должны подключать модуль **sort**, содержащий описания всех используемых в них процедур.
- ФОРТРАН-решения задач с **8** по **9** должны подключать модуль **quadra** (из предыдущего домашнего задания с описанием функции **rectan**), дополненный функциями **trap** (задача 8) и **sim** (задача 9).
- Инициировать запуск каждой программы должен **make-файл**.

1. Разработать функцию, определяющую число нечётных элементов вектора целого типа, и продемонстрировать её работоспособность на тестовых примерах. Описание функции поместить в модуль **sort**.
2. Разработать функцию и подпрограмму, которые объединяют два вектора в один с чередованием элементов исходных векторов целого типа, и продемонстрировать её работоспособность на тестовых примерах. Описание функции поместить в модуль **sort**
3. Разработать нерекурсивную и рекурсивную функции инвертирования вектора (перестановки элементов в обратном порядке). Описание функции поместить в модуль **sort**.
4. Разработать функцию циклического сдвига элементов вектора **y(n)**: при сдвиге вправо содержимое **k**-го элемента переносится в **k+1**-й элемент (**k=1, ... , n-1**), а содержимое **n**-го элемента — в **первый**; при сдвиге влево содержимое **k**-го переносится в **k-1**-й (**k=2, ... , n**), а содержимое элемента **y(1)** — в **y(n)**.
5. Разработать функцию проверки вектора на упорядоченность.
6. Разработать нерекурсивную и рекурсивную функции поиска образца в упорядоченном по величине элементов векторе.
7. Разработать функцию и процедуру объединения двух упорядоченных массивов в один с сохранением упорядоченности.

8. Значение интеграла по промежутку **[a,b]** от подинтегральной функции **f(x)** приближенно вычисляется через квадратурную сумму формулы трапеций (подробнее см. приложение **IV**):

$$\tilde{\mathbf{S}} = \mathbf{h} \cdot \Big[\frac{\mathbf{f(a)} + \mathbf{f(b)}}{\mathbf{2}} + \sum_{\mathbf{i=2}}^{\mathbf{n}} \mathbf{f(x_i)}\Big]$$

где

$$\mathbf{h} = \frac{\mathbf{b-a}}{\mathbf{n}}; \qquad \mathbf{x_i = a + (i-1)h} \qquad \mathbf{i = 2, \cdots n}; \quad \mathbf{x_1 \equiv a} \ , \ \mathbf{x_{n+1} \equiv b}.$$

n – количество промежутков равномерного дробления отрезка **[a,b]**.

Разработать функцию **trap(y,a,b,n)**, которая по **n+1** значениям подинтегральной функции, хранящихся в первых **n+1** элементах вектора **y** вычисляет значение квадратурной суммы формулы трапеций. Описание **trap** поместить в модуль **quarda**, в котором хранится и процедура **rectan** из предыдущего домашнего задания.

Главная программа должна подсоединять соответствующий модуль, вводить исходные данные из файла, вызывать нужную процедуру и выводить результат вместе с таблицей подинтегральной функции так, чтобы, получить её график, активируя цель **make plot**.

Для численного расчёта интегралов наряду с формулами прямоугольников и трапеций широко используется **квадратурная сумма составной формулы Симпсона**:

$$\tilde{\mathbf{S}} = \frac{\mathbf{h}}{\mathbf{3}} \cdot [\mathbf{f(a)} + \mathbf{f(b)} + \mathbf{4} \cdot \mathbf{S_1} + \mathbf{2} \cdot \mathbf{S_2}]$$

$$\mathbf{S_1} = \mathbf{f_2} + \mathbf{f_4} + \cdots + \mathbf{f_{n=2m}} \ , \ \mathbf{S_2} = \mathbf{f_3} + \mathbf{f_5} + \cdots + \mathbf{f_{n-1}}.$$

$$\mathbf{f_i = f(x_i)} \ \ \mathbf{x_i = a + h \cdot (i-1)}, \ \ \mathbf{i = 1, 2, \cdots, (n = 2m), n+1}, \ \ \mathbf{h} = \frac{\mathbf{b-a}}{\mathbf{n}}$$

причём **n** — число промежутков дробления обязательно чётно.

Разработать функцию **sim(y,a,b,n)**, которая по набору из **n+1** значений подинтегральной функции, хранящихся в первых **n+1** элементах вектора **y** вычисляет значение квадратурной суммы формулы трапеций. Описание функции **sim** поместить в модуль **quarda** (тот же самый, в котором уже размещены **rectan** и **trap**).

4.1 Задача 1

Разработать функцию, определяющую количество нечётных элементов вектора целого типа, и продемонстрировать её работоспособность на тестовых примерах (все элементы чётные, один элемент нечётный, три нечётных элемента, все элементы нечётные). Описание функции поместить в модуль **sort**.

```
module sort; implicit none
contains
function odd1(vec,n) result(k)    ! Так оформили бы заголовок
integer  n, i, k                  ! функции на ФОРТРАНе-77
integer :: vec(n)                 !
k=0                               ! На современном так тоже можно,
do i=1,n                          ! но можно и сократить число
  if  (mod(vec(i),2)==1) k=k+1    ! формальных аргументов только
enddo                             ! до имени массива (см. ниже odd2)
end function odd1

function odd2(vec) result(k)      ! Лишний аргумент в списке
integer  n, i, k                  ! формальных параметров ---
integer :: vec(:)                 ! дополнительная возможность
n=size(vec)                       ! для опечатки.
k=0                               ! Современный ФОРТРАН позволяет
do i=1,n                          ! узнавать длину массива (см. size)
  if  (mod(vec(i),2)==1) k=k+1    ! если формальный аргумент
enddo                             ! перенимает размер фактического.
end function odd2
end module sort

program test1; use sort; implicit none
integer ier
integer, allocatable :: v(:)
character(60) s
integer n, i, k
read (*,*) n, s
write(*,*) '  n=', n, trim(s)
allocate(v(n),stat=ier)
if (ier/=0) then; write(*,*) 'ier=',ier,' do not allocate array !!!'
                  stop 1
endif
read (*,*) (v(i),i=1,n)
write(*,'(2(2x,"i",2x,"v(i)",5x))')
write(*,'(2(i3,i5,6x))') (i, v(i), n/2+i, v(n/2+i), i=1,n/2+mod(n,2))
k=odd1(v,n); write(*,*) ' odd1: "Нечётных элементов  k=',k,'".'
k=odd2( v ); write(*,*) ' odd2: "Нечётных элементов  k=',k,'".'
end program test1
```

Тестовые результаты:

```
 n=          10 (число_элементов)
i  v(i)       i  v(i)
1    2        6    4
2    4        7    6
3    6        8    8
4    8        9    6
5    2       10    6
odd1: "Нечётных элементов  k=           0 ".
odd2: "Нечётных элементов  k=           0 ".

 n=          10 (число_элементов)
i  v(i)       i  v(i)
1    2        6    3
2    4        7    6
3    6        8    8
4    8        9    6
5    2       10    6
odd1: "Нечётных элементов  k=           1 ".
odd2: "Нечётных элементов  k=           1 ".

 n=          10 (число_элементов)
i  v(i)       i  v(i)
1    2        6    3
2    1        7    6
3    6        8    8
4    8        9    9
5    2       10    6
odd1: "Нечётных элементов  k=           3 ".
odd2: "Нечётных элементов  k=           3 ".

 n=          11 (число_элементов)
i  v(i)       i  v(i)
1    1        6   11
2    3        7   13
3    5        8   15
4    7        9   17
5    9       10   19
6   11       11   21
odd1: "Нечётных элементов  k=          11 ".
odd2: "Нечётных элементов  k=          11 ".
```

4.2 Задача 2

Разработать функцию и подпрограмму, которые объединяют два вектора в один с чередованием элементов исходных векторов целого типа, и продемонстрировать её работоспособность на тестовых примерах. Описание функции поместить в модуль **sort**

```
module sort; implicit none; contains
!...   ...   ...   ...   ...   ...
subroutine alter1(a,b,c,na,nb,nc)   ! Описание заголовка и тела
integer na, nb, nc, i               ! в стиле ФОРТРАНа-77
integer a(na), b(nb), c(nc)
if (na<=nb) then; do i=1, na;   c(2*i-1)=a(i); c(2*i)  =b(i); enddo
                  do i=na+1,nb; c( na+i)=b(i); enddo
            else; do i=1, nb;   c(2*i-1)=a(i); c(2*i)  =b(i); enddo
                  do i=nb+1,na; c( nb+i)=a(i); enddo
endif
end  subroutine alter1
subroutine alter2(a,b,c,na,nb,nc)   ! Заголовок в стиле ФОРТРАНа-77,
integer na, nb, nc, i               ! но синтаксис ФОРТРАНа-95:
integer a(na), b(nb), c(nc)         ! индексный триплет вместо цикла.
if (na<=nb) then; c(1:2*na-1 :2)=a(1:na); c(2: 2*na  :2)=b(1:na)
                                          c(2*na+1:nc:1)=b(na+1:nb)
            else; c(1:2*nb-1:2)=a(1:nb);  c(2:2*nb:2)=b(1:nb)
                                          c( 2*nb+1:nc)=a(nb+1:na)
endif
end  subroutine alter2
subroutine alter3(a,b,c)             ! Заголовок в стиле ФОРТРАНа-95
integer a(:), b(:), c(:), na, nb, nc ! Длину массива, перенимающего
na=size(a); nb=size(b); nc=size(c)   ! форму находит функция size.
if (na<=nb) then; c(1:2*na-1 :2)=a(1:na); c(2: 2*na  :2)=b(1:na)
                                          c(2*na+1:nc:1)=b(na+1:nb)
            else; c(1:2*nb-1:2)=a(1:nb);  c(2: 2*nb  :2)=b(1:nb)
                                          c( 2*nb+1:nc)=a(nb+1:na)
endif
end  subroutine alter3
function alter4(a,b) result(c)         ! Функция ФОРТРАНа-95
integer a(:), b(:), c(size(a)+size(b)) ! может возвращать через
integer na, nb, nc                     ! своё  имя   МАССИВ!
na=size(a); nb=size(b); nc=size(c)
if (na<=nb) then; c(1:2*na-1 :2)=a(1:na); c(2: 2*na  :2)=b(1:na)
                                          c(2*na+1:nc:1)=b(na+1:nb)
            else; c(1:2*nb-1:2)=a(1:nb);  c(2: 2*nb  :2)=b(1:nb)
                                          c( 2*nb+1:nc)=a(nb+1:na)
endif
end function alter4
end module sort
```

1. Строка с многоточием в файле с модулем означает процедуры, не востребованные в задаче **№2**.

2. Модуль содержит процедуры **alter1**, **alter2**, **alter3** и **alter4**, каждая из которых решает поставленную задачу, отличаясь по записи лишь синтаксисом возможностей современного ФОРТРАНа. **alter1**, **alter2** и **alter3** оформлены подпрограммами (старый ФОРТРАН допускал возврат массива-результата лишь через формальный аргумент). **alter4** — функция (современный ФОРТРАН разрешает возвращать через имя функции, в частности, и массив).

3. **alter1(a, b, c, na, nb, nc)** имеет шесть формальных аргументов: три массива **a, b, c** и соответствующие им размеры **na, nb, nc**. Последние должны быть заданы перед вызовом процедур. **alter1** годится и для старого и для современного ФОРТРАНа.

4. **alter2(a, b, c, na, nb, nc)** имеет те же шесть формальных аргументов, что и **alter1**. Однако решение задачи записано на языке индексных дескрипторов (их нет в старом ФОРТРАНе).

5. **alter3(a, b, c)** имеет только три формальных аргумента — массивы, объявленные массивами, перенимающими форму (*чью?* **a(:), b(:), c(:)**). Узнать размеры массивов внутри процедур можно посредством встроенной функции **size** (нет в старом ФОРТРАНе).

6. **alter4(a, b)** — описана как функция; имеет только два формальных аргумента — массивы, перенимающие форму (**a(:), b(:)**), а результирующий массив **c** возвращает через имя **alter4**, при этом описание размеров массива-результата даётся через посредство функции **size**.

7. Достоинство помещения процедур в модуль заключается в том, что

   - вызвать их можно лишь из тех программных единиц, которые этот модуль подключают;
   - при оформлении процедур просто внешними процедурами программная единица, вызывающая их, должна содержать явное описание их интерфейса (в случаях **alter1** и **alter2** это только полезно; но в случаях **alter3** и **alter4** необходимо). Интерфейс модульных процедур передаётся автоматически.

8. Обнуление вектора **c** перед обращением к **alter1**, **alter2** и **alter3** гарантирует невозможность принять за текущий результат (содержимое вектора **c**) значение предыдущего.

9. Повторный вызов каждой из процедур демонстрирует правильность её работы, когда длина первого массива меньше длины второго.

4.2.1 Главная программа.

```
program test2; use sort; implicit none; integer ier
integer, allocatable :: a(:), b(:), c(:)
character(60) s
integer na, nb, nc, i
read (*,*) na, s; write(*,*) '  na=', na, trim(s)
allocate(a(na),stat=ier); if (ier/=0) then; write(*,*) 'a: not allocate.'
                                            stop 1
                          endif
read(*,*) a; write(*,'(a,100i3)') "a:",a
read (*,*) nb, s; write(*,*) '  nb=', nb, trim(s)
allocate(b(nb),stat=ier); if (ier/=0) then; write(*,*) 'b: not allocate.'
                                            stop 2
                          endif
read (*,*) b; write(*,'(a,100i3)') "b:",b
nc=na+nb;     write(*,*) '  nc=', nc
allocate(c(nc),stat=ier); if (ier/=0) then; write(*,*) 'c: not allocate.'
                                            stop 3
                          endif
c=0; call alter1(a,b,c,na,nb,nc)
write(*,'(a,100i3)') "alter1(a,b,c,na,nb,nc) c:",c
call alter1(b,a,c,nb,na,nc)
write(*,'(a,100i3)') "alter1(b,a,c,nb,na,nc) c:",c
write(*,'(55("-"))')
c=0; call alter2(a,b,c,na,nb,nc)
write(*,'(a,100i3)') "alter2(a,b,c,na,nb,nc) c:",c
call alter2(b,a,c,nb,na,nc)
write(*,'(a,100i3)') "alter2(b,a,c,nb,na,nc) c:",c
write(*,'(55("-"))')
c=0; call alter3(a,b,c)
write(*,'(a,100i3)') "alter3(a,b,c)          c:",c
call alter3(b,a,c)
write(*,'(a,100i3)') "alter3(b,a,c)          c:",c
write(*,'(55("-"))')
write(*,'(a,100i3)') "alter4(a,b)             :", alter4(a,b)
write(*,'(a,100i3)') "alter4(b,a)             :", alter4(b,a)
deallocate(a,b,c)
stop 0
end program test2
```

4.2.2 Тестовые результаты:

```
    5  (число_элементов_a)                              !  Вводимые данные:
 1  2  3  4  5
    5  (число_элементов_b)
 10 20 30 40 50

   na=           5 (число_элементов_a)                  !  Результат:
a:  1  2  3  4  5
   nb=           5 (число_элементов_b)
b: 10 20 30 40 50
   nc=          10
alter1(a,b,c,na,nb,nc) c:  1 10  2 20  3 30  4 40  5 50
alter1(b,a,c,nb,na,nc) c: 10  1 20  2 30  3 40  4 50  5
-------------------------------------------------------
alter2(a,b,c,na,nb,nc) c:  1 10  2 20  3 30  4 40  5 50
alter2(b,a,c,nb,na,nc) c: 10  1 20  2 30  3 40  4 50  5
-------------------------------------------------------
alter3(a,b,c)          c:  1 10  2 20  3 30  4 40  5 50
alter3(b,a,c)          c: 10  1 20  2 30  3 40  4 50  5
-------------------------------------------------------
alter4(a,b)             :  1 10  2 20  3 30  4 40  5 50
alter4(b,a)             : 10  1 20  2 30  3 40  4 50  5

    5  (число_элементов_a)                              ! Вводимые данные
 1  2  3  4  5
    3  (число_элементов_b)
 6 7 8

   na=           5 (число_элементов_a)                  ! Результат:
a:  1  2  3  4  5
   nb=           3 (число_элементов_b)
b:  6  7  8
   nc=           8
alter1(a,b,c,na,nb,nc) c:  1  6  2  7  3  8  4  5
alter1(b,a,c,nb,na,nc) c:  6  1  7  2  8  3  4  5
-------------------------------------------------------
alter2(a,b,c,na,nb,nc) c:  1  6  2  7  3  8  4  5
alter2(b,a,c,nb,na,nc) c:  6  1  7  2  8  3  4  5
-------------------------------------------------------
alter3(a,b,c)          c:  1  6  2  7  3  8  4  5
alter3(b,a,c)          c:  6  1  7  2  8  3  4  5
-------------------------------------------------------
alter4(a,b)             :  1  6  2  7  3  8  4  5
alter4(b,a)             :  6  1  7  2  8  3  4  5
```

4.3 Задача 3

Разработать нерекурсивную и рекурсивную функции инвертирования вектора (перестановки элементов в обратном порядке). Описание функции поместить в модуль **sort**.

```
module sort
implicit none
contains
! ...   ...   ...   ...   ...   ...
subroutine invert_n1(a,n); integer n, a(n), i, j ! инвертирование
do i=1,n/2                                       ! вектора:
 j=a(i); a(i)=a(n+1-i); a(n+1-i)=j               !
enddo                                            !    invert_n1,
end subroutine invert_n1

subroutine invert_n2(a); integer n, a(:), i, j   !    invert_n2,
n=size(a)
do i=1, n/2
 j=a(i); a(i)=a(n+1-i); a(n+1-i)=j
enddo
end subroutine invert_n2

subroutine invert_n3(a); integer n, a(:)         !     invert_n3
n=size(a)
a(1:n)=a(n:1:-1)
end subroutine invert_n3

function invert_n4(a) result(b);                 !     invert_n4
integer  a(:), b(size(a))
b(1:size(a))=a(size(a):1:-1)
end function invert_n4

recursive subroutine invert_r1(a,n,n1,n2)        !  Рекурсивное
integer n, a(n), n2, n1, j                       ! инвертирование
if (n1/=n2) then;  j=a(n1); a(n1)=a(n2); a(n2)=j !    вектора
                  call invert_r1(a,n,n1+1,n2-1)  !
endif                                            !     invert_r1
end subroutine invert_r1

recursive subroutine invert_r2(a,n1,n2)          !     invert_r2
integer a(:), n2, n1, j
if (n1/=n2) then;  j=a(n1); a(n1)=a(n2); a(n2)=j
                  call invert_r2(a,n1+1,n2-1)
endif
end subroutine invert_r2
end module sort
```

- Строка с многоточием в файле с модулем означает процедуры, которые не востребованы в задаче №3.
- Процедуры **invert_n1**, **invert_n2**, **invert_n3**, **invert_n4** — нерекурсивные; а **invert_r1** и **invert_r2** — рекурсивные.
- Обычно подразумевается, что инвертирование выполняется *in situ*, т.е. не используя вспомогательный рабочий вектор (можно использовать одну простую рабочую переменную). Процедуры **invert_n1** и **invert_n2** удовлетворяют последнему требованию. **invert_n3** вообще не использует вспомогательной переменной. **invert_n4** возвращает через своё имя массив-результат, что позволяет сохранить массив, поданный на инвертирование.
- У **invert_n1** два формальных аргумента: массив **a** и его размер **n**.
- У **invert_n2**, **invert_n3** и **invert_n4** — один формальный аргумент, перенимающий форму фактического аргумента (*Почему?*).
- Второй аргумент у **invert_n1** (длина массива) может быть причиной ошибки (при опечатке), которую в принципе нельзя совершить, обращаясь к **invert_n2**, **invert_n3** или **invert_n4**.

```
program test3; use sort; implicit none
integer ier, k
integer, allocatable :: a(:)
character(60) s, sf; character(2) sn
integer n, n1, n2, i
read (*,*) n, s; write(*,*) '  n=', n, trim(s)
allocate(a(n),stat=ier);
if (ier/=0) then; write(*,*) 'a: not allocate.'; stop 1; endif
a=(/(i,i=1,n)/)
write(sn,'(i2)') n; sf='(a,'//sn//'i3)'; write(*,trim(sf)) "a:",a(1:n)
call invert_n1(a,n); write(*,trim(sf)) "invert_n1(a,n):  a=", a(1:n)
call invert_n2( a ); write(*,trim(sf)) "invert_n2( a ):  a=", a(1:n)
call invert_n3( a ); write(*,trim(sf)) "invert_n3( a ):  a=", a(1:n)
write(*,trim(sf)) "invert_n4( a ):   =", invert_n4(a(1:n))
write(*,trim(sf)) "                 a=",             a(1:n)
n1=1; n2=n; call invert_r1(a,n,n1,n2)
            write(*,trim(sf)) "invert_r1(a,n,n1,n2):  a=", a(1:n)
n1=1; n2=n; call invert_r2(a,n1,n2);
             write(*,trim(sf)) "invert_r2(a,n1,n2):    a=", a(1:n)
deallocate(a)
end program test3
```

1. Программа **test3** вводит **n** — размер массива. Содержимое массива формируется программно посредством конструктора массива и циклоподобного списка.

2. В соответствие с темой задания массив **a** объявлен **размещаемым**, на что указывает атрибут **allocatable**. Само размещение выполняется функцией **allocate** после того как главная программа ввела нужный размер массива.

3. Ключевой аргумент **stat** (необязательный, но полезный) позволяет выяснить успешность размещения: если соответствующий фактический аргумент (**ier**) равен нулю, то размещение удалось.

4. Символьная переменная **s** использована для хранения текста, поясняющего смысловую нагрузку вводимой переменной **n**. Этот текст удобно видеть не только при обзоре файла ввода **input**:

   ```
   11   (число_элементов_a)              !  Содержимое файла input
   ```

   но (для большей наглядности) и в файле результата **result**. Встроенная функция **trim** *отрубает* хвостовые пробелы в значении **s**.

5. Символьные переменные **sn** и **sf** использованы для демонстрации того, как можно в операторе вывода организовать динамический повторитель оператора **format**.

   В старых версиях ФОРТРАНа повторитель мог быть только целочисленной положительной константой. Например, пять целых двузначных чисел можно было вывести в строку, используя оператор

   ```
   write(*,'(5i3)') (a(i),i=1,5)
   ```

   Но в нашей задаче программа *узнаёт* количество элементов вектора лишь при своей работе. Старый ФОРТРАН не позволял использовать имя переменной вместо **5**. Некоторые ФОРТРАН-компиляторы предоставляют удобную синтаксическую конструкцию: имя переменной, заключённое в угловые скобки:

   ```
   n=5;   write(*,'(<n>i3)') (a(i),i=1,n)
   ```

Однако, подобная возможность не реализована в **gfortran**, поскольку считается синтаксически излишней, ввиду того, что добиться желаемого результата (смоделировать ситуацию динамического повторителя) можно обычными средствами современного ФОРТРАНа, используя понятия строковой переменной и внутреннего файла.

6. Современный ФОРТРАН в качестве устройства ввода/вывода может использовать не только его программный номер, но и строковую переменную (например, **sn**), которая и называется внутренним файлом. Так что оператор вывода **write** в контексте с операторами:

```
character(2) sn; integer n; n=5;  write(sn,'(i2)') n
```

запишет в строковую переменную **sn** значение целочисленной переменной **n** по формату **i2**, т.е. отведя под запись десятичной константы две позиции. Далее переменную **sn** через посредство операции конкатенации можно использовать в качестве вспомогательного *кирпичика* при формировке строки **sf** с нужными дескрипторами:

```
character(2) sn
character(60) sf
n=5;  write(sn,'(i2)') n;  sf='(a,'//sn//'i3)'
```

так что оператор

```
write(*,trim(sf))  'a: ', a(1:n)
```

выведет в строку и текст **a:** и **n** значений элементов массива **a**.

Результат работы программы test_3

```
   n=          11 (число_элементов_a)
a:  1  2  3  4  5  6  7  8  9 10 11
invert_n1(a,n):  a= 11 10  9  8  7  6  5  4  3  2  1
invert_n2( a ):  a=  1  2  3  4  5  6  7  8  9 10 11
invert_n3( a ):  a= 11 10  9  8  7  6  5  4  3  2  1
invert_n4( a ):   =  1  2  3  4  5  6  7  8  9 10 11
                  a= 11 10  9  8  7  6  5  4  3  2  1
invert_r1(a,n,n1,n2):  a=  1  2  3  4  5  6  7  8  9 10 11
invert_r2(a,n1,n2):    a= 11 10  9  8  7  6  5  4  3  2  1
```

Оценка временных затрат Интересно сравнить затраты времени на инвертирование у рассмотренных процедур. Программа **test3a** по сути — копия программы **test3**, из которой исключены операторы вывода результата инвертирования, добавлен **1001**-кратный вызов каждой из процедур, в переменных **t1** и **t2** фиксируются соответствующие моменты процессорного времени, а оценка временных затрат выполняется в единицах **1001**-кратного вызова процедуры **invert_n1**. Пример вывода программы **test3a**:

```
 kmax=        1001
    n=        2001 (число_элементов_a)
   Процедура      Время
invert_n1(a,n):    1.00
  invert_n2(a):    0.82
  invert_n3(a):    1.00
  invert_n4(a):    1.09
invert_r1,n1,n2(a,n):    1.91
invert_r2(a,n1,n2)  :    2.36
```

```
program test3a; use sort; implicit none; integer, parameter :: kmax=1001
integer ier, k
integer, allocatable :: a(:)
character(60) s
integer n, n1, n2, i;   real(4) t1, t2, t12
write(*,*) 'kmax=',kmax
read (*,*) n, s; write(*,*) '   n=', n, trim(s)
allocate(a(n),stat=ier)
if (ier/=0) then; write(*,*) 'a: not allocate.'; stop 1; endif
a=(/(i,i=1,n)/)
write(*,*) '  Процедура      Время'
call cpu_time(t1); do k=1,kmax; call invert_n1(a,n); enddo; call cpu_time(t2)
t12=t2-t1;                   write(*,'("invert_n1(a,n): ",f7.2)') 1.0
call cpu_time(t1); do k=1,kmax; call invert_n2(a); enddo; call cpu_time(t2)
                             write(*,'("  invert_n2(a): ",f7.2)') (t2-t1)/t12
call cpu_time(t1); do k=1,kmax; call invert_n3(a); enddo; call cpu_time(t2)
                             write(*,'("  invert_n3(a): ",f7.2)') (t2-t1)/t12
call cpu_time(t1); do k=1,kmax; a=invert_n4(a); enddo; call cpu_time(t2)
                             write(*,'("  invert_n4(a): ",f7.2)') (t2-t1)/t12
n1=1; n2=n;
call cpu_time(t1); do k=1,kmax; call invert_r1(a,n,n1,n2); enddo;
call cpu_time(t2); write(*,'("invert_r1,n1,n2(a,n): ",f7.2)') (t2-t1)/t12
n1=1; n2=n;
call cpu_time(t1); do k=1,kmax; call invert_r2(a,n1,n2); enddo
call cpu_time(t2); write(*,'("invert_r2(a,n1,n2)  : ",f7.2)') (t2-t1)/t12
deallocate(a)
end program test3a
```

Попытка сравнения временны́х замеров времени CPU_TIME и gprof. Из **flat**-профиля видно, что временны́е затраты на 1001 вызов процедуры **invert_n1 gprof** оценил в 0.09с. Поэтому, можно ожидать, что отношение временны́х **gprof**-затрат остальных процедур к **0.09** не должны сильно отличаться от отношения к **t12** соответствующих временны́х замеров **CPU_TIME**.

```
 kmax=        1001                        !       self second   t2-t1
    n=       16001 (число_элементов_a)     !       -----------   -----
   Процедура      Время                    !          0.09        t12
invert_n1(a,n):    1.00                   !   r2     2.8         3.4
  invert_n2(a):    0.81                   !   r1     1.8         2.6
  invert_n3(a):    0.58                   !   n1     1.0         1.0
  invert_n4(a):    0.55                   !   n2     0.8         0.81
invert_r1,n1,n2(a,n):    2.63            !   n3     0.55        0.58
invert_r2(a,n1,n2)  :    3.38            !   n4     0.22        0.55
                                          !
real    0m0.823s                          !
user    0m0.819s                          !       0.00819*%time  gprof
sys     0m0.004s                          !       --------------------
                                          !   r2     0.28         0.22
Аналогично, на основе известных           !   r1     0.19         0.15
                                          !   n1     0.12         0.09
%time и user, можно сравнить соответ-     !   n2     0.09         0.07
ствующие затраты времени в секундах:      !   n3     0.066        0.05
                                          !   n4     0.026        0.02
```

```
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 37.15      0.25     0.25     1001     0.24     0.24  __sort_MOD_invert_r2
 23.50      0.40     0.16     1001     0.15     0.15  __sort_MOD_invert_r1
 13.65      0.49     0.09     1001     0.09     0.09  __sort_MOD_invert_n1
 10.61      0.56     0.07     1001     0.07     0.07  __sort_MOD_invert_n2
  7.58      0.61     0.05     1001     0.05     0.05  __sort_MOD_invert_n3
  4.55      0.64     0.03        1    30.02   660.51  MAIN__
  3.03      0.66     0.02     1001     0.02     0.02  __sort_MOD_invert_n4
```

4.4 Задача 4

Разработать функцию циклического сдвига элементов вектора, т.е. при сдвиге вправо содержимое **k**-го элемента переносится в **k+1**-й элемент (**k=1, ... , n**) при переносе содержимого **последнего** элемента в **первый**; а при сдвиге влево содержимое **k**-го переносится в **k-1**-й (**k=2, ... , n**), при переносе содержимого **первого** элемента в **последний**.

```
module sort; implicit none
contains ! ...   ...   ...   ...   ...   ...   ...
subroutine cyclar1(a,n,key); integer a(n), n, w, i; logical key
if (key) then; w=a(n); do i=n,2,-1; a(i)=a(i-1); enddo; a(1)=w
         else; w=a(1); do i=1, n-1; a(i)=a(i+1); enddo; a(n)=w
endif
end subroutine cyclar1
subroutine cyclar2(a,key); integer a(:), n, w; logical key; n=size(a)
if (key) then; w=a(n); a(2:n)=a(1:n-1); a(1)=w
         else; w=a(1); a(1:n-1)=a(2:n); a(n)=w
endif
end subroutine cyclar2
end module sort

program test4; use sort; implicit none; integer ier, i, n
integer, allocatable :: a(:)
character(60) s, sf; character( 2) sn
read (*,*) n, s; write(*,*) '  n=', n, trim(s); allocate(a(n),stat=ier)
if (ier/=0) then; write(*,*) 'a: not allocate.'; stop 1; endif
a=(/(i,i=1,n)/)
write(sn,'(i2)') n; sf='(a,'//sn//'i3)'; write(*,trim(sf)) "a:",a(1:n)
call cyclar1(a,n, .true.); write(*,trim(sf)) "cyclar1(a,n, .true.): a=", a(1:n)
call cyclar2(a,   .true.); write(*,trim(sf)) "cyclar2(a,   .true.): a=", a(1:n)
call cyclar2(a,  .false.); write(*,trim(sf)) "cyclar2(a,  .false.): a=", a(1:n)
call cyclar1(a,n,.false.); write(*,trim(sf)) "cyclar1(a,n,.false.): a=", a(1:n)
write(*,*); do i=1,n; call cyclar1(a,n,.true.)
                      write(*,trim(sf)) "cyclar1(a,n, .true.): a=", a(1:n)
            enddo
write(*,*); do i=1,n; call cyclar2(a,.false.)
                      write(*,trim(sf)) "cyclar2(a,  .false.): a=", a(1:n)
            enddo
deallocate(a)
end program test4
```

1. Строка с многоточием — процедуры, не востребованные в задаче.
2. Назначение переменных **s**, **sn** и **sf** та же, что и в задаче №3.
3. Процедура **cyclar1(a,n,key)** написана в стиле старого ФОРТРАНа.

4. Процедура **cyclar2(a,key)** обходится без аргумента, указывающего число используемых элементов массива **a**, поскольку формальный аргумент **a** описан как массив, перенимающий форму (**integer a(:)**), так что размер фактического аргумента можно определить внутри процедуры, используя функцию **size**.

5. **cyclar2(a,key)** использует **индексные триплеты**, обходясь вообще без оператора цикла.

6. В процедурах аргумент **key** логического типа: при **key=.true.** массив циклически сдвигается вправо, а при **key=.false.** — влево.

7. Обратите внимание: нет нужды писать **if (key.eqv..true.)** (хотя и можно) — достаточно **if (key)**.

Результат работы программы test4.

```
   n=          10 (число_элементов_a)
a:  1  2  3  4  5  6  7  8  9 10
cyclar1(a,n, .true.): a= 10  1  2  3  4  5  6  7  8  9
cyclar2(a,   .true.): a=  9 10  1  2  3  4  5  6  7  8
cyclar2(a,  .false.): a= 10  1  2  3  4  5  6  7  8  9
cyclar1(a,n,.false.): a=  1  2  3  4  5  6  7  8  9 10

cyclar1(a,n, .true.): a= 10  1  2  3  4  5  6  7  8  9
cyclar1(a,n, .true.): a=  9 10  1  2  3  4  5  6  7  8
cyclar1(a,n, .true.): a=  8  9 10  1  2  3  4  5  6  7
cyclar1(a,n, .true.): a=  7  8  9 10  1  2  3  4  5  6
cyclar1(a,n, .true.): a=  6  7  8  9 10  1  2  3  4  5
cyclar1(a,n, .true.): a=  5  6  7  8  9 10  1  2  3  4
cyclar1(a,n, .true.): a=  4  5  6  7  8  9 10  1  2  3
cyclar1(a,n, .true.): a=  3  4  5  6  7  8  9 10  1  2
cyclar1(a,n, .true.): a=  2  3  4  5  6  7  8  9 10  1
cyclar1(a,n, .true.): a=  1  2  3  4  5  6  7  8  9 10

cyclar2(a,  .false.): a=  2  3  4  5  6  7  8  9 10  1
cyclar2(a,  .false.): a=  3  4  5  6  7  8  9 10  1  2
cyclar2(a,  .false.): a=  4  5  6  7  8  9 10  1  2  3
cyclar2(a,  .false.): a=  5  6  7  8  9 10  1  2  3  4
cyclar2(a,  .false.): a=  6  7  8  9 10  1  2  3  4  5
cyclar2(a,  .false.): a=  7  8  9 10  1  2  3  4  5  6
cyclar2(a,  .false.): a=  8  9 10  1  2  3  4  5  6  7
cyclar2(a,  .false.): a=  9 10  1  2  3  4  5  6  7  8
cyclar2(a,  .false.): a= 10  1  2  3  4  5  6  7  8  9
cyclar2(a,  .false.): a=  1  2  3  4  5  6  7  8  9 10
```

4.5 Задача 5

Разработать функцию проверки вектора на упорядоченность.

Алгоритм проверки для вектора с элементами целого типа прост, так как целочисленные значения в пределах допустимого диапазона записываются в соответствующие переменные абсолютно точно. Возможны лишь два варианта ответа: **упорядочен** или **неупорядочен**. Функция (назовём её **increasI1(a,n)**) может возвращать булево значение **.true.**, если вектор упорядочен по возрастанию, и **.false.**, если неупорядочен.

Для вектора типа **real** алгоритм, реализованный для целых чисел, не всегда позволит судить об упорядоченности, так как значения двух соседних элементов могут отличаться друг от друга в единице самого младшего разряда. Например, мы не можем со 100% гарантией утверждать, что на типе **real(4)** значения **1.4000000**, **1.3999999**, **1.4000001** изображают разные числа — в процессе расчёта каждого из них младший разряд мог быть заражён погрешностью округления (а мог быть и не заражён). Так что различие значений или обусловлено накоплением погрешности или нет, причём причина их различия неизвестна.

Если погрешность равна нулю, то — упорядоченность нарушена, но если погрешность ненулевая, то вполне возможно, что эти три значения могли бы считаться и равными в пределах старших восьми разрядов.

Хотелось бы, чтобы функция при обнаружении подобной ситуации сообщала о возможности двоякого толкования. Код (типа **integer**) результата анализа упорядоченности по возрастанию **real**-значений возвращается через имя функции **increasR**:

1. **0** — упорядоченность фиксируется однозначно, т.е. модуль разности двух соседних элементов оказывается больше **eps**-доли меньшего.

   eps должен определить пользователь, ориентируясь на количество верных значащих цифр в исходных данных.

2. **1** — упорядоченность нарушена в пределах **eps**-доли меньшего элемента (причина нарушения неизвестна), необходим дополнительный анализ для принятия окончательного решения.

3. **2** — фиксировано различие большее **eps**-доли меньшего из сравниваемых элементов, т.е. установлено, что нарушение упорядоченности не может быть следствием накопления ошибок.

```
module sort
implicit none
contains
!   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
function increas_I1(a,n) result(okey); integer n, a(n); logical okey
integer i
okey=.true.
i=1
do while (okey.and.i<n)
  okey=a(i)<=a(i+1)
  i=i+1
enddo
end function increas_I1

function increas_I2(a) result(okey); integer  a(:); logical okey
integer i, n
n=size(a)
okey=.true.
i=1
do while (okey.and.i<n)
  okey=a(i)<=a(i+1)
  i=i+1
enddo
end function increas_I2

function increasR(a,eps) result(key); real a(:), eps
integer key, i, n
real d
n=size(a)
key=0           ! Полагаем, что вектор нестрого упорядочен по возрастанию.
d=0.0           ! Начальная настройка разности двух соседних элементов.
do while(d<=0.0.and.i<n)
  d=a(i)-a(i+1)
  if (d>eps*abs(a(i))) then             ! Полагаем, что упорядоченность
      key=2; exit                       ! по возрастанию нарушена;
    else                                          ! полагаем, что нельзя
      if (d/=0.0.and.abs(d)<eps*abs(a(i))) key=1 ! объективно судить о
  endif                                           ! нарушении возрастания.
  i=i+1
enddo
end function increasR
end module sort
```

```
program test5; use sort; implicit none; integer ier, i, n
integer, allocatable :: a(:)
real,    allocatable :: b(:), c(:)
real eps
character(60) s, sf; character( 2) sn
read (*,*) n, s; write(*,*) '  n=', n, trim(s)
allocate(a(n), b(n),stat=ier)
if (ier/=0) then; write(*,*) 'a: not allocate.'; stop 1; endif
read(*,*) a
write(sn,'(i2)') n; sf='(a,'//sn//'i3'//',$)'
write(*,trim(sf)) "a:",a(1:n)
write(*,'( 1x,a,l2)') "increas_I1(a,n)=", increas_I1(a,n)
write(*,'(27x,a,l2)') "increas_I2( a )=", increas_I2( a )
read(*,*) a
write(*,trim(sf)) "a:",a(1:n)
write(*,*) "increas_I1(a,n)=", increas_I1(a,n)
write(*,'(27x,a,l2)') "increas_I2( a )=", increas_I2( a )
eps=epsilon(0.0)
do
  read(*,*,end=77) b; write(*,*) ' b:'
  write(*,'(i3,e16.8,i8,e16.8)') (i, b(i),i+n/2, b(i+n/2), i=1,n/2)
  write(*,'(a,e9.2,a,i3)') "increasR(b,",eps,")=", increasR(b,eps)
enddo
77 continue
deallocate(a)
end program test5
```

Файл input

```
  8   (число_элементов_a)
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 0
1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8
1.1 1.2 1.3 1.4000000 1.4000001 1.6 1.7 1.8
1.1 1.2 1.3 1.4000000 1.3999995 1.6 1.7 1.8
1.1 1.2 1.3 1.4000000 1.3999998 1.6 1.7 1.8
1.1 1.2 1.3 1.4000000 1.40000001 1.6 1.7 -1.8
1.25 1.25 1.25 1.5  1.5  1.5  1.75  1.75
```

1. Первый набор из целых чисел упорядочен; второй — нет.
2. Третий набор из чисел типа **real(4)** упорядочен.
3. Четвёртый тоже упорядочен, т.к. значение **1.4000001** введено. Если бы оно получалось в результате вычислений, то вывод об упорядоченности мог быть под вопросом.

4. Пятый набор: при **eps=epsilon(0.0)** различие больше **eps**-доли — функция *полагает*, что набор неупорядочен.

5. Шестой набор аналогичен четвёртому; различие элементов в пределах допуска **eps** — поэтому набор може оказаться и упорядоченным.

```
   n=           8 (число_элементов_a)          ! Результат работы test5
a:  1  2  3  4  5  6  7  8 increas_I1(a,n)= T  ! с данными из файла input.
                           increas_I2( a )= T
a:  1  2  3  4  5  6  7  0 increas_I1(a,n)= F
                           increas_I2( a )= F
  b:
  1  0.11000000E+01       5  0.15000000E+01
  2  0.12000000E+01       6  0.16000000E+01
  3  0.13000000E+01       7  0.17000000E+01
  4  0.14000000E+01       8  0.18000000E+01
increasR(b, 0.12E-06)=  0
  b:
  1  0.11000000E+01       5  0.14000001E+01
  2  0.12000000E+01       6  0.16000000E+01
  3  0.13000000E+01       7  0.17000000E+01
  4  0.14000000E+01       8  0.18000000E+01
increasR(b, 0.12E-06)=  1
  b:
  1  0.11000000E+01       5  0.13999995E+01
  2  0.12000000E+01       6  0.16000000E+01
  3  0.13000000E+01       7  0.17000000E+01
  4  0.14000000E+01       8  0.18000000E+01
increasR(b, 0.12E-06)=  2
  b:
  1  0.11000000E+01       5  0.13999999E+01
  2  0.12000000E+01       6  0.16000000E+01
  3  0.13000000E+01       7  0.17000000E+01
  4  0.14000000E+01       8  0.18000000E+01
increasR(b, 0.12E-06)=  1
  b:
  1  0.11000000E+01       5  0.14000000E+01
  2  0.12000000E+01       6  0.16000000E+01
  3  0.13000000E+01       7  0.17000000E+01
  4  0.14000000E+01       8 -0.18000000E+01
increasR(b, 0.12E-06)=  2
  b:
  1  0.12500000E+01       5  0.15000000E+01
  2  0.12500000E+01       6  0.15000000E+01
  3  0.12500000E+01       7  0.17500000E+01
  4  0.15000000E+01       8  0.17500000E+01
increasR(b, 0.12E-06)=  0
```

4.6 Задача 6

Разработать нерекурсивную и рекурсивную функции поиска образца среди элементов упорядоченного вектора (т.е. либо найти индекс элемента равного образцу, либо указать на отсутствие такового).

Если величины элементов вектора не упорядочены, то процедура поиска образца (т.е. числа равного одному из элементов вектора) требует в худшем случае перебора всех элементов.

Если же вектор упорядочен, то можно применить алгоритм бинарного поиска (метод дихотомии). Пусть **[n0,nn]** диапазон индексов элементов массива, среди которых ищется образец **x**. Пока поиск не закончен (т.е. пока **n0<=nn**) находим индекс **ns=(n0+nn)/2** элемента, находящегося в середине вектор. Если он равен образцу, то поиск закончен. Если же **x<a(ns)**, то в силу упорядоченности вектора по возрастанию достаточно проанализировать лишь элементы из диапазона индексов **[n0,ns-1]**, а если **x>a(ns)**, то — из диапазона индексов **[ns+1,nn]**.

```
module sort; implicit none
contains  !   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
function okey0(a,x) result (num)         ! Процедура поиска образца в
integer a(:), x, i, n, num               ! неупорядоченном массиве.
n=size(a)
num=0; do i=1,n; if (x==a(i)) then; num=i; exit; endif; enddo
end function okey0
function okey1(a,x) result (num)         ! Нерекурсивная функция поиска
integer a(:), x, i, n, num, n0, nn, ns   ! образца x в векторе a.
n=size(a); n0=1; nn=n;
num=0                          ! 0 --- индикатор отсутствия образца.
if (x>=a(1).and.x<=a(n)) then! Если образец возможен в [a(1),a(n)], то
  do while (n0<=nn)            ! пока начальный индекс не больше конечного
    ns=(n0+nn)/2               ! находим индекс среднего элемента массива.
    if (x == a(ns)) then       ! Если это образец, то
       nn=ns; exit             !   запоминаем индекс и выходим из цикла;
      else                     ! иначе
        if (x < a(ns)) then    ! если образец меньше среднего элемента, то
                nn=ns-1        ! сужаем вдвое исследуемый фрагмент массива,
                               ! изменяя конечный индекс на ns-1;
          else; n0=ns+1        ! если же больше, то сужение достигается за
        endif                  ! счёт изменения начального индекса на ns+1
    endif
  enddo                        ! Результат либо 0, если образца нет,
  num=nn                       ! либо наименьший индекс элемента равного
endif                          ! образцу x.
end function okey1
```

```
recursive function okey2(a,x, n0, nn) result (num)
integer a(:), x, i, n, num, n0, nn, ns
n=size(a)
if (x<a(1) .or. x>a(n)) then; num=0
  else
    ns=(n0+nn)/2
    if (x==a(ns)) then; num=ns
       else
         if (x < a(ns)) then; num=okey2(a,x,n0,ns-1)
                        else; num=okey2(a,x,ns+1,nn)
        endif
    endif
endif
end function okey2
end module sort

program test6
use sort
implicit none
integer ier, i, n
integer, allocatable :: a(:)
integer x, m0, m1, m2
character(60) s, sf; character( 2) sn
read (*,*) n, s; write(*,*) '  n=', n, trim(s)
allocate(a(n),stat=ier)
if (ier/=0) then
  write(*,*) 'a: not allocate.'
  stop 1
endif
read(*,*) a
write(sn,'(i2)') n
sf='(a,'//sn//'i3'//')'
write(*,trim(sf)) "a:",a(1:n)
write(*,'(4x,"x",8x,"okey0",3x,"okey1",3x,"okey2")')
do x=0,16
  m0=okey0(a,x)
  m1=okey1(a,x)
  m2=okey2(a,x,1,n)
  write(*,'(I5,3x,3x,I5,3x,I5,3x,I5)') x, m0,m1,m2
enddo
deallocate(a)
end program test6
```

```
   n=           8 (число_элементов_a)
a:  1  3  5  6  8 10 10 15
    x        okey0   okey1   okey2
    0          0       0       0
    1          1       1       1
    2          0       0       0
    3          2       2       2
    4          0       0       0
    5          3       3       3
    6          4       4       4
    7          0       0       0
    8          5       5       5
    9          0       0       0
   10          6       6       6
   11          0       0       0
   12          0       0       0
   13          0       0       0
   14          0       0       0
   15          8       8       8
   16          0       0       0
```

1. Заметим, что хотя наши процедуры и решают поставленную задачу, но трактуют массив, поданный на обработку, в соответствии с описанием формального аргумента **integer a(:)**, т.е. индексация массива трактуется процедурами по умолчанию, а именно нижний индекс равен **единице**. Например, в программе

```
program test6a; use sort; implicit none; integer ier, i, n
integer, allocatable :: a(:)
integer x, n0, nn, m0, m1, m2
character(60) s, sf; character( 2) sn
read (*,*) n, s; write(*,*) '  n=', n, trim(s)
allocate(a(n),stat=ier)
if (ier/=0) then; write(*,*) 'a: not allocate.'; stop 1; endif
read(*,*) a
write(sn,'(i2)') n; sf='(a,'//sn//'i3'//')'
write(*,trim(sf)) "a:",a(1:n)
write(*,'(4x,"x",8x,"okey0",3x,"okey1",3x,"okey2")')
do x=0,16
  m0=okey0(a(3:7),x)
  m1=okey1(a(3:7),x)
  m2=okey2(a(3:7),x,1,n)
  write(*,'(I5,3x,3x,I5,3x,I5,3x,I5)') x, m0,m1,m2
enddo
deallocate(a)
end program test6a
```

на обработку процедурам подан не весь массив **a(1:n)**, а лишь его секция **a(3:7)**. Другими словами, ищем образец не во всём массиве, а лишь в его части (с **третьего** по **седьмой** элемент включительно). В этом случае результат

```
   n=           8 (число_элементов_a)
a:  1  3  5  6  8 10 10 15
    x        okey0   okey1   okey2
    0          0       0       0
    1          0       0       0
    2          0       0       0
    3          0       0       0
    4          0       0       0
    5          1       1       1
    6          2       2       2
    7          0       0       0
    8          3       3       3
    9          0       0       0
   10          4       4       4
   11          0       0       0
   12          0       0       0
   13          0       0       0
   14          0       0       0
   15          0       0       0
   16          0       0       0
```

придётся осмысливать, так как окажется, что образец **5** находится в элементе массива с индексом **1**, образец **6** — в элементе массива с индексом **2** и т.д. Результат, конечно, верен, так как внутри процедур описание **a(:)** трактовало фактический аргумент **a(3:7)** как формальный **a(1:4)**.

2. Для того чтобы получить индексацию элементов исходного массива достаточно к найденным индексам добавить **2**, т.е. индекс начального элемента вырезки без единицы.

4.7 Задача 7

Разработать подпрограмму и функцию объединения двух упорядоченных массивов, обеспечивая упорядоченность массива-результата.

Задача решается вызовом подпрограммы **sort_merge_1** или функции **sort_merge_2**. Последняя демонстрирует возврат массива через имя функции (в старом ФОРТРАНе это было невозможно).

```
module sort
implicit none
contains !   ...   ...   ...   ...   ...   ...   ...   ...   ...
subroutine sort_merge_1(a, b, c)
integer a(:), b(:), c(:)
integer na, nb, i, j, k
na=size(a)
nb=size(b)
i=1; j=1; k=1;
do while (i<=na.and.j<=nb)
  if (a(i) < b(j)) then; c(k)=a(i); i=i+1
                   else; c(k)=b(j); j=j+1
  endif
  k=k+1
enddo
do while(i<=na); c(k)=a(i); k=k+1; i=i+1; enddo
do while(j<=nb); c(k)=b(j); k=k+1; j=j+1; enddo
end subroutine sort_merge_1

function sort_merge_2(a, b) result(c)
integer a(:), b(:), c(size(a)+size(b))
integer na, nb, i, j, k
na=size(a)
nb=size(b)
i=1; j=1; k=1;
do while (i<=na.and.j<=nb)
  if (a(i) < b(j)) then; c(k)=a(i); i=i+1
                   else; c(k)=b(j); j=j+1
  endif
  k=k+1
enddo
do while(i<=na); c(k)=a(i); k=k+1; i=i+1; enddo
do while(j<=nb); c(k)=b(j); k=k+1; j=j+1; enddo
end function sort_merge_2
end module sort
```

Строка с многоточиями обозначает модульные процедуры, не востребованные в главной программе.

```
program test7; use sort; implicit none
integer, allocatable :: a(:), b(:), c(:)
integer na, nb, nc, ier
character(3) sa, sb, sc; character(60) sfa, sfb, sfc
read(*,*) na, nb
nc=na+nb
write(*,'(a,i4,5x,a,i4,5x,a,i4)') ' na=', na, '  nb=', nb, '  nc=', nc
allocate(a(na), b(nb), c(nc), stat=ier)
if (ier/=0) then
  write(*,*) ' not allocate'
  stop 1
endif
write(sa,'(i3)') na; sfa='(a,'//sa//'I4)'
write(sb,'(i3)') nb; sfb='(a,'//sb//'i4)'
write(sc,'(i3)') nc; sfc='(a,'//sc//'i4)'
read (*,*) a, b
write(*,trim(sfa)) ' a:',a
write(*,trim(sfb)) ' b:',b
     call sort_merge_1(a,b,c); write(*,trim(sfc)) 'sort_merge_1(a,b,c):',c
c=0; call sort_merge_1(b,a,c); write(*,trim(sfc)) 'sort_merge_1(b,a,c):',c
c=0; c=sort_merge_2(a,b);      write(*,trim(sfc)) '  sort_merge_2(a,b):',c
c=0; c=sort_merge_2(b,a);      write(*,trim(sfc)) '  sort_merge_2(b,a):',c
deallocate(a,b,c);
end program test7
```

Главная программа:

1) вводит **na** и **nb** (числа элементов в размещаемых векторах **a** и **b**) и находит **nc** — число элементов в размещаемом векторе **c**;

2) размещает вектора **a**, **b** и **c** в оперативной памяти (при неудаче размещения программа завершает работу с кодом завершения **1**);

3) использует символьные переменные **sa**, **sb** и **sc** в качестве внутренних файлов с целью автоматической вставки нужных повторителей в форматные строки **sfa**, **sfb**, **sfc**;

4) вводит **na** упорядоченных по возрастанию значений элеменов вектора **a** и **nb** упорядоченных значений элементов вектора **b**, после чего осуществляет контрольный вывод обоих векторов;

5) вызывает подпрограмму **sort_merge_1(a, b, c)**, осуществляя слияние двух упорядоченных векторов **a** и **b** в один упорядоченный вектор **c** (в первый раз первый аргумент — более длинный массив, во второй раз — более короткий;

6) вызывает функцию **sort_merge_2(a,b)**, которая упорядоченный массив возвращает через своё имя.

7) Обнуление вектора **c** перед вызовом очередной процедуры выполняется во избежание вывода результата предыдущего слияния.

Результат работы программы test7.

```
 na=   8       nb=   4       nc=  12
 a:   1   3   5   7   9  11  13  16
 b:   2   4   6   8
sort_merge_1(a,b,c):   1   2   3   4   5   6   7   8   9  11  13  16
sort_merge_1(b,a,c):   1   2   3   4   5   6   7   8   9  11  13  16
  sort_merge_2(a,b):   1   2   3   4   5   6   7   8   9  11  13  16
  sort_merge_2(b,a):   1   2   3   4   5   6   7   8   9  11  13  16
```

Вариант СИ-решения задачи

```
#include<stdio.h>
void sort_merge_1(int*, int*, int*, int, int);
int main()
{
int i, j, c[102], a[51], b[51];
int na, nb;
scanf("%d %d", &na, &nb);
for (i=0; i<na; i++)  { scanf("%d", &a[i]); }
for (i=0; i<nb; i++)  { scanf("%d", &b[i]); }
 printf("a:"); for (i=0; i<na; i++) { printf(" %d", a[i]); } printf("\n");
 printf("b:"); for (i=0; i<nb; i++) { printf(" %d", b[i]); } printf("\n");
sort_merge_1(a,b,c, na,nb);
printf("c:"); for (i=0; i<na+nb; i++) {printf(" %d", c[i]);} printf("\n");
sort_merge_1(b,a,c, nb,na);
printf("c:"); for (i=0; i<na+nb; i++) {printf(" %d", c[i]);}
return 0;
}

void sort_merge_1(int* a, int*b, int*c, int na, int nb)
{ int i, j, k;
  i=j=k=0;
  while(i<na && j<nb)
    {
      if (a[i] < b[j]) c[k++]=a[i++];
        else           c[k++]=b[j++];
    }
  while(i<na) c[k++]=a[i++];
  while(j<nb) c[k++]=b[j++];
}
```

4.8 Задача 8

Значение интеграла по промежутку **[a,b]** от подинтегральной функции **f(x)** приближенно вычисляется по формуле трапеций:

$$\tilde{\mathbf{S}} = \mathbf{h} \cdot \Big[\frac{\mathbf{f(a)} + \mathbf{f(b)}}{\mathbf{2}} + \sum_{\mathbf{i=2}}^{\mathbf{n}} \mathbf{f}(\mathbf{x_i})\Big], \quad \text{где}$$

$$\mathbf{h} = \frac{\mathbf{b-a}}{\mathbf{n}}; \qquad \mathbf{x_i} = \mathbf{a} + (\mathbf{i-1})\mathbf{h} \qquad \mathbf{i} = \mathbf{2}, \cdots \mathbf{n}; \qquad \mathbf{x_1} \equiv \mathbf{a} \;\; , \;\; \mathbf{x_{n+1}} \equiv \mathbf{b}.$$

n – количество промежутков равномерного дробления отрезка **[a,b]**.

Разработать функцию **trap(y,a,b,n)**, которая по набору из **n+1** значений подинтегральной функции, хранящихся в первых **n+1** элементах одномерного массива **y** вычисляет значение квадратурной суммы формулы трапеций. Описание функции **trap** поместить в модуль **quarda** (тот же самый, в который помещена и процедура **rectan** из предыдущего домашнего задания).

Главная программа должна подсоединять соответствующий модуль, вводить исходные данные из файла, вызывать нужную процедуру и выводить результат вместе с таблицей подинтегральной функции так, чтобы получить её график, активируя цель **make plot**.

Для расчёта интегралов наряду с формулами прямоугольников и трапеций широко используется **составная формула Симпсона**:

$$\tilde{\mathbf{S}} = \frac{\mathbf{h}}{\mathbf{3}} \cdot [\mathbf{f(a)} + \mathbf{f(b)} + \mathbf{4} \cdot \mathbf{S_1} + \mathbf{2} \cdot \mathbf{S_2}]$$

$$\mathbf{S_1} = \mathbf{f_2} + \mathbf{f_4} + \cdots + \mathbf{f_{n=2m}} \;\; , \;\; \mathbf{S_2} = \mathbf{f_3} + \mathbf{f_5} + \cdots + \mathbf{f_{n-1}}.$$

$$\mathbf{f_i} = \mathbf{f}(\mathbf{x_i}) \;\; \mathbf{x_i} = \mathbf{a} + \mathbf{h} \cdot (\mathbf{i-1}), \;\; \mathbf{i} = \mathbf{1}, \mathbf{2}, \cdots, (\mathbf{n} = \mathbf{2m}), \mathbf{n+1}, \;\; \mathbf{h} = \frac{\mathbf{b-a}}{\mathbf{n}}$$

причём **n** (число промежутков дробления) — обязательно **чётное**. Учитывая требования к решению предыдущей задачи, разработать функцию **sim(y,a,b,n)**.

4.8.1 Некоторые замечания к решению

1. По условию задачи в качестве одного из аргументов функций, вычисляющих квадратурные суммы, должна быть задана длина вектора, элементы которого хранят значения подынтегральной функции в узлах дискретизации.

2. Все задачи четвёртого задания по теме «**размещаемые массивы**». Размещение должна осуществлять главная программа, а вызываемые процедуры должны тем или иным способом лишь использовать вектора, подготовленные главной программой.

3. Удобно в одной главной программе протестировать работу и **rectan**, и **trap**, и **sim**, так как в некоторых тестах их результаты должны совпадать полностью, что при выводе может служить дополнительным контролем.

4. Для тестирования процедур интегрирования естественно выбрать прежде всего такие подинтегральные функции, для которых квадратурные суммы должны получить точное (в пределах используемой разрядности) значение интеграла. Для формул средних прямоугольников и трапеций это — линейные функции, а для формулы Симпсона — даже кубический полином. Поэтому для тестирования процедур выбраны функции $\mathbf{x}$, $\mathbf{x^2}$, $\mathbf{x^3}$ и $\mathbf{x^4}$, для которых просто. вычислить и точные значения интегралов, что позволит оценить и относительные погрешности квадратурных сумм.

5. Поскольку значения подинтегральных функций передаются процедурам в в виде вектора, то задать значения его элементов должна главная программа.

6. Есть два способа задания: 1) ввести значения элементов и 2) вычислить их. Естественно второй способ более предпочтителен, когда речь идёт о тестировании. Поэтому в главной программе описаны внутренние функции $\mathbf{f1(x)}=\mathbf{x}$, $\mathbf{f2}(\mathbf{x}) = \mathbf{x^2}$, $\mathbf{f3}(\mathbf{x}) = \mathbf{x^3}$ и $\mathbf{f1}(\mathbf{x}) = \mathbf{x^4}$, и имеется фрагмент, который заполняет элементы векторов посредством вызова этих функций для соответствующих значений точек дискретизации промежутка **[a,b]**.

7. И главная программа, и модуль **quadra**, в котором описаны процедуры интегрирования, используют модуль **my_prec**, так что с лёгкостью можно вести расчёты на всех допустимых разновидностях типа **real(mp)**.

8. Процедуры используют встроенную **sum** и индексные триплеты.

9. Современный ФОРТРАН позволяет не передавать в качестве аргумента процедуры интегрирования число используемых элементов вектора (если передача нам невыгодна), определяя это число внутри процедуры посредством вызова встроенной функции **size**. Поэтому в модуле **quadra** описано два семейства процедур:

   - **rectan0(y,a,b,n)**, **trap0(y,a,b,n)** и **sim0(y,a,b,n)**;
   - **rectan1(y,a,b)**, **trap1(y,a,b)** и **sim1(y,a,b)**;

   а главная программа тестирует оба семейства.

4.8.2 Исходный текст модуля my_prec

```
 module my_prec              !                        Файл my_prec.f90
 implicit none
 integer, parameter :: mp=16 ! <--= тут задаём mp=4, 8, 10, 16
 contains
 function frmp() result(f)   ! Функция frmp() по значению mp возвращает
 character(60) f             ! строку, хранящую дескриптор вывода
 character(10),parameter :: frm(4)=              ! одного значения
>(/'(e14.6 )','(e24.14)','(e27.18)','(e42.35)'/) ! типа real(mp)
 integer m
 m=int(mp/4.0+0.6)
 f=frm(m)
 end function frmp
 end module my_prec
```

4.8.3 Исходный текст модуля quadra

```
module quadra; use my_prec; implicit none
contains
                                    ! RECTAN0 находит квадратурную
function rectan0(y,a,b,n) result(s) ! сумму n-точечной формулы
integer n                           ! средних прямоугольников по
real(mp) y(n), a, b, s, h           ! [a,b] для функции
h=(b-a)/n; s=h*sum(y)               ! табулированной в векторе
end function rectan0                !                     y(n).
                                      ! TRAP0 находит квадратурную
function trap0(y,a,b,n1) result(s)    ! сумму формулы трапеций по
integer n1, n                         ! промежутку [a,b] для функции
real(mp) y(n1), a, b, s, h            ! табулированной в векторе
n=n1-1; h=(b-a)/n;                    ! y(n+1). n --- число участков
s=(y(1)+y(n1))/2; s=h*(s+sum(y(2:n))) ! равномерного дробления [a,b]
end function trap0
                                      ! SIM0 находит квадратурную
function sim0(y,a,b,n1) result(s)     ! сумму формулы Симпсона
integer n1, n                         ! промежутку [a,b] для
real(mp) y(n1), a, b, s, h            ! функции, табулированной
n=n1-1; h=(b-a)/n                     ! в векторе y(n1). n1=n+1
s=y(1)+y(n1)
s=s+4*sum(y(2:n:2))+2*sum(y(3:n-1:2))
s=s*h/3                               ! n должно быть ЧЁТНО !!!
end function sim0
                                    ! RECTAN1 находит квадратурную
function rectan1(y,a,b) result(s)   ! сумму формулы средних прямо-
real(mp) y(:), a, b, s, h           ! угольников по [a,b] для функ-
h=(b-a)/size(y); s=h*sum(y)         ! ции, табулированной в y(:)
end function rectan1
                                    ! TRAP1 находит квадратурную
function trap1(y,a,b) result(s)     ! сумму формулы трапеций по
integer n                           ! [a,b] для функции, табули-
real(mp) y(:), a, b, s, h           ! рованной в векторе y(:),
n=size(y)                           ! перенимающем форму фактиче-
h=(b-a)/(n-1); s=(y(1)+y(n))/2      ! ского аргумента
               s=h*(s+sum(y(2:n-1)))
end function trap1
                                      ! SIM1 находит квадратурную
function sim1(y,a,b) result(s)        ! сумму формулы Симпсона по
integer  n                            ! [a,b] для функции, табули-
real(mp) y(:), a, b, s, s1, s2, h     ! рованной в векторе y(:)
n=size(y); h=(b-a)/(n-1); s=y(1)+y(n) ! (перенимающем форму)
s1=sum(y(2:n:2)); s2=sum(y(3:n-1:2))  ! Число узлов дискретизации
s=s+4*s1+2*s2;     s=s*h/3            ! [a,b] (т.е. число элементов
end function sim1                     ! вектора должно быть НЕЧЁТНО).
end module quadra
```

1. Для формулы средних прямоугольников всё просто: число участков дробления совпадает с количеством точек дискретизации промежутка интегрирования. Поэтому в качестве размера вектора достаточно указать введённое **n**.

2. В случае формул трапеций и Симпсона число участков дробления на единицу меньше количества точек дискретизации. Возникает вопрос (актуальный в старых версиях ФОРТРАНа):

   Что выгоднее использовать в качестве аргумента процедуры:

   число точек или число промежутков.

   Ответ должен каждый сформулировать сам для себя.

   Кому-то приятнее видеть описание **real(mp) y(n)**, так как ясно, что элементов с индексом бóльшим **n** в процедуре быть не должно.

   Кто-то предпочтёт число промежутков (проще оценить величину шага), указав при описании аргумента **real(mp) y(n+1)**.

3. Современный ФОРТРАН позволяет не указывать в качестве входного аргумента количество узлов дискретизации, предоставляя возможность получить его посредством встроенной функции **size**.

   Однако, пользуясь этой возможностью, следует помнить, что для процедур с аргументами перенимающими форму (чего не было в старом ФОРТРАНе) в вызывающих программных единицах обязательно должен быть явно указан **interface** процедуры (т.е. атрибуты её аргументов): либо через **interface**-ный блок, либо через модуль. Часто, указав интерфейс в одной вызывающей процедуре, забывают указать его в других, которым он тоже необходим.

4.8.4 Исходный текст главной программы

```
program test_4_8; use my_prec; use quadra; implicit none
real(mp), allocatable :: y1(:), y2(:), y3(:), y4(:)
real(mp) a, b, h, x, r, t(4)
integer i, n, ier, n1;       character(60) fr,frm,frn
read(*,'(e10.3)') a, b;  read(*,'(i10)') n
fr=frmp();  frm='(a,2x,'//trim(fr)//')'
            frn='(5x,a,2x,'//trim(fr)//',7x,e10.2)'
write(*, '(a,i5)')  ' # mp =', mp
write(*,trim(frm))  ' #   a=', a
write(*,trim(frm))  ' #   b=', b
write(*, '(a,i10)') ' #   n=', n
t=exact(a,b)                                  ! Расчёт точного ответа
write(*,'(" # ",2x,"Exact")')
do i=1,4; write(*,*) t(i); enddo

write(*,'(" # rectan0:")')
allocate(y1(n),y2(n),y3(n),y4(n),stat=ier)   ! Размещение массивов.
if (ier/=0) then; write(*,*) 'not allocate'  ! Реакция на неудачу
                  stop 1                     ! при их размещении.
endif
h=(b-a)/n                                     ! Расчёт шага и
do i=1,n                                      ! заполнение массивов
  x=a+(i-0.5_mp)*h; y1(i)=f1(x); y2(i)=f2(x) ! подынтегральных
                    y3(i)=f3(x); y4(i)=f4(x) ! функций для RECTAN
enddo
write(*,'(4x,"Функция",3x,"Квадратурная сумма",3x,"Отн.погр")')
r=rectan0(y1,a,b,n); write(*,trim(frn)) '  x  ',r,abs(r-t(1))/t(1)
r=rectan0(y2,a,b,n); write(*,trim(frn)) ' x^2 ',r,abs(r-t(2))/t(2)
r=rectan0(y3,a,b,n); write(*,trim(frn)) ' x^3 ',r,abs(r-t(3))/t(3)
r=rectan0(y4,a,b,n); write(*,trim(frn)) ' x^4 ',r,abs(r-t(4))/t(4)
write(*,'(" # rectan1:")')
write(*,'(4x,"Функция",3x,"Квадратурная сумма",3x,"Отн.погр")')
r=rectan1(y1,a,b); write(*,trim(frn)) '  x  ',r,abs(r-t(1))/t(1)
r=rectan1(y2,a,b); write(*,trim(frn)) ' x^2 ',r,abs(r-t(2))/t(2)
r=rectan1(y3,a,b); write(*,trim(frn)) ' x^3 ',r,abs(r-t(3))/t(3)
r=rectan1(y4,a,b); write(*,trim(frn)) ' x^4 ',r,abs(r-t(4))/t(4)
deallocate(y1,y2,y3,y4,stat=ier)                ! Y-ки придётся
if (ier/=0) then                                ! переразмещать !
  write(*,*) 'not escape'; stop 1               ! Реакция на неудачу
endif                                           ! при высвобождении.
write(*,'(" #   trap0:")')
n1=n+1
allocate(y1(n1),y2(n1),y3(n1),y4(n1),stat=ier) ! Размещение массивов.
if (ier/=0) then; write(*,*) 'not allocate'    ! Реакция на неудачу
                  stop 2                       ! при их размещении.
endif
```

```
do i=1,n1                                          ! Заполнение массивов
  x=a+(i-1)*h; y1(i)=f1(x); y2(i)=f2(x)            ! подынтегральных
               y3(i)=f3(x); y4(i)=f4(x)            ! функций для TRAP.
enddo
r=trap0(y1,a,b,n1); write(*,trim(frn)) '  x  ',r,abs(r-t(1))/t(1)
r=trap0(y2,a,b,n1); write(*,trim(frn)) ' x^2 ',r,abs(r-t(2))/t(2)
r=trap0(y3,a,b,n1); write(*,trim(frn)) ' x^3 ',r,abs(r-t(3))/t(3)
r=trap0(y4,a,b,n1); write(*,trim(frn)) ' x^4 ',r,abs(r-t(4))/t(4)
write(*,'(" #   trap1:")')
write(*,'(4x,"Функция",3x,"Квадратурная сумма",3x,"Отн.погр")')
r=trap1(y1,a,b); write(*,trim(frn)) '  x  ',r,abs(r-t(1))/t(1)
r=trap1(y2,a,b); write(*,trim(frn)) ' x^2 ',r,abs(r-t(2))/t(2)
r=trap1(y3,a,b); write(*,trim(frn)) ' x^3 ',r,abs(r-t(3))/t(3)
r=trap1(y4,a,b); write(*,trim(frn)) ' x^4 ',r,abs(r-t(4))/t(4)
deallocate(y1,y2,y3,y4,stat=ier)                   ! Возможно n придётся
if (ier/=0) then                                   ! делать ЧЁТНЫМ.
   write(*,*) 'not escape'; stop 2                 ! Реакция на неудачу
endif                                              ! при высвобождении.
write(*,'(" #    sim0:")')
if (mod(n,2)/=0) then; n=n+1
            write(*,'(a,i10,a)') ' # n=',n,' must be even !!!'
endif
n1=n+1
allocate(y1(n1),y2(n1),y3(n1),y4(n1),stat=ier) ! Размещение массивов.
if (ier/=0) then; write(*,*) 'not allocate'    ! Реакция на неудачу
                  stop 3                       ! при их размещении.
endif
h=(b-a)/n                                          ! Перерасчёт шага и
do i=1,n1                                          ! Заполнение массивов
  x=a+(i-1)*h; y1(i)=f1(x); y2(i)=f2(x)            ! подынтегральных
               y3(i)=f3(x); y4(i)=f4(x)            ! функций для SIM.
enddo
write(*,'(4x,"Функция",3x,"Квадратурная сумма",3x,"Отн.погр")')
r=sim0(y1,a,b,n1); write(*,trim(frn)) '  x  ',r,abs(r-t(1))/t(1)
r=sim0(y2,a,b,n1); write(*,trim(frn)) ' x^2 ',r,abs(r-t(2))/t(2)
r=sim0(y3,a,b,n1); write(*,trim(frn)) ' x^3 ',r,abs(r-t(3))/t(3)
r=sim0(y4,a,b,n1); write(*,trim(frn)) ' x^4 ',r,abs(r-t(4))/t(4)
write(*,'(" #    sim1:")')
write(*,'(4x,"Функция",3x,"Квадратурная сумма",3x,"Отн.погр")')
r=sim1(y1,a,b); write(*,trim(frn)) '  x  ',r,abs(r-t(1))/t(1)
r=sim1(y2,a,b); write(*,trim(frn)) ' x^2 ',r,abs(r-t(2))/t(2)
r=sim1(y3,a,b); write(*,trim(frn)) ' x^3 ',r,abs(r-t(3))/t(3)
r=sim1(y4,a,b); write(*,trim(frn)) ' x^4 ',r,abs(r-t(4))/t(4)
deallocate(y1,y2,y3,y4,stat=ier)
if (ier/=0) then; write(*,*) 'not escape'  ! Реакция на неудачу
                  stop 2                   ! при высвобождении.
endif
stop 0
```

```
contains
function f1(x) result(w); real(mp) x, w; w=x;    end function f1
function f2(x) result(w); real(mp) x, w; w=x**2; end function f2
function f3(x) result(w); real(mp) x, w; w=x**3; end function f3
function f4(x) result(w); real(mp) x, w; w=x**4; end function f4
function exact(a,b) result(t)
real(mp) a, b, t(4)
integer i
do i=1,4                                        ! Расчёт точного
  t(i)=(b**(i+1)-a**(i+1))/(i+1)                ! результата для
enddo                                           ! тестирующих функций.
end function exact
end program test_4_8
```

В программе используются строковые переменные **character(60) fr, frm, frn** для формировки дескрипторов вывода некоторых результатов по формату, соответствующему типу **real(mp)**. Именно **e14.6** – для типа одинарной точности, **e24.14** – для удвоенной, **e27.18** – для расширенной и **e42.35** – для четверной. Алгоритм выбора формата упрятан в функции **frmp**, описанной в модуле **my_prec**.

Главная же программа просто присваивает переменной **fr** результат вызова **frmp**.

В переменной **frm** генерируется форматная строка для наглядного вывода параметров, используемых программой, с желательной для типа **real(mp)** разрядностью. Аналогичная цель по генерации формата вывода результата достигается формировкой переменной **frn**.

Размер длины строки в **60** символов — просто фрахтовка места с запасом для результата генерации. Из **60** символов всегда используется ровно столько сколько нужно для формировки, что достигается посредством вызова функции **trim**, убирающей хвостовые пробелы строки.

Безусловно, если касаться только темы: «Одномерный динамический массив», то работа со строками может показаться излишним усложнением. Однако, на простом примере полезно узнать и что-то новое.

4.8.5 Результаты тестирования

n=1, mp=4

```
 # mp =    4
 #   a=    0.000000E+00
 #   b=    0.100000E+01
 #   n=         1
 #   Exact
  0.50000000
  0.33333334
  0.25000000
  0.20000000
 # rectan0:
    Функция   Квадратурная сумма   Отн.погр
       x      0.500000E+00         0.00E+00  <--= RECTAN точна по 1 точке,
      x^2     0.250000E+00         0.25E+00       а для нелинейных функций
      x^3     0.125000E+00         0.50E+00       одноточечный режим
      x^4     0.625000E-01         0.69E+00       RECTAN непригоден
 # rectan1:
    Функция   Квадратурная сумма   Отн.погр
       x      0.500000E+00         0.00E+00
      x^2     0.250000E+00         0.25E+00
      x^3     0.125000E+00         0.50E+00
      x^4     0.625000E-01         0.69E+00
 #   trap0:
       x      0.500000E+00         0.00E+00  <--= TRAP точна по 2 точкам
      x^2     0.500000E+00         0.50E+00       для линейных, а для
      x^3     0.500000E+00         0.10E+01       полиномов более высоких
      x^4     0.500000E+00         0.15E+01       степеней её точность
 #   trap1:                                       вдвое хуже чем точность
    Функция   Квадратурная сумма   Отн.погр       RECTAN:  0.5 вместо 0.25
       x      0.500000E+00         0.00E+00                1.0 -"--"- 0.5
      x^2     0.500000E+00         0.50E+00                1.5 -"--"- 0.69
      x^3     0.500000E+00         0.10E+01
      x^4     0.500000E+00         0.15E+01
 #    sim0:
 # n=         2 must be even !!!
    Функция   Квадратурная сумма   Отн.погр
       x      0.500000E+00         0.00E+00  <--= SIM по трём точкам точно
      x^2     0.333333E+00         0.00E+00  <--= вычисляет квадратурные
      x^3     0.250000E+00         0.00E+00  <--= суммы для полиномов
      x^4     0.208333E+00         0.42E-01       степеней 0, 1, 2 и 3.
 #    sim1:
    Функция   Квадратурная сумма   Отн.погр    Для полиномов 4-ой степени
       x      0.500000E+00         0.00E+00    при дроблении промежутка
      x^2     0.333333E+00         0.00E+00    интегрирования на 2
      x^3     0.250000E+00         0.00E+00    подучастка точность расчёта
      x^4     0.208333E+00         0.42E-01    SIM низка.
```

n=1, mp=8

```
 # mp =    8
 #   a=      0.00000000000000E+00
 #   b=      0.10000000000000E+01
 #   n=         1
 #   Exact
  0.50000000000000000
  0.33333333333333331
  0.25000000000000000
  0.20000000000000001
 # rectan0:
    Функция   Квадратурная сумма   Отн.погр
       x        0.50000000000000E+00         0.00E+00 <--= Подтверждена
      x^2       0.25000000000000E+00         0.25E+00      правильность
      x^3       0.12500000000000E+00         0.50E+00      работы RECTAN
      x^4       0.62500000000000E-01         0.69E+00      при n=1
 # rectan1:
    Функция   Квадратурная сумма   Отн.погр
       x        0.50000000000000E+00         0.00E+00
      x^2       0.25000000000000E+00         0.25E+00
      x^3       0.12500000000000E+00         0.50E+00
      x^4       0.62500000000000E-01         0.69E+00
 #   trap0:
       x        0.50000000000000E+00         0.00E+00 <--= Подтверждена
      x^2       0.50000000000000E+00         0.50E+00      правильность
      x^3       0.50000000000000E+00         0.10E+01      работы TRAP
      x^4       0.50000000000000E+00         0.15E+01      при n=2
 #   trap1:
    Функция   Квадратурная сумма   Отн.погр
       x        0.50000000000000E+00         0.00E+00
      x^2       0.50000000000000E+00         0.50E+00
      x^3       0.50000000000000E+00         0.10E+01
      x^4       0.50000000000000E+00         0.15E+01
 #    sim0:
 # n=         2 must be even !!!
    Функция   Квадратурная сумма   Отн.погр
       x        0.50000000000000E+00         0.00E+00 <--= Подтверждена
      x^2       0.33333333333333E+00         0.00E+00 <--= правильность
      x^3       0.25000000000000E+00         0.00E+00 <--= работы SIM
      x^4       0.20833333333333E+00         0.42E-01      при n=2 (n1=3)
 #    sim1:
    Функция   Квадратурная сумма   Отн.погр
       x        0.50000000000000E+00         0.00E+00
      x^2       0.33333333333333E+00         0.00E+00
      x^3       0.25000000000000E+00         0.00E+00
      x^4       0.20833333333333E+00         0.42E-01
```

n=1, mp=10

```
 # mp =   10
 #   a=     0.00000000000000000E+00
 #   b=     0.10000000000000000E+01
 #   n=         1
 #   Exact
  0.50000000000000000000
  0.33333333333333333334
  0.25000000000000000000
  0.20000000000000000000
 # rectan0:
    Функция   Квадратурная сумма   Отн.погр
       x       0.50000000000000000E+00         0.00E+00
      x^2      0.25000000000000000E+00         0.25E+00
      x^3      0.12500000000000000E+00         0.50E+00
      x^4      0.62500000000000000E-01         0.69E+00
 # rectan1:
    Функция   Квадратурная сумма   Отн.погр
       x       0.50000000000000000E+00         0.00E+00
      x^2      0.25000000000000000E+00         0.25E+00
      x^3      0.12500000000000000E+00         0.50E+00
      x^4      0.62500000000000000E-01         0.69E+00
 #   trap0:
       x       0.50000000000000000E+00         0.00E+00
      x^2      0.50000000000000000E+00         0.50E+00
      x^3      0.50000000000000000E+00         0.10E+01
      x^4      0.50000000000000000E+00         0.15E+01
 #   trap1:
    Функция   Квадратурная сумма   Отн.погр
       x       0.50000000000000000E+00         0.00E+00
      x^2      0.50000000000000000E+00         0.50E+00
      x^3      0.50000000000000000E+00         0.10E+01
      x^4      0.50000000000000000E+00         0.15E+01
 #    sim0:
 # n=         2 must be even !!!
    Функция   Квадратурная сумма   Отн.погр
       x       0.50000000000000000E+00         0.00E+00
      x^2      0.33333333333333333E+00         0.00E+00
      x^3      0.25000000000000000E+00         0.00E+00
      x^4      0.20833333333333333E+00         0.42E-01
 #    sim1:
    Функция   Квадратурная сумма   Отн.погр
       x       0.50000000000000000E+00         0.00E+00
      x^2      0.33333333333333333E+00         0.00E+00
      x^3      0.25000000000000000E+00         0.00E+00
      x^4      0.20833333333333333E+00         0.42E-01
```

n=1, mp=16

```
 # mp =   16
 #   a=   0.00000000000000000000000000000000000E+00
 #   b=   0.10000000000000000000000000000000000E+01
 #   n=         1
 #   Exact
  0.50000000000000000000000000000000000
  0.33333333333333333333333333333333317
  0.25000000000000000000000000000000000
  0.20000000000000000000000000000000010
 # rectan0:
    Функция   Квадратурная сумма   Отн.погр
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.25000000000000000000000000000000000E+00         0.25E+00
      x^3    0.12500000000000000000000000000000000E+00         0.50E+00
      x^4    0.62500000000000000000000000000000000E-01         0.69E+00
 # rectan1:
    Функция   Квадратурная сумма   Отн.погр
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.25000000000000000000000000000000000E+00         0.25E+00
      x^3    0.12500000000000000000000000000000000E+00         0.50E+00
      x^4    0.62500000000000000000000000000000000E-01         0.69E+00
 #   trap0:
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.50000000000000000000000000000000000E+00         0.50E+00
      x^3    0.50000000000000000000000000000000000E+00         0.10E+01
      x^4    0.50000000000000000000000000000000000E+00         0.15E+01
 #   trap1:
    Функция   Квадратурная сумма   Отн.погр
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.50000000000000000000000000000000000E+00         0.50E+00
      x^3    0.50000000000000000000000000000000000E+00         0.10E+01
      x^4    0.50000000000000000000000000000000000E+00         0.15E+01
 #    sim0:
 # n=         2 must be even !!!
    Функция   Квадратурная сумма   Отн.погр
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.33333333333333333333333333333333332E+00         0.00E+00
      x^3    0.25000000000000000000000000000000000E+00         0.00E+00
      x^4    0.20833333333333333333333333333333334E+00         0.42E-01
 #    sim1:
    Функция   Квадратурная сумма   Отн.погр
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.33333333333333333333333333333333332E+00         0.00E+00
      x^3    0.25000000000000000000000000000000000E+00         0.00E+00
      x^4    0.20833333333333333333333333333333334E+00         0.42E-01
```

n=2, mp=16

```
 # mp =   16
 #   a=   0.00000000000000000000000000000000000E+00
 #   b=   0.10000000000000000000000000000000000E+01
 #   n=         2
 #   Exact
  0.50000000000000000000000000000000000
  0.33333333333333333333333333333333317
  0.25000000000000000000000000000000000
  0.20000000000000000000000000000000010
 # rectan0:
    Функция   Квадратурная сумма   Отн.погр
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.31250000000000000000000000000000000E+00         0.62E-01
      x^3    0.21875000000000000000000000000000000E+00         0.12E+00
      x^4    0.16015625000000000000000000000000000E+00         0.20E+00
 # rectan1:
    Функция   Квадратурная сумма   Отн.погр
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.31250000000000000000000000000000000E+00         0.62E-01
      x^3    0.21875000000000000000000000000000000E+00         0.12E+00
      x^4    0.16015625000000000000000000000000000E+00         0.20E+00
 #   trap0:
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.37500000000000000000000000000000000E+00         0.13E+00
      x^3    0.31250000000000000000000000000000000E+00         0.25E+00
      x^4    0.28125000000000000000000000000000000E+00         0.41E+00
 #   trap1:
    Функция   Квадратурная сумма   Отн.погр
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.37500000000000000000000000000000000E+00         0.13E+00
      x^3    0.31250000000000000000000000000000000E+00         0.25E+00
      x^4    0.28125000000000000000000000000000000E+00         0.41E+00
 #    sim0:
    Функция   Квадратурная сумма   Отн.погр
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.33333333333333333333333333333333332E+00         0.00E+00
      x^3    0.25000000000000000000000000000000000E+00         0.00E+00
      x^4    0.20833333333333333333333333333333334E+00         0.42E-01
 #    sim1:
    Функция   Квадратурная сумма   Отн.погр
       x     0.50000000000000000000000000000000000E+00         0.00E+00
      x^2    0.33333333333333333333333333333333332E+00         0.00E+00
      x^3    0.25000000000000000000000000000000000E+00         0.00E+00
      x^4    0.20833333333333333333333333333333334E+00         0.42E-01
```

n=3, mp=16

```
# mp =   16
#   a=   0.00000000000000000000000000000000000E+00
#   b=   0.10000000000000000000000000000000000E+01
#   n=         3
#   Exact
 0.50000000000000000000000000000000000
 0.33333333333333333333333333333333317
 0.25000000000000000000000000000000000
 0.20000000000000000000000000000000010
# rectan0:
   Функция   Квадратурная сумма   Отн.погр
      x     0.50000000000000000000000000000000000E+00         0.00E+00
     x^2    0.32407407407407407407407407407407404E+00         0.28E-01
     x^3    0.23611111111111111111111111111111106E+00         0.56E-01
     x^4    0.18184156378600823045267489711934152E+00         0.91E-01
# rectan1:
   Функция   Квадратурная сумма   Отн.погр
      x     0.50000000000000000000000000000000000E+00         0.00E+00
     x^2    0.32407407407407407407407407407407404E+00         0.28E-01
     x^3    0.23611111111111111111111111111111106E+00         0.56E-01
     x^4    0.18184156378600823045267489711934152E+00         0.91E-01
#   trap0:
      x     0.50000000000000000000000000000000000E+00         0.00E+00
     x^2    0.35185185185185185185185185185185183E+00         0.56E-01
     x^3    0.27777777777777777777777777777777774E+00         0.11E+00
     x^4    0.23662551440329218106995884773662549E+00         0.18E+00
#   trap1:
   Функция   Квадратурная сумма   Отн.погр
      x     0.50000000000000000000000000000000000E+00         0.00E+00
     x^2    0.35185185185185185185185185185185183E+00         0.56E-01
     x^3    0.27777777777777777777777777777777774E+00         0.11E+00
     x^4    0.23662551440329218106995884773662549E+00         0.18E+00
#    sim0:
# n=         4 must be even !!!
   Функция   Квадратурная сумма   Отн.погр
      x     0.50000000000000000000000000000000000E+00         0.00E+00
     x^2    0.33333333333333333333333333333333332E+00         0.00E+00
     x^3    0.25000000000000000000000000000000000E+00         0.00E+00
     x^4    0.20052083333333333333333333333333334E+00         0.26E-02
#    sim1:
   Функция   Квадратурная сумма   Отн.погр
      x     0.50000000000000000000000000000000000E+00         0.00E+00
     x^2    0.33333333333333333333333333333333332E+00         0.00E+00
     x^3    0.25000000000000000000000000000000000E+00         0.00E+00
     x^4    0.20052083333333333333333333333333334E+00         0.26E-02
```

n=128, mp=16

```
# mp =   16
#   a=   0.00000000000000000000000000000000E+00
#   b=   0.10000000000000000000000000000000E+01
#   n=       128
#   Exact
 0.50000000000000000000000000000000
 0.33333333333333333333333333333317
 0.25000000000000000000000000000000
 0.20000000000000000000000000000010
# rectan0:
   Функция   Квадратурная сумма   Отн.погр
      x     0.50000000000000000000000000000000E+00         0.00E+00
     x^2    0.33332824707031250000000000000000E+00         0.15E-04
     x^3    0.24999237060546875000000000000000E+00         0.31E-04
     x^4    0.19998982758261263370513916015625000E+00         0.51E-04
# rectan1:
   Функция   Квадратурная сумма   Отн.погр
      x     0.50000000000000000000000000000000E+00         0.00E+00
     x^2    0.33332824707031250000000000000000E+00         0.15E-04
     x^3    0.24999237060546875000000000000000E+00         0.31E-04
     x^4    0.19998982758261263370513916015625000E+00         0.51E-04
#   trap0:
      x     0.50000000000000000000000000000000E+00         0.00E+00
     x^2    0.33334350585937500000000000000000E+00         0.31E-04
     x^3    0.25001525878906250000000000000000E+00         0.61E-04
     x^4    0.20002034492790699005126953125000000E+00         0.10E-03
#   trap1:
   Функция   Квадратурная сумма   Отн.погр
      x     0.50000000000000000000000000000000E+00         0.00E+00
     x^2    0.33334350585937500000000000000000E+00         0.31E-04
     x^3    0.25001525878906250000000000000000E+00         0.61E-04
     x^4    0.20002034492790699005126953125000000E+00         0.10E-03
#    sim0:
   Функция   Квадратурная сумма   Отн.погр
      x     0.50000000000000000000000000000000E+00         0.00E+00
     x^2    0.33333333333333333333333333333332E+00         0.00E+00
     x^3    0.25000000000000000000000000000000E+00         0.00E+00
     x^4    0.20000000496705373128255208333333334E+00         0.25E-08
#    sim1:
   Функция   Квадратурная сумма   Отн.погр
      x     0.50000000000000000000000000000000E+00         0.00E+00
     x^2    0.33333333333333333333333333333332E+00         0.00E+00
     x^3    0.25000000000000000000000000000000E+00         0.00E+00
     x^4    0.20000000496705373128255208333333334E+00         0.25E-08
```

n=512, mp=16

```
# mp =   16
#   a=   0.0000000000000000000000000000000000E+00
#   b=   0.1000000000000000000000000000000000E+01
#   n=       512
#   Exact
 0.5000000000000000000000000000000000000
 0.3333333333333333333333333333333333317
 0.2500000000000000000000000000000000000
 0.2000000000000000000000000000000000010
# rectan0:
   Функция   Квадратурная сумма   Отн.погр
      x     0.5000000000000000000000000000000000E+00         0.00E+00
     x^2    0.3333330154418945312500000000000000E+00         0.95E-06
     x^3    0.2499995231628417968750000000000000E+00         0.19E-05
     x^4    0.1999993642175468266941606998443603 5E+00       0.32E-05
# rectan1:
   Функция   Квадратурная сумма   Отн.погр
      x     0.5000000000000000000000000000000000E+00         0.00E+00
     x^2    0.3333330154418945312500000000000000E+00         0.95E-06
     x^3    0.2499995231628417968750000000000000E+00         0.19E-05
     x^4    0.1999993642175468266941606998443603 5E+00       0.32E-05
#   trap0:
      x     0.5000000000000000000000000000000000E+00         0.00E+00
     x^2    0.3333339691162109375000000000000000E+00         0.19E-05
     x^3    0.2500009536743164062500000000000000E+00         0.38E-05
     x^4    0.2000012715652701444923877716064453 1E+00       0.64E-05
#   trap1:
   Функция   Квадратурная сумма   Отн.погр
      x     0.5000000000000000000000000000000000E+00         0.00E+00
     x^2    0.3333339691162109375000000000000000E+00         0.19E-05
     x^3    0.2500009536743164062500000000000000E+00         0.38E-05
     x^4    0.2000012715652701444923877716064453 1E+00       0.64E-05
#    sim0:
   Функция   Квадратурная сумма   Отн.погр
      x     0.5000000000000000000000000000000000E+00         0.00E+00
     x^2    0.3333333333333333333333333333333332E+00         0.00E+00
     x^3    0.2500000000000000000000000000000000E+00         0.00E+00
     x^4    0.2000000000019402553637822469075520 9E+00       0.97E-11
#    sim1:
   Функция   Квадратурная сумма   Отн.погр
      x     0.5000000000000000000000000000000000E+00         0.00E+00
     x^2    0.3333333333333333333333333333333332E+00         0.00E+00
     x^3    0.2500000000000000000000000000000000E+00         0.00E+00
     x^4    0.2000000000019402553637822469075520 9E+00       0.97E-11
```

4.8.6 Поучительный пример A (начало)

Рассмотрим упрощённый вариант нашей программы, корректно исключив из неё всё, кроме одного единственного вызова **rectan1**.

```
program ex4_8a; use my_prec; use quadra
implicit none
real(mp), allocatable :: y(:)
real(mp) a, b, h, r
integer i, n
read(*,'(e10.3)') a, b
read(*,'(i10)') n
write(*,*)  ' # mp =', mp
write(*,*)  ' #   a=', a
write(*,*)  ' #   b=', b
write(*,*)  ' #   n=', n
allocate(y(n))
h=(b-a)/n
do i=1,n
  y(i)=a+(i-0.5_mp)*h
enddo
r=rectan1(y,a,b); write(*,*)  ' r=',r
end program ex4_8a
```

Результат её работы:

```
# mp =          16
#   a=   0.00000000000000000000000000000000000
#   b=   1.00000000000000000000000000000000000
#   n=         555
r=  0.50000000000000000000000000000000193
```

Ничего неожиданного. Всё верно. Теперь предположим, что главная программа посложнее. Причём усложнение состоит в том, что наряду с вызовом **rectan1** в теле главной программы потребовался вызов **rectan1** и внутри какой-то другой подпрограммы, вызываемой опять-таки из главной (см. пример **B**).

4.8.7 Поучительный пример B (продолжение)

Назовём эту подпрограмму, например, **sub(y,a,b)**. Её исходный текст

```
subroutine sub(y,a,b)
use my_prec
use quadra
implicit none
real(mp) y(:), a, b, r
r=rectan1(y,a,b)
write(*,*) ' sub:  r=',r
end subroutine sub
```

Пусть исходный текст главной программы теперь такой:

```
program ex4_8b; use my_prec; use quadra; implicit none
real(mp), allocatable :: y(:)
real(mp) a, b, h, r; integer i, n
read(*,'(e10.3)') a, b;  read(*,'(i10)') n
write(*,*)  ' # mp =', mp
write(*,*)  ' #   a=', a; write(*,*)  ' #   b=', b
write(*,*)  ' #   n=', n
allocate(y(n))
h=(b-a)/n; do i=1,n; y(i)=a+(i-0.5_mp)*h; enddo
r=rectan1(y,a,b)
write(*,*)  ' main:  r=',r
call sub(y,a,b)  ! <--= Единственное отличие от ex4_8a
end program ex4_8b
```

Результат её работы при **mp=4, 8, 10 — ошибка сегментирования** (т.е. ошибка адресации). Причина — в главной программе не указан интерфейс **sub**.

Старый ФОРТРАН *не понимал* описания **y(:)**. Современный ФОРТРАН *понимает*, но главная программа *не знает*, как описан внутри **sub** аргумент **y**. Без явного указания интерфейса, последний относительно **y** *понимается по умолчанию* (как в старом ФОРТРАНе). Для исправления ситуации нужно главной программе явно указать интерфейс **sub** (см. пример **C**).

При **mp=16** можно получить результат, маскирующий указанную причину:

```
# mp =          16
#   a=   0.00000000000000000000000000000000000
#   b=   1.00000000000000000000000000000000000
#   n=         555
main:  r=  0.50000000000000000000000000000000193
sub:  r=                                      NaN
```

4.8.8 Поучительный пример C (продолжение)

Исходный текст подпрограммы **sub** тот же, что и в примере B, но исходный текст главной программы теперь содержит описание интефейса:

```
program ex4_8c; use my_prec; use quadra; implicit none
interface
  subroutine sub(y,a,b)    !      / Единственное
  real(mp) y(:), a, b      ! <--=      отличие
  end subroutine sub       !      \      от ex4_8b
end interface
real(mp), allocatable :: y(:)
real(mp) a, b, h, r;     integer i, n
read(*,'(e10.3)') a, b;  read(*,'(i10)') n
write(*,*)  ' # mp =', mp
write(*,*)  ' #   a=', a; write(*,*)  ' #   b=', b
write(*,*)  ' #   n=', n
allocate(y(n))
h=(b-a)/n
do i=1,n
  y(i)=a+(i-0.5_mp)*h
enddo
r=rectan1(y,a,b)
write(*,*)  ' main:  r=',r
call sub(y,a,b)
end program ex4_8c
```

Однако, ещё на этапе компиляции получим:

```
main.f:5.13:
             real(mp) y(:), a, b
                   1
ошибка: Parameter 'mp' at (1) has not been declared or
        is a variable, which does not reduce to a constant expression
main.f:24.15:
              call sub(y,a,b)
                       1
ошибка: Type mismatch in argument 'y' at (1);
        passed REAL(10) to REAL(4)
make: *** [main.o] Ошибка 1
```

Вторая ошибка наведена первой, так как описанию интерфейса **sub** не известно имя **mp**, несмотря на то, что оно известно главной программе. Укажем это интерфейсу (см. пример **D**).

4.8.9 Поучительный пример D (продолжение)

Исходный текст главной программы с подправленным интерфейсом:

```
program ex4_8d; use my_prec; use quadra; implicit none
interface
  subroutine sub(y,a,b)
  use my_prec           ! <--=  Единственное
  use quadra            ! <--=               отличие от ex4_8c
  real(mp) y(:), a, b
  end subroutine sub
end interface
real(mp), allocatable :: y(:)
real(mp) a, b, h, r
integer i, n
read(*,'(e10.3)') a, b
read(*,'(i10)') n
write(*,*)  ' # mp =', mp
write(*,*)  ' #   a=', a
write(*,*)  ' #   b=', b
write(*,*)  ' #   n=', n
allocate(y(n))
h=(b-a)/n
y=(/(a+(i-0.5_mp)*h,i=1,n)/) !<--= Заполнить массив Y можно и так !
r=rectan1(y,a,b)
write(*,*)  ' main:  r=',r
call sub(y,a,b)
end program ex4_8d
```

Результат её работы верен:

```
# mp =          16
#   a=   0.00000000000000000000000000000000000
#   b=   1.00000000000000000000000000000000000
#   n=         555
main:  r=  0.50000000000000000000000000000000193
sub:  r=  0.50000000000000000000000000000000193
```

Однако, заметим, что, если бы

1) описание **sub**, было включено в модуль **quadra**, то описывать интерфейс **sub** в главной программе не пришлось, поскольку он уже обеспечен программной единицей **module quadra**.

2) описание **rectan1** было помещено не в модуль **quadra**, а в обычный файл, так что имя **rectan1** воспринималось компилятором как обычное имя внешней функции, то потребовались дополнительные указания интерфейса, причём в двух местах (см. пример **E**).

4.8.10 Поучительный пример E (продолжение)

Продемонстрируем пункт 2) предыдущего пункта. Пусть в нашем проекте только один модуль **my_prec**, а подпрограмма **sub** и функция **rectan1** — обычные внешние пользовательские процедуры:

```
                                    ! RECTAN1 находит квадратурную
function rectan1(y,a,b) result(s)   ! сумму формулы средних прямо-
real(mp) y(:), a, b, s, h           ! угольников по [a,b] для фун-
h=(b-a)/size(y)                     ! кции табулированной в y(:)
s=h*sum(y)                          ! (перенимающим форму векторе).
end function rectan1

subroutine sub(y,a,b)
use my_prec
implicit none
real(mp) y(:), a, b, r
r=rectan1(y,a,b)
write(*,*) ' sub:  r=',r
end subroutine sub

program ex4_8_E; use my_prec   ! Модуль quadra не подключён
implicit none
interface
  subroutine sub(y,a,b)
  use my_prec
  real(mp) y(:), a, b
  end subroutine sub
end interface
real(mp), allocatable :: y(:)
real(mp) a, b, h, r
integer i, n
read(*,'(e10.3)') a, b
read(*,'(i10)') n
write(*,*)  ' # mp =', mp
write(*,*)  ' #   a=', a
write(*,*)  ' #   b=', b
write(*,*)  ' #   n=', n
allocate(y(n))
h=(b-a)/n
y=(/(a+(i-0.5_mp)*h,i=1,n)/)
r=rectan1(y,a,b)
write(*,*)  ' main:  r=',r
call sub(y,a,b)
end program ex4_8_E
```

Заметим, что здесь функция **rectan** присутствует ровно в том виде, в каком она формально была в модуле **quadra**.

Попытка компиляции такого проекта:

```
$ gfortran -c my_prec.f rectan1.f sub.f main.f
rectan1.f:3.11:

      real(mp) y(:), a, b, s, h
           1
ошибка: Parameter 'mp' at (1) has not been declared or is a variable,
                      which does not reduce to a constant expression
rectan1.f:4.19:

      h=(b-a)/size(y)
                   1
ошибка: 'array' argument of 'size' intrinsic at (1) must be an array
rectan1.f:5.14:

      s=h*sum(y)
              1
ошибка: 'array' argument of 'sum' intrinsic at (1) must be an array
sub.f:5.8:

      r=rectan1(y,a,b)
        1
ошибка: Function 'rectan1' at (1) has no IMPLICIT type
main.f:23.8:

      r=rectan1(y,a,b)
        1
ошибка: Function 'rectan1' at (1) has no IMPLICIT type
```

Сразу целый фейеверк ошибок.

1. Процедуре **rectan1** не известно имя **mp**.

   Когда **rectan1** была в модуле **quadra** имя **mp** ей было известно, ввиду наличия в модуле **quadra** описания **use my_prec**. Наличие же последнего описания просто в главной программе делает доступным имя **mp** только внутри главной программы, но не во внешних программных единицах, которые она вызывает. В этом принципиальное отличие **module**.

2. Остальные ошибки — следствие неизвестности **mp** внутри **rectan1**.

Попытаемся исправить ситуацию, подключив модуль **my_prec** к функции **rectan1** (см.пример **F**).

4.8.11 Поучительный пример F (продолжение)

Подключим модуль **my_prec** к **rectan1**:

```
      function rectan1(y,a,b) result(s)
      use my_prec                          ! <--= ДОБАВИЛИ подключение
      real(mp) y(:), a, b, s, h
      h=(b-a)/size(y)
      s=h*sum(y)
      end function rectan1
```

После

```
$ gfortran -c my_prec.f rectan1.f sub.f main.f
sub.f:5.8:

      r=rectan1(y,a,b)
        1
ошибка: Function 'rectan1' at (1) has no IMPLICIT type
main.f:23.8:

      r=rectan1(y,a,b)
        1
ошибка: Function 'rectan1' at (1) has no IMPLICIT type
```

- Законная реакция компилятора на безалаберное изъятие процедуры из модуля. Изъять-то — изъяли, но забыли при этом описать и в **main.f**, и в **sub.f** тип значения возвращаемого функцией **rectan1**. Когда **rectan1** была в модуле, передачу типа обеспечивал именно модуль. Теперь же (при **implicit none**) об этом нужно заботиться особо (см. пример **G**).

4.8.12 Поучительный пример G (продолжение)

Внесём исправления требуемые в предыдущем пункте:

```
      subroutine sub(y,a,b)
      use my_prec
      implicit none
      real(mp) y(:), a, b, r, rectan1
      r=rectan1(y,a,b)
      write(*,*) ' sub:  r=',r
      end subroutine sub

      program ex4_8_G; use my_prec; implicit none
      interface
        subroutine sub(y,a,b)
        use my_prec
        real(mp) y(:), a, b
        end subroutine sub
      end interface
      real(mp), allocatable :: y(:)
      real(mp) a, b, h, r, rectan1
      integer i, n
      read(*,'(e10.3)') a, b;   read(*,'(i10)') n
      write(*,*)  ' # mp =', mp
      write(*,*)  ' #   a=', a; write(*,*)  ' #   b=', b
      write(*,*)  ' #   n=', n
      allocate(y(n))
      h=(b-a)/n
      y=(/(a+(i-0.5_mp)*h,i=1,n)/)
      r=rectan1(y,a,b)
!      write(*,*)  ' main:  r=',r
!      call sub(y,a,b)
      end program ex4_8_G
```

- Компиляция пройдёт удачно. Но при запуске — вновь **ошибка сегментирования**.
- Закомментируем в главной программе вызовы **rectan1**, **sub** и вывод **r**. При пропуске ошибка сегментирования исчезнет.
- Теперь расскоментируем вызов **rectan1**. Опять ошибка сегментирования. Причина: главная программа сама по себе вызывает **rectan1**. И, поскольку первый формальный аргумент функции **rectan1** перенимает форму соответствующего фактического аргумента, то главной программе необходимо сообщить об этом особо. Короче, помимо интерфейса **sub**, главной программе необходимо указать и интерфейс **rectan1** (см. пример **H**).

4.8.13 Поучительный пример H (продолжение)

После добавления интерфейса **rectan1** в главную программу

```
      program ex4_8_H; use my_prec; implicit none
      interface
        subroutine sub(y,a,b)
        use my_prec
        real(mp) y(:), a, b
        end subroutine sub
        function rectan1(y,a,b) result (s)   ! / Добавили
        use my_prec                          !<      описание
        real(mp) y(:), a, b, s               ! \        интерфейса rectan1
        end function rectan1
      end interface
      real(mp), allocatable :: y(:)
      real(mp) a, b, h, r, rectan1;  integer i, n
      read(*,'(e10.3)') a, b;   read(*,'(i10)') n
      write(*,*)  ' # mp =', mp
      write(*,*)  ' #   a=', a; write(*,*)  ' #   b=', b
      write(*,*)  ' #   n=', n
      allocate(y(n))
      h=(b-a)/n;    y=(/(a+(i-0.5_mp)*h,i=1,n)/); r=rectan1(y,a,b)
      write(*,*)  ' main:  r=',r
!      call sub(y,a,b)
      end program ex4_8_H
```

и компиляции получим сообщение о том, что теперь в главной программе дважды сообщается о типе значения, возвращаемого **rectan1**

```
main.f:13.34:
      real(mp) a, b, h, r, rectan1
                                1 ошибка:
  Symbol 'rectan1' at (1) already has basic type of REAL  main.f:15.25:
  ...   ...   ...   ...   ...   ...   ...   ...
```

первый раз при описании интерфейса (который только что добавили), а второй — в тридцать четвертой позиции тринадцатой строки. Конечно, можно и возмутиться по поводу такого высказывания:

То ему (компилятору) опиши **rectan1**;
то, когда описали в примере **G**, так потребовал интерфейс,
а, когда и интерфейс подключили, то заругался, что rectan1 уже был описан.

На самом деле, компилятор прав: после указания полного интерфейса функции уже не нужно *старорежимное ущербное* описание типа значения, возвращаемого ею (см. пример **I**).

4.8.14 Поучительный пример I (продолжение)

После добавления интерфейса **rectan1** в главную программу

```
      program ex4_8_I; use my_prec; implicit none
      interface
        subroutine sub(y,a,b)
        use my_prec
        real(mp) y(:), a, b
        end subroutine sub
        function rectan1(y,a,b) result (s)
        use my_prec
        real(mp) y(:), a, b, s
        end function rectan1
      end interface
      real(mp), allocatable :: y(:)
      real(mp) a, b, h, r !<--= Убран тип значения возвращаемого rectan1.
      integer i, n
      read(*,'(e10.3)') a, b;   read(*,'(i10)') n
      write(*,*)  ' # mp =', mp
      write(*,*)  ' #   a=', a; write(*,*)  ' #   b=', b
      write(*,*)  ' #   n=', n
      allocate(y(n))
      h=(b-a)/n;    y=(/(a+(i-0.5_mp)*h,i=1,n)/); r=rectan1(y,a,b)
      write(*,*)  ' main:  r=',r
!      call sub(y,a,b)
      end program ex4_8_I
```

Компиляция проходит, и, пока закомментирована строка с вызовом подпрограммы **sub** функция **rectan1** работает и программа выводит верный результат.

Однако, как только раскомментируем строку с вызовом **sub**, снова — **ошибка сегментирования**, т.е. что-то ещё неладно в описании **sub**.

Дело в том, что, когда **rectan1** была в модуле **quadra** и последний подключался к **sub**, то **sub** знала интерфейс **rectan1** через посредство модуля. Теперь же, когда и **sub**, и **rectan1** — простые внешние процедуры, компиляция которых происхолит абсолютно независимо друг от друга, подпрограмма **sub** не знает интерфейса **rectan1**.

Короче, интерфейс **rectan1** нужно описать не только в главной программе, но и в подпрограмме **sub** (см. пример **J**).

4.8.15 Поучительный пример J (окончание)

После добавления интерфейса **rectan1** в **sub**

```
program ex4_8_I; use my_prec; implicit none
interface
  subroutine sub(y,a,b)
  use my_prec
  real(mp) y(:), a, b
  end subroutine sub
  function rectan1(y,a,b) result (s)   ! / Добавили
  use my_prec                          !<      описание
  real(mp) y(:), a, b, s               ! \        интерфейса rectan1
  end function rectan1
end interface
real(mp), allocatable :: y(:)
real(mp) a, b, h, r           !<-- Убрали старорежимное описание
integer i, n
read(*,'(e10.3)') a, b
read(*,'(i10)') n
write(*,*)  ' # mp =', mp
write(*,*)  ' #   a=', a
write(*,*)  ' #   b=', b
write(*,*)  ' #   n=', n
allocate(y(n))
h=(b-a)/n
y=(/(a+(i-0.5_mp)*h,i=1,n)/)
r=rectan1(y,a,b)
write(*,*)  ' main:  r=',r
call sub(y,a,b)
end program ex4_8_I
```

И вот теперь получили желаемый результат:

```
# mp =          16
#   a=   0.00000000000000000000000000000000000
#   b=   1.00000000000000000000000000000000000
#   n=         555
main:  r=  0.50000000000000000000000000000000193
sub:  r=  0.50000000000000000000000000000000193
```

4.8.16 Выводы

1. В ФОРТРАНЕ приём формальным аргументом процедуры, который является массивом, значения фактического аргумента можно осуществить разными способами.

2. Можно в качестве одного из аргументов указать имя переменной, хранящей число используемых элементов массива. Тогда описание массива (формального аргумента) может иметь вид, например, **real y(n)**. При этом важно, чтобы значение **n** не превосходило заявленный размер фактического аргумента. Подобный способ приемлем, когда интерфейс задан неявно, т.е. формой вызова процедуры. Древние версии ФОРТРАНа только такой способ и допускали.

3. Современный ФОРТРАН позволяет явно описать интерфейс процедуры в вызывающей её программной единице (либо через оператор **interface**, либо через подсоединение модуля, содержащего процедуру). Без этого способа невозможно, например, указать вызывающей программе, что формальный аргумент (массив) перенимает форму фактического, т.е. объявлен оператором **real y(:)**.

4. Достоинство последнего способа: не нужно явно передавать количество элементов массива, что особенно актуально при использовании размещаемых массивов. Поэтому вопрос о числе элементов массива, передаваемого в качестве фактического аргумента уже не так актуален как раньше, поскольку оператор **allocate** выделит под массив ровно столько элементов сколько нужно.

5. Описание **real y(:)**

   (a) явно не зависит от числа элементов (т.е. более универсально);

   (b) встроенная ФОРТРАН-функция **size** позволяет внутри процедуры найти число элементов массива, так что включать в заголовок процедуры дополнительную переменную уже не имеет смысла; а встроенная ФОРТРАН-функция **shape** (если нужно) позволит определить и форму массива и длину каждого его измерения.

6. Поэтому, если массив — размещаемый, то соответствующий формальный аргумент процедуры, выгодно оформить перенимающим форму.

7. В последнем случае важно помнить о необходимости описания явного интерфейса в каждой программной единице, вызывающей эту процедуру.

8. Наиболее бесхлопотно явный интерфейс подключается автоматически при использовании программной единицы **module**, в которой описана процедура.

9. Если же процедура является обычной внешней процедурой ФОРТРАНа (т.е. не описана в модуле), то в каждой из вызывающих её программных единицах придётся (тем или иным способом) использовать оператор **interface**. Возможные варианты либо через старорежимный оператор **include**, либо через модуль, хранящий только интерфейс. Правда, в последнем случае проще, пожалуй, поместить в модуль саму процедуру.

5 Операции ФОРТРАНа над массивами (второй семестр)

5.1 Задача 1

5.1.1 Условие

Разработать функции **sxy**, **sxz**, **syz**, получающие соответственно в качестве результата матрицы **k**-го слоя параллельного грани трёхмерного динамического массива **a(nx,ny,nz)**. В качестве тестового примера ввести **nx=3**, **ny=4**, **nz=5** и **a=**

1 2 3 4 5 6 7 8 9 10 11 12
10 20 30 40 50 60 70 80 90 100 110 120
100 200 300 400 500 600 700 800 900 1000 1100 1200
1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 11000 12000
10000 20000 30000 40000 50000 60000 70000 80000 90000 100000
11000 12000

5.1.2 Пояснение к условию

Условие требует, чтобы каждая из функций возвращала через своё имя именно массив. В старом ФОРТРАНе такой возможности не было. Поэтому приходилось оформлять алгоритм подпрограммой, которая в списке формальных аргументов помимо входной информации должна была иметь и аргумент для результата. Интерфейс такой подпрограммы мог бы выглядеть так:

```
subroutine sxy(k,nx,ny,nz,a, res)
integer k. nx,ny,nz
integer res(nx,ny), a(nx,ny,nz)
```

По условию задачи интерфейс, например, функции, sxy должен иметь вид:

```
                                      !         ИЛИ, что одно и то же:
function sxy(k,nx, ny,nz,a) result(res) ! function sxy(k,nx,ny,nz,a)
integer k, nx, ny, nz                 ! integer k, nx, ny, nz
integer res(nx,ny), a(nx,ny,nz)       ! integer sxy(nx,ny), a(nx,ny,nz)
```

Однако, её интерфейс может выглядеть существенно проще и надёжнее:

```
function sxy(k,a) result(res)       ! (если формальный аргумент-массив
 integer a(:,:,:)                   !  описан перенимающим форму
 integer res(size(a,1),size(a,2))   !          фактического аргумента)
```

Два аргумента в заголовке процедуры значительно уменьшают возможность опечатки при вызове процедуры по сравнению со случаем, когда количество аргументов равно пяти. Приведём два варианта тестирующей программы. В первом оформим процедуры обычными внешними, поместив их исходные тексты в простой текстовый файл **sxyz.f90**; во втором — поместим процедуры в модуль **sxyzm.f90**, причём в последнем используем их интерфейс лишь с двумя формальными аргументами.

5.1.3 Содержимое файла input (исходные данные)

```
3
4
5
1   2   3
4   5   6
7   8   9
10    11   12
10    20   30
40    50   60
70    80   90
100  110  120
 100   200   300
 400   500   600
 700   800   900
1000 1100 1200
 1000    2000   3000
 4000    5000   6000
 7000    8000   9000
10000   11000 12000
10000    20000   30000
40000    50000   60000
70000    80000   90000
100000 110000 120000
```

5.1.4 Вариант 1 (внешние процедуры)

Исходный текст процедур (файл sxyz.f95)

```
function sxy(k,nx,ny,nz,a); implicit none; integer k, nx, ny, nz
integer sxy(nx,ny), a(nx,ny,nz)
sxy=a(:,:,k)
end
function sxz(k,nx,ny,nz,a); implicit none; integer k, nx, ny, nz
integer sxz(nx,nz), a(nx,ny,nz)
sxz=a(:,k,:)
end
function syz(k,nx,ny,nz,a); implicit none; integer k, nx, ny, nz
integer syz(ny,nz), a(nx,ny,nz)
syz=a(k,:,:)
end
subroutine wrt_page(a); implicit none; integer a(:,:,:), sh(3)
integer i, j, k, nx, ny, nz
sh=shape(a); nx=sh(1); ny=sh(2); nz=sh(3); write(*,*) ' wrt_page(a):'
do k=1,nz
  write(*,'(4i7)') ((a(i,j,k),j=1,ny),i=1,nx); write(*,*)
enddo
end subroutine wrt_page
```

Исходный текст главной программы test1

```
program test1; implicit none
interface
 function sxy(k,nx,ny,nz,a); integer k, nx,ny,nz;
                             integer sxy(nx,ny), a(nx,ny,nz)
 end function sxy;
 function sxz(k,nx,ny,nz,a); integer k, nx,ny,nz;
                             integer sxz(nx,nz), a(nx,ny,nz)
 end function sxz;
 function syz(k,nx,ny,nz,a); integer k, nx,ny,nz;
                             integer syz(ny,nz), a(nx,ny,nz)
 end function syz;
 subroutine wrt_page(a); integer a(:,:,:); end subroutine
end interface
integer,allocatable :: a(:,:,:)              ! Исходный трёхмерный массив.
integer,allocatable :: axy(:,:)              ! Его вертикальный слой (фас);
integer,allocatable :: axz(:,:)              ! верт. слой || бок.грани;
integer,allocatable :: ayz(:,:)              ! горизонт. слой || основанию.
integer nx, ny, nz, k, ier
read (*,*) nx, ny, nz                        ! Ввод числа строк, столбцов и
write(*,*) ' nx=', nx, ' ny=', ny,' nz=', nz  ! листов массива A и их  вывод
allocate(a(nx,ny,nz),stat=ier); if (ier.ne.0) stop 'not allocate  a !'
allocate(axy(nx, ny),stat=ier); if (ier.ne.0) stop 'not allocate axy!'
allocate(axz(nx, nz),stat=ier); if (ier.ne.0) stop 'not allocate axz!'
allocate(ayz(ny, nz),stat=ier); if (ier.ne.0) stop 'not allocate ayz!'
read (*,*) a                                 ! Ввод и вывод A(3,4,5)
write(*,*) ' a:'; write(*,'(3i7)') a         ! по умолчанию  (ПО СТОЛБЦАМ).
call wrt_page(a)                             ! Вывод A(3,4,5) ПО СТРОКАМ
do k=1,nz;  axy=sxy(k,nx,ny,nz,a)            ! Получение k-ой из матриц XY.
  write(*,*) 'axy:  k=',k                    ! Вывод номера её листа и
  write(*,'(3i7)') axy                       !                 содержимого.
enddo
do k=1,ny;  axz=sxz(k,nx,ny,nz,a)            ! Цикл по ny матрицам XZ
 write(*,*) 'axz: k=',k                      ! Вывод номера каждой и её
 write(*,'(3i7)') axz                        !                 содержимого.
enddo
do k=1,nx;  ayz=syz(k,nx,ny,nz,a)            ! Цикл по nx матрицам YZ
  write(*,*) 'ayz: k=',k                     ! Вывод номера каждой и её
  write(*,'(4i7)') ayz                       !                 содержимого.
enddo
deallocate(a,stat=ier);   if (ier.ne.0) stop ' Не могу высвободить a!'
deallocate(axy,stat=ier); if (ier.ne.0) stop ' Не могу высвободить axy!'
deallocate(axz,stat=ier); if (ier.ne.0) stop ' Не могу высвободить axz!'
deallocate(ayz,stat=ier); if (ier.ne.0) stop ' Не могу высвободить ayz!'
end program test1
```

Содержимое файла с результатами работы test1

```
      Вывод программы       !                Пояснения
 nx=           3            !
 ny=           4            !
 nz=           5            ! Реальное расположение листов массива A(3,4,5)
 a:                         !
                            !                 __________/_________________
     1      2      3        !  Пятая         | 10000   40000  70000 100000
     4      5      6        !  страница      | 20000  /50000  80000 110000
     7      8      9        !                | 30000 / 60000  90000 120000
    10     11     12        !             __________/_________________
    10     20     30        !  Четвёртая  | 1000   4000   7000  10000
    40     50     60        !  страница   | 2000  /5000   8000  11000
    70     80     90        !             | 3000 / 6000   9000  12000
   100    110    120        !          __________/_________________
   100    200    300        !         | 100    /400    700   1000 Третья
   400    500    600        !         | 200   / 500    800   1100 страница
   700    800    900        !         | 300  /  600    900   1200
  1000   1100   1200        !       _________/_________________
  1000   2000   3000        !      | 10    40    70    100     Вторая
  4000   5000   6000        !      | 20   /50    80    110       страница
  7000   8000   9000        !      | 30  / 60    90    120
 10000  11000  12000        ! __________/_______________
 10000  20000  30000        ! | 1      4     7     10     Первая страница
 40000  50000  60000        ! | 2      5     8     11         (фасад)
 70000  80000  90000        ! | 3      6.    9     12
100000 110000 120000        !         ^...вертикальное сечение A(:,2,:)
 wrt_page(a):               !
     1      4      7     10 !  Первый вертикальный слой:
     2      5      8     11 !  ФАСАД (первая страница)
     3      6      9     12 !         Слой A(:,:,1)
                            !
    10     40     70    100 !         Вторая страница     --"--
    20     50     80    110 !         Слой A(:,:,2)
    30     60     90    120 !
                            !
   100    400    700   1000 !         Третья страница     --"--
   200    500    800   1100 !         Слой A(:,:,3)
   300    600    900   1200 !
                            !
  1000   4000   7000  10000 !         Четвёртая страница  --"--
  2000   5000   8000  11000 !         Слой A(:,:,4)
  3000   6000   9000  12000 !
                            !
 10000  40000  70000 100000 !         Пятая страница      --"--
 20000  50000  80000 110000 !         Слой A(:,:,5)
 30000  60000  90000 120000 !
```

```
axy:  k=           1        !   Те же ВЕРТИКАЛЬНЫЕ слои,
     1      2      3        !   выведенные не по математическому формату
     4      5      6        !   матрицы (три строки, четыре столбца),
     7      8      9        !   а по формату умолчания ФОРТРАНа, когда
    10     11     12        !   в строку вывода прежде всего выводится
axy:  k=           2        !   содержимое столбца. Поэтому в данном
    10     20     30        !   выводе выведенные строки файла являются
    40     50     60        !   столбцами матрицы, а выведенные столбцы
    70     80     90        !   строками матрицы.
   100    110    120        !
axy:  k=           3        !
   100    200    300        !
   400    500    600        !
   700    800    900        !
  1000   1100   1200        !
axy:  k=           4        !
  1000   2000   3000        !
  4000   5000   6000        !
  7000   8000   9000        !
 10000  11000  12000        !
axy:  k=           5        !
 10000  20000  30000        !
 40000  50000  60000        !
 70000  80000  90000        !
100000 110000 120000        !
axz: k=            1        !   Вертикальные слои, перпендикулярные фасаду
     1      2      3        !   (самый левый слой A(:,1,:).
    10     20     30        !
   100    200    300        !
  1000   2000   3000        !
 10000  20000  30000        !
axz: k=            2        !
     4      5      6        !   Слой A(:,2,:). Он намечен на реальном
    40     50     60        !   расположении листов прямым слэшем "/".
   400    500    600        !   Не забываем, что вывод матриц axz
  4000   5000   6000        !   производится по умолчанию, т.е.
 40000  50000  60000        !   очередной столбец  матрицы выводится
axz: k=            3        !   в соответствующую строку файла результата.
     7      8      9        !   о чём нужно помнить при сравнении с
    70     80     90        !   реальной схемой расположения слоёв.
   700    800    900        !
  7000   8000   9000        !
 70000  80000  90000        !
axz: k=            4        !  Слой A(:,4,:)
    10     11     12        !
   100    110    120        !
  1000   1100   1200        !
 10000  11000  12000        !
```

```
100000 110000 120000        !
ayz: k=           1         ! Горизонтальные слои, перпендикулярные фасаду
     1      4      7     10 ! (слой A(1,:,:)
    10     40     70    100 !
   100    400    700   1000 !
  1000   4000   7000  10000 !
 10000  40000  70000 100000 !
ayz: k=           2         ! Слой A(2,:,:)
     2      5      8     11 !
    20     50     80    110 !
   200    500    800   1100 !
  2000   5000   8000  11000 !
 20000  50000  80000 110000 !
ayz: k=           3         ! Слой A(3,:,:)
     3      6      9     12 !
    30     60     90    120 !
   300    600    900   1200 !
  3000   6000   9000  12000 !
 30000  60000  90000 120000 !
```

1. Включение в главную программу интерфейсного блока в данном случае необходимо, поскольку каждая из функций **sxy**, **sxz** и **syz** в качестве результата возвращает через своё имя матрицу.

2. Конечно, можно было обойтись одним оператором **allocate**:

   ```
   allocate(a(nx,ny,nz),axy(nx,ny),a(xz(nx,nz),ayz(ny,nz), stat=ier)
   ```

   Пораздельное размещение каждого массива обусловлено желанием конкретизировать причину невозможности размещения, если такое произойдёт. Так, при **nx=3000**, **ny=4000**, **nz=5000** получим: **STOP not allocate a !**, т.е. разместить массив с таким гигантским количеством элементов не удалось.

3. Значение фактического аргумента **ier** должно передаваться исключительно в ключевой форме из необязательного формального аргумента **stat**. При позиционной форме компилятор воспримет имя фактического аргумента **ier** как имя переменной, размещаемой в процессе работы программы, что будет противоречить её размещению, выполненному на шаге компиляции. Например,

   ```
   allocate(a(nx,ny,nz),ier); if (ier.ne.0) stop 'not allocate  a !'
                         1
   ошибка: Allocate-object at (1) is neither a data pointer nor an
   allocatable variable
   ```

5.1.5 Вариант 1a (модульные процедуры)

Исходный текст модуля sxyz (файл sxyz.f95)

```
module sxyz
implicit none
contains
function sxy(k,nx,ny,nz,a)       ! Можно описывать функцию и без
integer k, nx, ny, nz            ! служебного аргумента result.
integer sxy(nx,ny), a(nx,ny,nz)  ! В этом случае тип значения,
sxy=a(:,:,k)                     ! возвращаемого функцией и
end                              ! присваивание результата
function sxz(k,nx,ny,nz,a)       ! приходится описывать соответственно.
integer k, nx, ny, nz            ! Здесь использована подобная форма
integer sxz(nx,nz), a(nx,ny,nz)  ! для большей наглядности
sxz=a(:,k,:)                     ! сопоставления имени функции
end                              ! нужного сечения трёхмерного массива.
function syz(k,nx,ny,nz,a)       ! Правда, наглядность эта чисто
integer k, nx, ny, nz            ! условная. Вполне можно было бы
integer syz(ny,nz), a(nx,ny,nz)  ! использовать, например, result(s).
syz=a(k,:,:)                     ! И тогда, вместо sxy=, sxz= и syz=
end                              ! всюду было бы s=.
subroutine wrt_page(a)
integer a(:,:,:), sh(3)
integer i, j, k, nx, ny, nz
sh=shape(a); nx=sh(1); ny=sh(2); nz=sh(3); write(*,*) ' wrt_page(a):'
do k=1,nz
  write(*,'(4i7)') ((a(i,j,k),j=1,ny),i=1,nx); write(*,*)
enddo
end subroutine wrt_page
end module sxyz
```

Исходный текст главной программы test1a

```
program test1a
use sxyz
implicit none
integer,allocatable :: a(:,:,:)              ! Исходный трёхмерный массив.
integer,allocatable :: axy(:,:)              ! Его вертикальный слой (фас);
integer,allocatable :: axz(:,:)              ! верт. слой || бок.грани;
integer,allocatable :: ayz(:,:)              ! горизонт. слой || основанию.
integer nx, ny, nz, k, ier
read (*,*) nx, ny, nz                        ! Ввод числа строк, столбцов и
write(*,*) ' nx=', nx, ' ny=', ny,' nz=', nz  ! листов массива A и их  вывод
allocate(a(nx,ny,nz),stat=ier); if (ier.ne.0) stop 'not allocate  a !'
allocate(axy(nx, ny),stat=ier); if (ier.ne.0) stop 'not allocate axy!'
allocate(axz(nx, nz),stat=ier); if (ier.ne.0) stop 'not allocate axz!'
allocate(ayz(ny, nz),stat=ier); if (ier.ne.0) stop 'not allocate ayz!'
```

```
read (*,*) a                                    ! Ввод и вывод A(3,4,5)
write(*,*) ' a:'; write(*,'(3i7)') a            ! по умолчанию  (ПО СТОЛБЦАМ).
call wrt_page(a)                                ! Вывод A(3,4,5) ПО СТРОКАМ
do k=1,nz;  axy=sxy(k,nx,ny,nz,a)               ! Получение k-ой из матриц XY.
  write(*,*) 'axy:  k=',k                       ! Вывод номера её листа и
  write(*,'(3i7)') axy                          !                 содержимого.
enddo
do k=1,ny;  axz=sxz(k,nx,ny,nz,a)               ! Цикл по ny матрицам XZ
 write(*,*) 'axz: k=',k                         ! Вывод номера каждой и её
 write(*,'(3i7)') axz                           !                 содержимого.
enddo
do k=1,nx;  ayz=syz(k,nx,ny,nz,a)               ! Цикл по nx матрицам YZ
  write(*,*) 'ayz: k=',k                        ! Вывод номера каждой и её
  write(*,'(4i7)') ayz                          !                 содержимого.
enddo
deallocate(a,stat=ier);   if (ier.ne.0) stop ' Не могу высвободить a!'
deallocate(axy,stat=ier); if (ier.ne.0) stop ' Не могу высвободить axy!'
deallocate(axz,stat=ier); if (ier.ne.0) stop ' Не могу высвободить axz!'
deallocate(ayz,stat=ier); if (ier.ne.0) stop ' Не могу высвободить ayz!'
end program test1a
```

1. Размещение процедур в модуле значительно упрощает запись исходных текстов. Во-первых, не надо в главную программу включать интерфейсный блок. Интерфейс любой процедуры, включённой в модуль **sxyz**, будет известен программной единице, содержащей оператор **use sxyz**.

2. Программным единицам, не содержащим **use sxyz**, доступ к процедурам последнего будет закрыт.

3. Результат работы программы **test1a**, естественно, должен совпасть с результатом **test1**.

4. В пояснениях к условию (см. пункт **5.1.2**) говорилось, что современный ФОРТРАН позволяет значительно упростить запись интерфейса каждой из используемых процедур, исключив из их заголовка явную информацию о размерах массивов. Реализуем эту возможность в следующем пункте.

5.1.6 Вариант 1b (модульные процедуры)

Исходный текст модуля sxyzb (файл sxyzb.f95)

```
module sxyzb; implicit none
contains
function sxy(k,a); integer k,a(:,:,:),sxy(size(a,1),size(a,2)); sxy=a(:,:,k)
         end function sxy
function sxz(k,a); integer k,a(:,:,:),sxz(size(a,1),size(a,3)); sxz=a(:,k,:)
         end function sxz
function syz(k,a); integer k,a(:,:,:),syz(size(a,2),size(a,3)); syz=a(k,:,:)
         end function syz
subroutine wrt_page(a); integer a(:,:,:), sh(3)
integer i, j, k
sh=shape(a); write(*,*) ' wrt_page(a):'
do k=1,sh(3)
  write(*,'(4i7)') ((a(i,j,k),j=1,sh(2)),i=1,sh(1)); write(*,*)
enddo
end subroutine wrt_page
end module sxyzb
```

Исходный текст главной программы test1b

```
program test1b; use sxyzb; implicit none
integer,allocatable :: a(:,:,:),&            ! Исходный трёхмерный массив.
&                      axy(:,:), axz(:,:),&  ! Его верт. слои (фас и ||
&                      ayz(:,:)              ! бок.грани) и горизонтальный.
integer nx, ny, nz, k, ier
read (*,*) nx, ny, nz                        ! Ввод числа строк, столбцов и
write(*,*) ' nx=', nx, ' ny=', ny,' nz=', nz ! листов массива A и их  вывод
allocate(a(nx,ny,nz),stat=ier); if (ier.ne.0) stop 'not allocate  a !'
allocate(axy(nx, ny),stat=ier); if (ier.ne.0) stop 'not allocate axy!'
allocate(axz(nx, nz),stat=ier); if (ier.ne.0) stop 'not allocate axz!'
allocate(ayz(ny, nz),stat=ier); if (ier.ne.0) stop 'not allocate ayz!'
read (*,*) a;                                 ! Ввод и вывод A(3,4,5)
write(*,*) ' a:'; write(*,'(3i7)') a          ! по умолчанию  (ПО СТОЛБЦАМ).
call wrt_page(a)                              ! Вывод A(3,4,5) ПО СТРОКАМ
do k=1,nz;  axy=sxy(k,a);                     ! Выбор k-ой из nz матриц XY.
  write(*,*) 'axy: k=',k; write(*,'(3i7)') axy ! Вывод номера матрицы и её
enddo                                         !                 содержимого.
do k=1,ny;  axz=sxz(k,a)                      ! Выбор k-ой из ny матриц XZ.
  write(*,*) 'axz: k=',k; write(*,'(3i7)') axz ! Вывод номера матрицы и её
enddo                                         !                  содержимого.
do k=1,nx;  ayz=syz(k,a)                      ! Выбор k-ой из nx матриц YZ.
  write(*,*) 'ayz: k=',k; write(*,'(4i7)') ayz ! Вывод номера матрицы и её
enddo                                         !                  содержимого.
deallocate(a,axy,axz,ayz,stat=ier); if (ier.ne.0) stop ' Do not free !'
end program test1b
```

1. Результат **test1b** должен совпасть с результатами **test1a** и **test1**.
2. Возможна процедура **section(key,k,a,sha)**, которая
   - по ключу **key** (номер измерения массива **a**),
   - по **k** (номер сечения, соответствующего **key**),
   - по числам строк и столбцов последнего, хранящимся в столбцах матрицы **sha(2,3)**,

   возвращала бы через своё имя матрицу соответствующего сечения **a**. Приведём её в следующем пункте.

5.1.7 Варианты 1c и 1d

Уяснение ситуации

1. **a** — трёхмерный массив с длинами экстентов **nx**, **ny** и **nz**.
2. Сопоставим ключ **key=1** первому измерению с длиной экстента **nx**;
   ключ **key=2** — второму (с длиной экстента **ny**) и
   ключ **key=3** — третьему (с длиной экстента **nz**).
3. **key=1** соответствуют **nx** матриц, каждая размером **ny*nz**.
4. **key=2** соответствуют **ny** матриц, каждая размером **nx*nz**.
5. **key=3** соответствуют **nz** матриц, каждая размером **nx*ny**.
6. Тогда, если три столбца двухстроковой матрицы **sha(2,3)** заполнить соответственно парами чисел (**ny**,**nz**), (**nx**,**nz**) и (**nx**,**ny**) (того же, кстати, можно достичь и посредством

```
sha(:,1)=shape(a(1,:,:))
sha(:,2)=shape(a(:,1,:))
sha(:,3)=shape(a(:,:,1))
```

   ), то в зависимости от значения ключа **key=1, 2, 3** можно при описании **w** (формального аргумента-результата) — матрицы, возвращаемой через имя функции, указать длины экстентов, ей соответствующих, например:

```
function section(key,k,a,sha)
integer key, k, a(:,:,:), sha(2,3)
integer w(sha(1,key),sha(2,key))
```

Исходный текст модуля sxyzc (файл sxyzc.f95)

```
module sxyzc; implicit none
contains
function section(key,k,a, sha) result (w)
integer key, k, a(:,:,:), sha(2,3), w(sha(1,key),sha(2,key))
select case (key)
  case(1); w=a(k,:,:); case(2); w=a(:,k,:); case(3); w=a(:,:,k)
end select
end function section
subroutine wrt_page(a); integer a(:,:,:), i, j, k, sh(3)
sh=shape(a); write(*,*) ' wrt_page(a):'
do k=1,sh(3)
  write(*,'(4i7)') ((a(i,j,k),j=1,sh(2)),i=1,sh(1)); write(*,*)
enddo
end subroutine wrt_page
end module sxyzc
```

Исходный текст главной программы test1c

```
program test1c; use sxyzc; implicit none
integer,allocatable :: a(:,:,:) ! Исходный трёхмерный массив A
integer,allocatable :: w(:,:)   ! W - его сечение (матрица).
character(3) sn                 ! внутр.файл для хранения числа строк.
character(60) sf                ! строка формата вывода столбца.
integer sha(2,3)   ! Столбцы - размеры матриц, ортогональной измерению.
integer nx, ny, nz ! Длины экстентов (измерений) массива A.
integer key,ier, i ! key: номер оси, для которой выводятся сечения A.
read (*,*) nx, ny, nz                               ! Ввод длин экстентов A
write(*,'(" nx=",i3/" ny=",i3/" nz=",i3)') nx,ny,nz ! Их контрольный вывод.
allocate(a(nx,ny,nz),stat=ier); if (ier.ne.0) stop 'not allocate  a !'
read (*,*) a; write(*,*) ' a:'; write(*,'(3i7)') a  ! Ввод/вывод по столбцам
call wrt_page(a)                                    ! Вывод A ПО СТРОКАМ.
sha(:,1)=shape(a(1,:,:)); sha(:,2)=shape(a(:,1,:))
                          sha(:,3)=shape(a(:,:,1))
do key=3,1,-1                                  ! Цикл по номеру оси:
  allocate(w(sha(1,key),sha(2,key)),stat=ier)! размещение сечения
  if (ier/=0) stop ' not allocate w !'         !  (удачно ли оно?);
  write(sn,'(i3)') sha(1,key)                  ! sn : запись числа строк
  sf='('//sn//'i7)'                            ! sf : формат вывода столбцов
  do i=1,size(a,key)                           ! Цикл по сечениям оси key.
    write(*,'(" Форма ",i2,"-го слоя при key=",i1," : ",2i3)') i,key, shape(w)
    w=section(key,i,a,sha)                     ! Выборка i-го сечения.
    write(*,trim(sf)) w                        ! Его вывод по столбцам.
  enddo
  deallocate(w)                                ! Освобождение памяти W
enddo
deallocate(a,stat=ier); if (ier.ne.0) stop ' Do not free !'
end program test1c
```

```
nx=  3                          !     Результат работы test1c:

ny=  4
nz=  5
 a:
     1      2      3
     4      5      6
     7      8      9
    10     11     12
    10     20     30
    40     50     60
    70     80     90
   100    110    120
   100    200    300
   400    500    600
   700    800    900
  1000   1100   1200
  1000   2000   3000
  4000   5000   6000
  7000   8000   9000
 10000  11000  12000
 10000  20000  30000
 40000  50000  60000
 70000  80000  90000
100000 110000 120000
 wrt_page(a):
     1      4      7     10
     2      5      8     11
     3      6      9     12

    10     40     70    100
    20     50     80    110
    30     60     90    120

   100    400    700   1000
   200    500    800   1100
   300    600    900   1200

  1000   4000   7000  10000
  2000   5000   8000  11000
  3000   6000   9000  12000

 10000  40000  70000 100000
 20000  50000  80000 110000
 30000  60000  90000 120000


Форма  1-го слоя при key=3 :   3  4
```

```
     1      2      3
     4      5      6
     7      8      9
    10     11     12
Форма  2-го слоя при key=3 :   3  4
    10     20     30
    40     50     60
    70     80     90
   100    110    120
Форма  3-го слоя при key=3 :   3  4
   100    200    300
   400    500    600
   700    800    900
  1000   1100   1200
Форма  4-го слоя при key=3 :   3  4
  1000   2000   3000
  4000   5000   6000
  7000   8000   9000
 10000  11000  12000
Форма  5-го слоя при key=3 :   3  4
 10000  20000  30000
 40000  50000  60000
 70000  80000  90000
100000 110000 120000
Форма  1-го слоя при key=2 :   3  5
     1      2      3
    10     20     30
   100    200    300
  1000   2000   3000
 10000  20000  30000
Форма  2-го слоя при key=2 :   3  5
     4      5      6
    40     50     60
   400    500    600
  4000   5000   6000
 40000  50000  60000
Форма  3-го слоя при key=2 :   3  5
     7      8      9
    70     80     90
   700    800    900
  7000   8000   9000
 70000  80000  90000
Форма  4-го слоя при key=2 :   3  5
    10     11     12
   100    110    120
  1000   1100   1200
 10000  11000  12000
100000 110000 120000
```

```
Форма  1-го слоя при key=1 :   4  5
     1      4      7     10
    10     40     70    100
   100    400    700   1000
  1000   4000   7000  10000
 10000  40000  70000 100000
Форма  2-го слоя при key=1 :   4  5
     2      5      8     11
    20     50     80    110
   200    500    800   1100
  2000   5000   8000  11000
 20000  50000  80000 110000
Форма  3-го слоя при key=1 :   4  5
     3      6      9     12
    30     60     90    120
   300    600    900   1200
  3000   6000   9000  12000
 30000  60000  90000 120000
```

1. Заметим, что при возврате массива именно через имя функции недопустимо описывать возвращаемый формальный аргумент **w** (матрицу) посредством **w(:,:)**, т.е. как массив, перенимающий форму фактического аргумента, так как в данном случае **w** локальный объект функции, а не адрес фактического аргумента. Именно поэтому в качестве дополнительного аргумента функции и нужен массив **sha(2,3)**: для указания размеров локального массива **w**.

2. Если бы мы описывали подпрограмму, то упомянутой проблемы не было, так как работа подпрограммы велась бы по адресу фактического аргумента и соответствующий формальный аргумент на законном основании мог бы быть описан, как массив, перенимающий форму фактического аргумента. Например,

Исходный текст модуля sxyzd (файл sxyzd.f95)

```
module sxyzd; implicit none
contains
subroutine section1(key,k,a, w); integer key, k, a(:,:,:), w(:,:)
select case (key)
  case(1); w=a(k,:,:); case(2); w=a(:,k,:); case(3); w=a(:,:,k)
end select
end subroutine section1
subroutine wrt_page(a); integer a(:,:,:), i, j, k, sh(3)
sh=shape(a); write(*,*) ' wrt_page(a):'
```

```
do k=1,sh(3)
  write(*,'(4i7)') ((a(i,j,k),j=1,sh(2)),i=1,sh(1)); write(*,*)
enddo
end subroutine wrt_page
end module sxyzd
```

Исходный текст главной программы test1d

```
program test1d; use sxyzd; implicit none
integer,allocatable :: a(:,:,:) ! Исходный трёхмерный массив A
integer,allocatable :: w(:,:)   ! W - его сечение (матрица).
character(3) sn                 ! внутр.файл для хранения числа строк.
character(60) sf                ! строка формата вывода столбца.
integer sha(2,3)   ! Столбцы - размеры матриц, ортогональной измерению.
integer nx, ny, nz ! Длины экстентов (измерений) массива A.
integer key,ier, i ! key: номер оси, для которой выводятся сечения A.
read (*,*) nx, ny, nz                                ! Ввод длин экстентов A
write(*,'(" nx=",i3/" ny=",i3/" nz=",i3)') nx,ny,nz ! Их контрольный вывод.
allocate(a(nx,ny,nz),stat=ier); if (ier.ne.0) stop 'not allocate  a !'
read(*,*) a;write(*,*)' a:'; write(*,'(3i7)') a ! Ввод/вывод по столбцам
call wrt_page(a)                                  ! Вывод A ПО СТРОКАМ.
sha(:,1)=shape(a(1,:,:)); sha(:,2)=shape(a(:,1,:))
                          sha(:,3)=shape(a(:,:,1))
do key=3,1,-1                                    ! Цикл по номеру оси:
  allocate(w(sha(1,key),sha(2,key)),stat=ier)! размещение сечения
  if (ier/=0) stop ' not allocate w !'       !  (удачно ли оно?);
  write(sn,'(i3)') sha(1,key)                ! sn : запись числа строк
  sf='('//sn//'i7)'                          ! sf : формат вывода столбцов
  do i=1,size(a,key)                         ! Цикл по сечениям оси key.
   write(*,'(" Форма ",i2,"-го слоя при key=",i1," : ",2i3)')i,key,shape(w)
    call section1(key,i,a,w)                 ! Выборка i-го сечения.
    write(*,trim(sf)) w                      ! Его вывод по столбцам.
  enddo
  deallocate(w)                              ! Освобождение памяти W
enddo
deallocate(a,stat=ier); if (ier.ne.0) stop ' Do not free !'
end program test1d
```

Для экономии времени и памяти оформление алгоритма подпрограммой в данном случае предпочтительнее.

5.2 Задача 2

5.2.1 Условие

Написать функцию **myresh2(ord,shape,b)** (по сути — упрощённый аналог встроенной **reshape**), преобразующую вектор **b(s(1)*s(2))** в матрицу из **s(1)** строк и **s(2)** столбцов. Заполнение матрицы проводить согласно значениям элементов аргумента **ord(1:2)**:

при **ord=(/1,2/)** вести заполнение по росту номера строки;

при **ord=(/2,1/)** вести заполнение по росту номера столбца.

В главной программе можно использовать **reshape**, но исключительно с целью подтверждения правильности работы **myresh2**.

5.2.2 Вариант 1 (myresh2 — внешняя функция)

1. Поскольку требуется написать именно **функцию**, т.е. процедуру, возвращающую через своё имя **матрицу**), то придётся использовать явное описание интерфейса функции оператором **interface**.

2. Заголовок функции **myresh2(ord,shape,b)**, приведённый в условии задачи имеет три аргумента: **ord(1:2)** — вектор из двух элементов; **shape(1:2)** — вектор из двух элементов (первый содержит число строк матрицы, второй — число её столбцов). **b** — вектор, содержимое которого следует разместить (способом, указываемым **ord**) по элементам матрицы, возвращаемой через имя функции **myresh2**.

3. В тексте программы использованы статические массивы. Для указания указания числа строк и числа столбцов матрицы используются константы **nx=3** и **ny=4** соответственно.

4. Результат функции **myresh2** присваивается матрице **aa(nx,ny)**, а результат функции **reshape** — матрице **a**.

5. Вектор **b**, из которого черпаются значения, заполняющие матрицы **a** и **aa**, изначально заполняется посредством **reshape** значениями введённых элементов матрицы **a**.

```
b=reshape(a,(/nx*ny/))
```

6. Вызывать **myresh2** придётся два раза: сначала — при **ord=(/1,2/)**, а затем — при **ord=(/2,1/)**.

Исходный текст главной программы test2

```
program test2; implicit none
interface
function myresh2(o,s,b); integer s(2),o(2); integer b(s(1)*s(2))
                                            integer myresh2(3,4)
end function myresh2
end interface
integer, parameter :: nx=3, ny=4  ! nx - число строк; ny - число столбцов
integer i
integer  a(nx,ny)                 !  a - вводимая матрица
integer b(nx*ny)                  !  b - её одномерный эквивалент
integer aa(nx,ny)                 ! aa - матрица, получаемая myresh2 из b
integer s(2)                      !  s - форма матрицы.
integer o(2)                      !  o - вектор очерёдности заполнения
write(*,*) ' nx=', nx, ' ny=', ny ! Вывод числа строк и числа столбцов.
read (*,*) a                      ! Ввод матрицы a.
write(*,*) ' a:'                  ! Её вывод
write(*,'(3i7)') a                !    в режиме умолчания.
b=reshape(a,(/nx*ny/))            ! Формировка из матрицы a вектора b
write(*,*) ' b='                  ! Его вывод
write(*,'(12i5)') b               !    в режиме умолчания.
a=reshape(b,(/3,4/));             ! Формировка из вектора b матрицы a
write(*,*)' a: ';                 ! Её вывод в режиме умолчания должен
write(*,'(3i7)') a; write(*,*)    ! совпасть с её предыдущим выводом.
s(1)=nx; s(2)=ny                  ! Задание формы матрицы и
o(1)= 1; o(2)=2                   ! очерёдности заполнения её элементов
 a=reshape(b,s,order=o)           ! Формировка a из b через reshape
aa=myresh2(o,s,b)                 ! Формировка aa из b через myresh2
write(*,1000)                     ! Вывод имён табличек  результатов
do i=1,nx                         !
  write(*,1002) a(i,:), aa(i,:)   ! Вывод результатов reshape и myresh2
enddo                             !
o(1)=2                            ! Изменение очерёдности заполнения.
o(2)=1                            !
a=reshape(b,s,order=o)            ! Формировка a из b через reshape
aa=myresh2(o,s,b)                 ! Формировка aa из b через myresh2
write(*,1001)                     ! Вывод имён табличек  результатов
do i=1,nx                         !
  write(*,1002) a(i,:), aa(i,:)   ! Вывод результатов reshape и myresh2
enddo
1000 format(  '   reshape:(1,2)      myresh2:(1,2)')
1001 format(/,'   reshape:(2,1)      myresh2:(2,1)')
1002 format(4i4,3x,4i4)
end
```

Исходный текст функции myresh2

```
function myresh2(o,s,b)           ! Заголовок функции myresh2,
implicit none                     ! возвращающей чрез своё имя МАССИВ.
integer s(2),o(2)
integer b(s(1)*s(2))
integer myresh2(s(1),s(2))
integer l, i, j                   !  k=o(1)-1
do i=1,s(1)                       !
do j=1,s(2)                       !
  select case(o(1))               !
    case(1)                       !  l=  k  * (j+(i-1)*s(2)) +
           l=i+(j-1)*s(1)         !    (1-k)* (i+(j-1)*s(1))
    case(2)                       !
           l=j+(i-1)*s(2)         !
  end select
  myresh2(i,j)=b(l)
enddo
enddo
end
```

Результат тестирования myresh2

```
 nx=           3  ny=           4
 a:                                  ! Напоминание! Поскольку матрица A
     1       2       3              ! выводится по формату умолчания, то
     4       5       6              ! в строку выводится содержимое столбцавы
     7       8       9              ! матрицы. Так как столбцов 4, то и
    10      11      12              ! выведенных строк тоже 4.
 b=
   1    2    3    4    5    6    7    8    9   10   11   12
 a:
     1       2       3              ! Та же самая матрица A, полученная
     4       5       6              ! reshape из вектора B, который в свою
     7       8       9              ! очередь был получен из введённой A
    10      11      12              ! опять-таки посредством reshape.

  reshape:(1,2)       myresh2:(1,2)  ! Здесь вывод матриц A и AA
  1   4   7  10       1   4   7  10  ! осуществляется не по умолчанию
  2   5   8  11       2   5   8  11  ! а по строкам. Как видно, результаты
  3   6   9  12       3   6   9  12  ! по смыслу совпадают с предыдущими.

  reshape:(2,1)       myresh2:(2,1)  ! Здесь вывод тоже по строкам. Но
  1   2   3   4       1   2   3   4  ! заполнение матриц A и AA происходило
  5   6   7   8       5   6   7   8  ! не по столбцам, а по строкам.
  9  10  11  12       9  10  11  12
```

5.2.3 Вариант 1a (myresh2 — модульная функция)

Исходный текст главной программы test2a

```
program test2a; use myresh; implicit none
!interface
!function myresh2(o,s,b); integer s(2),o(2); integer b(s(1)*s(2))
!                                            integer myresh2(3,4)
!end function myresh2
!end interface
integer, parameter :: nx=3, ny=4  ! nx - число строк; ny - число столбцов
integer i
integer  a(nx,ny)                 !  a - вводимая матрица
integer b(nx*ny)                  !  b - её одномерный эквивалент
integer aa(nx,ny)                 ! aa - матрица, получаемая myresh2 из b
integer s(2)                      !  s - форма матрицы.
integer o(2)                      !  o - вектор очерёдности заполнения
write(*,*) ' nx=', nx, ' ny=', ny ! Вывод числа строк и числа столбцов.
read (*,*) a                      ! Ввод матрицы a.
write(*,*) ' a:'                  ! Её вывод
write(*,'(3i7)') a                !    в режиме умолчания.
b=reshape(a,(/nx*ny/))            ! Формировка из матрицы a вектора b
write(*,*) ' b='                  ! Его вывод
write(*,'(12i5)') b               !    в режиме умолчания.
a=reshape(b,(/3,4/));             ! Формировка из вектора b матрицы a
write(*,*)' a: ';                 ! Её вывод в режиме умолчания должен
write(*,'(3i7)') a; write(*,*)    ! совпасть с её предыдущим выводом.
s(1)=nx; s(2)=ny                  ! Задание формы матрицы и
o(1)= 1; o(2)=2                   ! очерёдности заполнения её элементов
 a=reshape(b,s,order=o)           ! Формировка a из b через reshape
aa=myresh2(o,s,b)                 ! Формировка aa из b через myresh2
write(*,1000)                     ! Вывод имён табличек  результатов
do i=1,nx                         !
  write(*,1002) a(i,:), aa(i,:)   ! Вывод результатов reshape и myresh2
enddo                             !
o(1)=2                            ! Изменение очерёдности заполнения.
o(2)=1                            !
a=reshape(b,s,order=o)            ! Формировка a из b через reshape
aa=myresh2(o,s,b)                 ! Формировка aa из b через myresh2
write(*,1001)                     ! Вывод имён табличек  результатов
do i=1,nx                         !
  write(*,1002) a(i,:), aa(i,:)   ! Вывод результатов reshape и myresh2
enddo
1000 format(  '   reshape:(1,2)      myresh2:(1,2)')
1001 format(/,'   reshape:(2,1)      myresh2:(2,1)')
1002 format(4i4,3x,4i4)
end program test2a
```

Исходный текст модуля myresh с функцией myresh2

```
module myresh; implicit none
contains
function myresh2(o,s,b) result(r) ! Заголовок функции myresh2,
integer s(2),o(2)                 ! возвращающей чрез своё имя МАССИВ.
integer b(s(1)*s(2))
integer r(s(1),s(2))
integer l, i, j                   !  k=o(1)-1
do i=1,s(1)                       !
do j=1,s(2)                       !
  select case(o(1))               !
    case(1)                       !  l=  k  * (j+(i-1)*s(2)) +
          l=i+(j-1)*s(1)          !    (1-k)* (i+(j-1)*s(1))
    case(2)                       !
          l=j+(i-1)*s(2)          !
  end select
  r(i,j)=b(l)
enddo
enddo
end function myresh2
end module myresh
```

Результат работы test2a

```
 nx=           3  ny=           4
 a:                                    ! Напоминание! Поскольку матрица A
     1      2      3                   ! выводится по формату умолчания, то
     4      5      6                   ! в строку выводится содержимое столбцавы
     7      8      9                   ! матрицы. Так как столбцов 4, то и
    10     11     12                   ! выведенных строк тоже 4.
 b=
   1    2    3    4    5    6    7    8    9   10   11   12
 a:
     1      2      3                   ! Та же самая матрица A, полученная
     4      5      6                   ! reshape из вектора B, который в свою
     7      8      9                   ! очередь был получен из введённой A
    10     11     12                   ! опять-таки посредством reshape.

  reshape:(1,2)      myresh2:(1,2)     ! Здесь вывод матриц A и AA
  1   4   7  10      1   4   7  10     ! осуществляется не по умолчанию
  2   5   8  11      2   5   8  11     ! а по строкам. Как видно, результаты
  3   6   9  12      3   6   9  12     ! по смыслу совпадают с предыдущими.

  reshape:(2,1)      myresh2:(2,1)     ! Здесь вывод тоже по строкам. Но
  1   2   3   4      1   2   3   4     ! заполнение матриц A и AA происходило
  5   6   7   8      5   6   7   8     ! не по столбцам, а по строкам.
  9  10  11  12      9  10  11  12
```

5.3 Задача 3

5.3.1 Условие

Модифицировать **myresh2** так, чтобы параметр **ord** можно было не писать при вызове функции, если **ord=(/1,2/)**.

1. Вопрос:

   «Как сообщить компилятору, что какой-то аргумент функции может или присутствовать, или отсутствовать при её вызове?»

   Ответ:

   «Описание соответствующего формального аргумента необходимо снабдить атрибутом **optional**».

   Почему атрибуту сопоставлено слово **optional** — *опционный*?

   У многих **Linux**-команд имеются опции: хотим — пользуемся ими; не хотим — не пользуемся (даже и не пишем их). Поэтому аргументы процедур ФОРТРАНа, наделяемые подобным свойством, стали называть **опционными** (соответствующий им атрибут описания — служебное слово **optional**). Например,

```
function myresh2(s,b,o) result(r); implicit none
integer s(2), b(s(1)*s(2)), r(s(1),s(2))
integer, optional:: o(2)    ! Параметр O может при вызове myresh2
                            ! или присутствовать, или отсутствовать.
```

2. Вопрос:

   «Как внутри функции проверить: имеется ли при её вызове аргумент, который может отсутствовать, или нет?»

   Ответ:

   «Для этой цели в современном ФОРТРАНе используется встроенная функция **present**.»

   Например, **if (present(o)) then... ; else... ; endif**.

3. Организовать реакцию тела функции на альтернативные ситуации можно по разному. Приведённый ниже вариант нацелен на краткость исходного кода, которая достигается за счёт мнокократных проверок условия **q(1)==2**.

Исходный текст главной программы test3

```
program test3
implicit none
interface
function myresh2(s,b,o); integer, optional:: o(2); integer s(2);
                         integer b(s(1)*s(2)); integer myresh2(3,4)
end function myresh2
end interface
integer, parameter :: nx=3, ny=4  ! nx - число строк; ny - число столбцов
integer i
integer  a(nx,ny)                 !  a - вводимая матрица
integer b(nx*ny)                  !  b - её одномерный эквивалент
integer aa(nx,ny)                 ! aa - матрица, получаемая myresh2 из b
integer s(2)                      !  s - форма матрицы.
integer o(2)                      !  o - вектор очерёдности заполнения
write(*,*) ' nx=', nx, ' ny=', ny ! Вывод числа строк и числа столбцов.
read (*,*) a                      ! Ввод матрицы a.
write(*,*) ' a:'                  ! Её вывод
write(*,'(3i7)') a                !    в режиме умолчания.
b=reshape(a,(/nx*ny/))            ! Формировка из матрицы a вектора b
write(*,*) ' b='                  ! Его вывод
write(*,'(12i5)') b               !    в режиме умолчания.
a=reshape(b,(/3,4/));             ! Формировка из вектора b матрицы a
write(*,*)' a: ';                 ! Её вывод в режиме умолчания должен
write(*,'(3i7)') a; write(*,*)    ! совпасть с её предыдущим выводом.
s(1)=nx; s(2)=ny                  ! Задание формы матрицы.
                                  ! Очерёдность заполнения ПО УМОЛЧАНИЮ.
 a=reshape(b,s)                   ! Формировка a из b через reshape
aa=myresh2(s,b)                   ! Формировка aa из b через myresh2
write(*,1000)                     ! Вывод имён табличек  результатов
do i=1,nx                         !
  write(*,1002) a(i,:), aa(i,:)   ! Вывод результатов reshape и myresh2
enddo                             !
o(1)=2                            ! Изменение очерёдности заполнения.
o(2)=1                            !
a=reshape(b,s,order=o)            ! Формировка a из b через reshape
aa=myresh2(s,b,o)                 ! Формировка aa из b через myresh2
write(*,1001)                     ! Вывод имён табличек  результатов
do i=1,nx                         !
  write(*,1002) a(i,:), aa(i,:)   ! Вывод результатов reshape и myresh2
enddo
1000 format(  '   reshape:(1,2)      myresh2:(1,2)')
1001 format(/,'   reshape:(2,1)      myresh2:(2,1)')
1002 format(4i4,3x,4i4)
end program test3
```

Исходный текст функции myresh2

```
function myresh2(s,b,o) result(r) ! Заголовок функции myresh2,
implicit none                     ! возвращающей через своё имя МАССИВ.
integer s(2)
integer b(s(1)*s(2))
integer r(s(1),s(2))
integer, optional:: o(2)     ! Параметр o обязателен лишь при заполнении
integer q(2) /1,2/           ! матрицы вдоль строки; заполнение матрицы
integer l, i, j              ! вниз по столбцу  его не требует.
if (present(o)) q=o          ! Функция PRESENT (o) выработает true, если
do i=1,s(1)                  ! аргумент ‘o‘ присутствует при вызове
do j=1,s(2)                  ! myresh2.
  l=i+(j-1)*s(1)
  if (q(1)==2) l=j+(i-1)*s(2)
  r(i,j)=b(l)
enddo
enddo
end function myresh2
```

Результат тестирования myresh2

```
 nx=           3  ny=           4
 a:
     1       2       3
     4       5       6
     7       8       9
    10      11      12
 b=
   1    2    3    4    5    6    7    8    9   10   11   12
 a:
     1       2       3
     4       5       6
     7       8       9
    10      11      12

  reshape:(1,2)      myresh2:(1,2)
  1   4   7  10      1   4   7  10
  2   5   8  11      2   5   8  11
  3   6   9  12      3   6   9  12

  reshape:(2,1)      myresh2:(2,1)
  1   2   3   4      1   2   3   4
  5   6   7   8      5   6   7   8
  9  10  11  12      9  10  11  12
```

5.4 Задача 4

5.4.1 Условие

Написать функцию **myresh3(ord,shape,b)** (упрощённый аналог встроенной **reshape**), которая преобразует вектор **b(s(1)*s(2)*s(3))** в трёхмерный массив из **s(1)** строк, **s(2)** столбцов и **s(3)** листов. Очерёдность заполнения элементов трёхмерного массива должна определяться содержимым вектора **ord(1:3)**. Продемонстрировать правильную работу **myresh3** для всех наборов **ord**=(/**1,2,3**/), (/**2,1,3**/), (/**1,3,2**/), (/**3,1,2**/), (/**2,3,1**/), (/**3,2,1**/).

5.4.2 Уяснение ситуации

```
 Заполнение массива a(3,4,5) ..................: o=(/1,2,3/)

b:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
   21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
   41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

 Быстрее всего меняется номер  строки: o(1)=1.
 Медленнее всего меняется номер листа: o(3)=3, т.е. идёт
 заполнение каждого листа по его столбцам. Иначе говоря,
 последовательно по столбцам заполняются сечения a(:,:,k).
 Это матрицы размером из 3 строк и 4 столбцов для каждого k=1(1)5.
 l=1; do k=1,nz; do j=1,ny; do i=1,nx; aa(i,j,k)=b(l); l=l+1; enddo
                   _________/___________
                   | 49    52  55  58     k=5  Пятый лист
            К      | 50   /53  56  59
                   | 51  / 54  57  60
          О      _______/____________
            И   | 37   40   43  46      k=4  Четвёртый лист
        Ш       | 38  /41   44  47
          Г     | 39 / 42   45  48
      Е     ________/______________
        И  | 25    28    31   34     k=3  Третий лист
    Р      | 26   /29    32   35
      Н    | 27  / 30    33   36
  О     ________/_______________
    К  | 13    16    19    22      k=2  Второй лист
К      | 14   /17    20    23
       | 15  / 18    21    24
  __________/_______________
  | 1      4     7     10        k=1  Первый лист
  | 2      5     8     11
  | 3      6.    9     12
```

```
Заполнение массива a(3,4,5) при o=(/2,1,3/)

b:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
   21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
   41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

 Быстрее всего меняется номер столбца: o(1)=2.
 Медленнее всего меняется номер листа: o(3)=3.
 Заполняются те же самые сечения a(:,:,k) по СТРОКАМ:

                  _________/___________
                 | 49    50   51  52     k=5  Пятый лист
                 | 53   /54   55  56
                 | 57  / 58   59  60
               _______/____________
              | 37   38    39   40      k=4  Четвёртый лист
              | 41  /42    43   44
              | 45 / 46    47   48
            ________/______________
           | 25    26    27   28        k=3  Третий лист
           | 29   /30    31   32
           | 23  / 34    35   36
         ________/_______________
        | 13    14    15    16         k=2  Второй лист
        | 17   /18    19    20
        | 21  / 22    23    24
  __________/_______________
  | 1      2     3     4           k=1  Первый лист
  | 5      6     7     8
  | 9     10    11    12

l=1
do j=1,ny
  do k=1,nz
    do i=1,nx
      aa(i,j,k)=b(l)
      l=l+1
    enddo; ...
```

```
 Заполнение массива a(3,4,5) при o=(/1,3,2/)

 b:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
    21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
    41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

Быстрее всего  меняется  номер  строки a(3,4,5): o(1)=1.
Медленнее всего меняется номер столбца a(3,4,5): o(3)=2,
т.е. идёт заполнение по столбцам сечения a(:,j,:),
которое параллельно корешку книги.

Иначе говоря, для каждого из четырёх столбцов
массива a(3,4,5) заполняем соответствующую матрицу
размером 3*5 по её столбцам.

                  _________/___________
                  | 13    28   43  58    k=5  Пятый лист
                  | 14   /29   44  59
                  | 15  / 30   45  60
                _______/____________
               | 10   25    40   55    k=4  Четвёртый лист
               | 11  /26    41   56
               | 12 / 27    42   57
             _______/______________
            | 7    22    37   52     k=3 Третий лист
            | 8   /23    38   53
            | 9  / 24    39   54
          _______/_______________
         | 4    19    34    49      k=2 Второй лист
         | 5   /20    35    50
         | 6  / 21    36    51
    ________/_______________
   | 1     16     31     46     k=1 Первый лист
   | 2     17     32     47
   | 3     18     33     48

l=1
do j=1,ny
  do k=1, nz
    do i=1,nx
      aa(i,j,k)= b(l)
      l=l+1
    enddo ...
```

Заполнение массива a(3,4,5) при o=(/3,1,2/)

```
 b:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
    21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
    41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
```

Быстрее всего меняется номер листа a(3,4,5): o(1)=3.
Медленнее всего меняется номер столбца a(3,4,5): o(3)=2,
т.е. идёт заполнение по строкам тех же, что и в предыдущем случае
сечений a(:,j,:) --- матриц размером из 3 строк и 5 столбцов.
Заполняются последовательно: a(:,1,k), k=1,5; a(:,2,k), k=1,5;
                               a(:,3,k), k=1,5; a(:,4,k), k=1,5;
Иначе говоря, для каждого из четырёх столбцов
массива a(3,4,5) заполняем соответствующую матрицу
размером 3*5 по её строкам.

```
                  _________/___________
                  |  5    20   35  50    k=5  Пятый лист
                  | 10   /25   40  55
                  | 15  / 30   45  60
               ________/____________
               |  4    19    34   49    k=4  Четвёртый лист
               |  9   /24    39   54
               | 14  / 29    44   59
            ________/______________
            |  3    18    33   48     k=3 Третий лист
            |  8   /23    38   53
            | 13  / 28    43   58
         ________/_______________
         |  2    17    32    47      k=2 Второй лист
         |  7   /22    37    52
         | 12  / 27    42    57
   _________/_______________
  |  1     16     31     46     k=1 Первый лист
  |  6     21     36     51
  | 11     26     41     56
```

```
l=1
do j=1,ny
  do i=1,nx
    do k=1,nz
      a(i,j,k)=b(l)
       l=l+1
    enddo ...
```

```
Заполнение массива a(3,4,5) при o=(/2,3,1/)

 b:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
    21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
    41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

Быстрее  всего меняется номер столбца a(3,4,5): o(1)=2.
Медленнее всего меняется номер строки a(3,4,5): o(3)=1,
т.е. идёт заполнение по строкам  сечений a(i,:,:) ---
матриц размером из 4 строк и 5 столбцов.

Иначе говоря, для каждой из трёх строк массива a(3,4,5)
заполняем соответствующую матрицу размером 4*5 по её строкам.

                 --------/-----------
                 | 17   18   19  20    k=5  Пятый лист
                 | 37  /38   39  40
                 | 57 / 58   59  60
               -------/------------
              | 13   14    15   16    k=4  Четвёртый лист
              | 33  /34    35   36
              | 53 / 54    55   56
            -------/--------------
           |  9   10    11   12     k=3 Третий лист
           | 29  /30    31   32
           | 49 / 50    51   52
         -------/---------------
        |  5   /6    7    8      k=2 Второй лист
        | 25  /26   27   28
        | 45 / 46   47   48
  --------/---------------
 |  1     2     3     4     k=1 Первый лист
 | 21    22    23    24
 | 41    42    43    44

l=1
do i=1,nx
  do k=1,nz
    do j=1,ny
      a(i,j,k)=b(l)
      l=l+1
    enddo ...
```

Заполнение массива a(3,4,5) при o=(/3,2,1/)

```
 b:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
    21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
    41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
```

Быстрее всего меняется номер листа a(3,4,5): o(1)=3.
Медленнее всего меняется номер строки a(3,4,5): o(3)=1,
т.е. идёт заполнение тех же сечений a(i,:,:)
--- матриц размером из 4 строк и 5 столбцов для каждой из
трёх строковых плоскостей a(3,4,5).

Отличие лишь в том, что вместо строки матрицы (как было
в предыдущем случае) заполняется её столбец.

```
                  _________/___________
                  |  5   10   15  20    k=5  Пятый лист
                  | 25  /30   35  40
                  | 45 / 50   55  60
               _______/____________
               |  4   /9   14   19    k=4  Четвёртый лист
               | 24  /29   34   39
               | 44 / 49   54   59
            _______/______________
            |  3   /8   13   18     k=3 Третий лист
            | 23  /28   33   38
            | 43 / 48   53   58
         _______/_______________
         |  2   /7   12   17      k=2 Второй лист
         | 22  /27   32   37
         | 42 / 47   52   57
   ________/_______________
   |  1     6    11    16     k=1 Первый лист
   | 21    26    31    36
   | 41    46    51    56
```

```
l=1
do i=1,nx
  do j=1,ny
    do k=1,nz
      a(i,j,k)=b(l)
      l=l+1
    enddo ...
```

5.4.3 Исходный текст главной программы test4

```
program test4; implicit none
interface
function myresh3(o,s,b); integer s(3),o(3); integer b(s(1)*s(2)*s(3))
                                integer myresh3(s(1),s(2),s(3))
end function myresh3
end interface
integer, parameter :: nx=3, ny=4, nz=5
character(39), parameter :: txt(0:1)=&
& (/'reshape                          myresh3',&
&   '                                        '/)
integer i, k, l
integer  b(nx*ny*nz)           !  b - вводимый одномерный векторт
integer  a(nx,ny,nz)           !  a - матрица, получаемая из b reshape
integer aa(nx,ny,nz)           ! aa - матрица, получаемая из и myresh3
integer s(3)                   !  s - вектор формы массивов a и aa.
integer o(3)                   !  o - вектор очерёдности заполнения a aa
read (*,*) b                                ! Ввод вектора b
write(*,*) ' b:'; write(*,'(20i3)') b       !                и его вывод.
read (*,*) o                                ! Ввод вектора очерёдности и
s=shape(a);                                 ! Определение формы массива aa.
write(*,'(a,i4,i12,i12)') ' вектор формы       s=', s ! Её  вывод
write(*,'(a,i4,i12,i12)') ' вектор очерёдности o=', o ! Его вывод.
!write(*,'(25x,25("="))')                    !
 a=reshape(b,(/s(1),s(2),s(3)/),order=o)  !  a из b через reshape
aa=myresh3(o,s,b)                           ! aa из b через myresh3
do k=1,nz;   write(*,'(i3,"-й лист:",7x,"reshape",25x,"myresh3")') k
  do i=1,nx; write(*,'(5x,4i7,4x,4i7)') a(i,:,k), aa(i,:,k)
  enddo
enddo
end program test4
```

5.4.4 Исходный текст функции myresh3

```
function myresh3(o,s,b) result(r)
implicit none
integer s(3),o(3)
integer b(s(1)*s(2)*s(3))
integer r(s(1),s(2),s(3))
integer l, i, j, k, s12, s23, s13
s12=s(1)*s(2); s23=s(2)*s(3); s13=s(1)*s(3)
do i=1,s(1)                          !
  do j=1,s(2)
    do k=1,s(3)
      if (o(1)==1.and.o(2)==2) l=i+(j-1)*s(1)+(k-1)*s(1)*s(2)
      if (o(1)==1.and.o(2)==3) l=i+(k-1)*s(1)+(j-1)*s(1)*s(3)
      if (o(1)==2.and.o(2)==1) l=j+(i-1)*s(2)+(k-1)*s(1)*s(2)
      if (o(1)==2.and.o(2)==3) l=j+(k-1)*s(2)+(i-1)*s(2)*s(3)
      if (o(1)==3.and.o(2)==1) l=k+(i-1)*s(3)+(j-1)*s(1)*s(3)
      if (o(1)==3.and.o(2)==2) l=k+(j-1)*s(3)+(i-1)*s(2)*s(3)
      r(i,j,k)=b(l)
enddo; enddo; enddo
end function myresh3
```

5.4.5 Результат тестирования myresh3 o=(/1,2,3/)

```
 b:
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

вектор формы       s=   3           4           5

вектор очерёдности o=   1           2           3

 1-й лист:       reshape                        myresh3

          1       4       7      10           1       4       7      10
          2       5       8      11           2       5       8      11
          3       6       9      12           3       6       9      12

 2-й лист:       reshape                        myresh3

         13      16      19      22          13      16      19      22
         14      17      20      23          14      17      20      23
         15      18      21      24          15      18      21      24

 3-й лист:       reshape                        myresh3

         25      28      31      34          25      28      31      34
         26      29      32      35          26      29      32      35
         27      30      33      36          27      30      33      36

 4-й лист:       reshape                        myresh3

         37      40      43      46          37      40      43      46
         38      41      44      47          38      41      44      47
         39      42      45      48          39      42      45      48

 5-й лист:       reshape                        myresh3

         49      52      55      58          49      52      55      58
         50      53      56      59          50      53      56      59
         51      54      57      60          51      54      57      60
```

5.4.6 Результат тестирования myresh3 o=(/2,1,3/)

```
 b:
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

вектор формы       s=   3           4           5

вектор очерёдности o=   2           1           3

 1-й лист:       reshape                       myresh3

          1       2       3       4           1       2       3       4
          5       6       7       8           5       6       7       8
          9      10      11      12           9      10      11      12

 2-й лист:       reshape                       myresh3

         13      14      15      16          13      14      15      16
         17      18      19      20          17      18      19      20
         21      22      23      24          21      22      23      24

 3-й лист:       reshape                       myresh3

         25      26      27      28          25      26      27      28
         29      30      31      32          29      30      31      32
         33      34      35      36          33      34      35      36

 4-й лист:       reshape                       myresh3

         37      38      39      40          37      38      39      40
         41      42      43      44          41      42      43      44
         45      46      47      48          45      46      47      48

 5-й лист:       reshape                       myresh3

         49      50      51      52          49      50      51      52
         53      54      55      56          53      54      55      56
         57      58      59      60          57      58      59      60
```

5.4.7 Результат тестирования myresh3 o=(/1,3,2/)

```
 b:
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

вектор формы       s=   3           4           5

вектор очерёдности o=   1           3           2

 1-й лист:       reshape                        myresh3

          1     16     31     46          1     16     31     46
          2     17     32     47          2     17     32     47
          3     18     33     48          3     18     33     48

 2-й лист:       reshape                        myresh3

          4     19     34     49          4     19     34     49
          5     20     35     50          5     20     35     50
          6     21     36     51          6     21     36     51

 3-й лист:       reshape                        myresh3

          7     22     37     52          7     22     37     52
          8     23     38     53          8     23     38     53
          9     24     39     54          9     24     39     54

 4-й лист:       reshape                        myresh3

         10     25     40     55         10     25     40     55
         11     26     41     56         11     26     41     56
         12     27     42     57         12     27     42     57

 5-й лист:       reshape                        myresh3

         13     28     43     58         13     28     43     58
         14     29     44     59         14     29     44     59
         15     30     45     60         15     30     45     60
```

5.4.8 Результат тестирования myresh3 o=(/3,1,2/)

```
 b:
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

вектор формы       s=   3           4           5

вектор очерёдности o=   3           1           2

 1-й лист:       reshape                        myresh3

          1     16     31     46          1     16     31     46
          6     21     36     51          6     21     36     51
         11     26     41     56         11     26     41     56

 2-й лист:       reshape                        myresh3

          2     17     32     47          2     17     32     47
          7     22     37     52          7     22     37     52
         12     27     42     57         12     27     42     57

 3-й лист:       reshape                        myresh3

          3     18     33     48          3     18     33     48
          8     23     38     53          8     23     38     53
         13     28     43     58         13     28     43     58

 4-й лист:       reshape                        myresh3

          4     19     34     49          4     19     34     49
          9     24     39     54          9     24     39     54
         14     29     44     59         14     29     44     59

 5-й лист:       reshape                        myresh3

          5     20     35     50          5     20     35     50
         10     25     40     55         10     25     40     55
         15     30     45     60         15     30     45     60
```

5.4.9 Результат тестирования myresh3 o=(/2,3,1/)

```
 b:
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

вектор формы       s=   3           4           5

вектор очерёдности o=   2           3           1

 1-й лист:       reshape                       myresh3

          1       2       3       4           1       2       3       4
         21      22      23      24          21      22      23      24
         41      42      43      44          41      42      43      44

 2-й лист:       reshape                       myresh3

          5       6       7       8           5       6       7       8
         25      26      27      28          25      26      27      28
         45      46      47      48          45      46      47      48

 3-й лист:       reshape                       myresh3

          9      10      11      12           9      10      11      12
         29      30      31      32          29      30      31      32
         49      50      51      52          49      50      51      52

 4-й лист:       reshape                       myresh3

         13      14      15      16          13      14      15      16
         33      34      35      36          33      34      35      36
         53      54      55      56          53      54      55      56

 5-й лист:       reshape                       myresh3

         17      18      19      20          17      18      19      20
         37      38      39      40          37      38      39      40
         57      58      59      60          57      58      59      60
```

5.4.10 Результат тестирования myresh3 o=(/3,2,1/)

```
 b:
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

вектор формы       s=   3           4           5

вектор очерёдности o=   3           2           1

 1-й лист:       reshape                        myresh3

          1      6      11     16          1      6      11     16
         21     26      31     36         21     26      31     36
         41     46      51     56         41     46      51     56

 2-й лист:       reshape                        myresh3

          2      7      12     17          2      7      12     17
         22     27      32     37         22     27      32     37
         42     47      52     57         42     47      52     57

 3-й лист:       reshape                        myresh3

          3      8      13     18          3      8      13     18
         23     28      33     38         23     28      33     38
         43     48      53     58         43     48      53     58

 4-й лист:       reshape                        myresh3

          4      9      14     19          4      9      14     19
         24     29      34     39         24     29      34     39
         44     49      54     59         44     49      54     59

 5-й лист:       reshape                        myresh3

          5     10      15     20          5     10      15     20
         25     30      35     40         25     30      35     40
         45     50      55     60         45     50      55     60
```

5.4.11 Возможный вариант функции myresh3

```
function myresh3a(o,s,b) result(r)
implicit none
integer s(3),o(3)
integer b(s(1)*s(2)*s(3))
integer r(s(1),s(2),s(3))
integer l, i, j, k, s12, s23, s13
integer :: key(1:3,1:3)=reshape((/0,3,5,1,0,6,2,4,0/),&
&                                               (/3,3/),&
&                                               (/1,2/) )
s12=s(1)*s(2)
s23=s(2)*s(3)
s13=s(1)*s(3)
do i=1,s(1)
  do j=1,s(2)
    do k=1,s(3)
      select case (key(o(1),o(2)))
        case(1); l=i+(j-1)*s(1)+(k-1)*s12
        case(2); l=i+(k-1)*s(1)+(j-1)*s13
        case(3); l=j+(i-1)*s(2)+(k-1)*s12
        case(4); l=j+(k-1)*s(2)+(i-1)*s23
        case(5); l=k+(i-1)*s(3)+(j-1)*s13
        case(6); l=k+(j-1)*s(3)+(i-1)*s23
      end select
      r(i,j,k)=b(l)
    enddo
  enddo
enddo
end function myresh3a
```

Список литературы

1. Горелик А.М. 2006. Программирование на современном Фортране. – М.: Финансы и статистика, – 352 с.

2. Бартеньев О.В. 2000. Современный ФОРТРАН. "3-е изд., доп. и перераб. – М.; ДИАЛОГ – МИФИ, – 448 с.

3. Рыжиков Ю.И. 2004. Современный ФОРТРАН: Учебник. – СПб.: КОРОНА принт, –288 с.

4. Немнюгин М.А., Стесик О.Л. 2004. Современный ФОРТРАН: Самоучитель. – СПб.: БХВ-Петербург, 496 с.

5. Немнюгин М.А., Стесик О.Л. 2008. ФОРТРАН в задачах и примерах многопроцессорных вычислительных систем. – СПб.: БХВ-Петербург, 320 с.

6. Игнатов В. 2000. Эффективное использование GNU Make.

7. Д.Кнут. 2017. - Искусство программирования для ЭВМ т. 1. Основные алгоритмы.

8. Numerical recipes in FORTRAN–77: The art of scientific computing (ISBN 0-521-43064-X) Copyright(C) 1986–1992 by Cambridge University Press.

9. Numerical recipes in FORTRAN–90: The art of scientific computing (ISBN 0-521-43064-X) Copyright(C) 1986–1992 by Cambridge University Press.

10. Numerical recipes in C: The art of scientific computing (ISBN 0-521-43108-5) Copyright(C) 1988–1992 by Cambridge University Press.

11. 1994. Лабораторный практикум по высшей математике: Учеб. пособие для втузов.–2-е изд., перераб. и доп.–М.: Высш. шк., –416 с.