Модуль № 2 «Ввод / вывод. Составные типы данных»

Тема № 1. Создание библиотек и работа с ними

На данном занятии необходимо познакомиться с созданием статических библиотек *Фортрана* и рассмотреть особенности работы с такими библиотеками. Кроме того, следует рассмотреть особенности форматного вывода.

Список заданий

Задание № 1

На основе ранее разработанных процедур вычисления производных, интегралов, решения нелинейных уравнений и обыкновенных дифференциальных уравнений создать статическую библиотеку пользователя.

На предыдущих занятиях мы создали программные модули для численного решения ряда задач:

- модуль NDeriv — методы численного дифференцирования функции одной переменной;
- модуль NIntegr — методы численного интегрирования функции одной переменной;
- модуль NSolve — методы численного решения уравнений одной переменной;
- модуль NDSolve — метод численного решения обыкновенного дифференциального уравнения одной переменной.

Модули были созданы, проверены, протестированы и могут быть использованы в различных наших проектах. Для упрощения дальнейшей работы на основе разработанных модулей создадим *статическую библиотеку* функций пользователя и проверим ее работу.

Процесс создания библиотеки показан в *Рекомендациях по выполнению* этого задания.

Проверить работу библиотеки, вычислив выражение

$$f'(x)\int_0^x \sin\left(\cos(t)^2\right)\ \mathrm{d}t\text{, где }\ f(x)=\sin(x)\exp(-x)$$

для $x\in[0,6]$ с заданным шагом. Результаты вычисления вывести на экран и сохранить в текстовый файл. По результатам расчетов построить график.

Задание № 2

С заданной точностью вычислить значение функции Бесселя по ее интегральному представлению.

$$\mathrm{J}_m(x)=\frac{1}{2\pi}\int_0^{2\pi}\cos\left(m\cdot t-x\cdot\sin(t)\right)\ \mathrm{d}t$$

Результат (значения x, $J_m(x)$) сохранить в текстовом файле. Значение для m получить из текстового файла, а начальное, конечное значения и шаг изменения x получить с клавиатуры. По результатам расчетов построить график. Сравнить полученные численные значения с результатом, вычисленным в *Mathematica*. Численные методы, необходимые для решения задачи взять из разработанной библиотеки. Замечания по решению задачи приведены в *Рекомендациях по выполнению*.

Задание № 3

Численным методом с заданной точностью найти первые два положительных корня уравнения $J'_m(x)=0$ методом секущих. Значение m получить из файла. Интервал локализации корней найти графическим методом. Результат расчетов вывести на экран. Сравнить полученные численные значения с результатом, вычисленным в *Mathematica*. Численные методы, необходимые для решения задачи взять из разработанной библиотеки. Замечания по решению задачи приведены в *Рекомендациях по выполнению*.

Задание № 4

Задано обыкновенное дифференциальное уравнение с начальным условием:

$$y'(x)=y(x)\cdot\cos(x+y(x)),\ y(0)=a.$$

1). В диапазоне $x\in[0,30]$ получить численное решение уравнения для нескольких различных значений параметра a $(a=0.1,\ 0.5,\ 3.0,\ 5.0)$. В текстовый файл с данными сохранить значения x, $y(x)$, $y'(x)$. Построить, оформить графики, обозначить кривые, экспортировать в PNG формате. Правильность вычислений проверить с помощью системы *Mathematica*.

2). С помощью численных методов найти второй положительный корень производной решения (x_{root}) при $a=0.5$ и вычислить $\int_0^{x_{\text{root}}} y(x)\ \mathrm{d}x$. Результат расчетов вывести на экран. Сравнить полученные численные значения с результатом, вычисленным в *Mathematica*.

Численные методы, необходимые для решения задачи взять из разработанной библиотеки. Замечания по решению задачи приведены в *Рекомендациях по выполнению*.

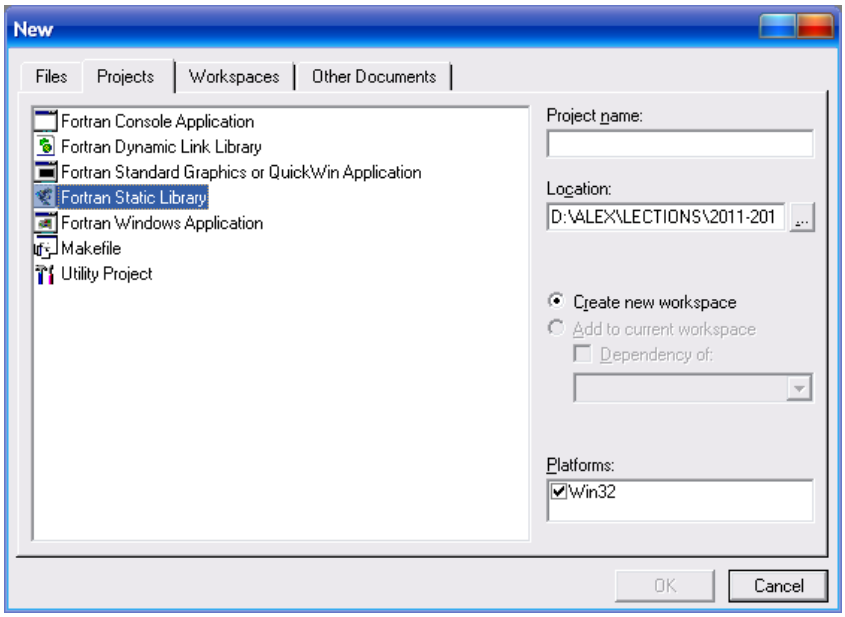
Рекомендации по выполнению

Задание № 1

Процесс создания и подключения к проекту библиотеки зависит используемой среды программирования. Рассмотрим две используемые на уроках среды: *Compaq Visual Fortran 6.1* и *PGI Visual Fortran 2010*.

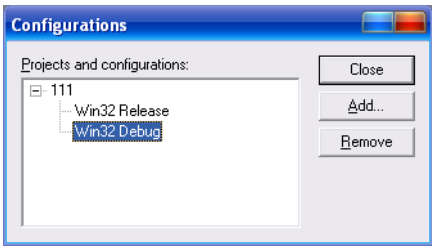
Компилятор *Compaq Visual Fortran 6.1*

Для создания библиотеки нужно создать новый проект особого вида. Выполнив команду *File | New* (комбинация клавиш ***Ctrl + N***) на экран будет выведено окно создания нового проекта. В данном окне следует выбрать требуемый тип проекта (*Fortran Static Library*), в текстовом поле *Project Name* указать имя проекта (имя будущей библиотеки, например NumLib), а в текстовом поле *Location* стандартным образом (окно выбора папки, вызываемое с помощью кнопки ..., расположенной справа от указанного поля) указать месторасположение проекта.



После того, как указана вся требуемая информация, следует нажать кнопку *OK* и перейти к следующему окну мастера создания нового проекта. Далее стандартным образом нужно закончить работу мастера проектов. В результате будет создан пустой проект, предназначенный для создания статической библиотеки. Далее следует добавить в проект исходные файлы. В папку проекта запишем (скопируем) созданные ранее файлы NDeriv.f90, NIntegr.f90, NSolve.f90 и NDSolve.f90. После этого стандартным образом добавим эти файл к нашему проекту (*Project | Add To Project | Files...*).

Будем делать Debug версию библиотеки. Для этого вначале нужно выбрать нужную конфигурацию проекта. Для этого вызовем команду меню *Build | Configurations...* . На экран будет выведено диалоговое окно выбора рабочей конфигурации проекта.



В данном окне нужно выбрать нужную конфигурацию (*Win32 Debug*). Далее откомпилируем (*Build | Compile*) и соберем (*Build | Build*) наш проект. Запускать его НЕ НАДО.

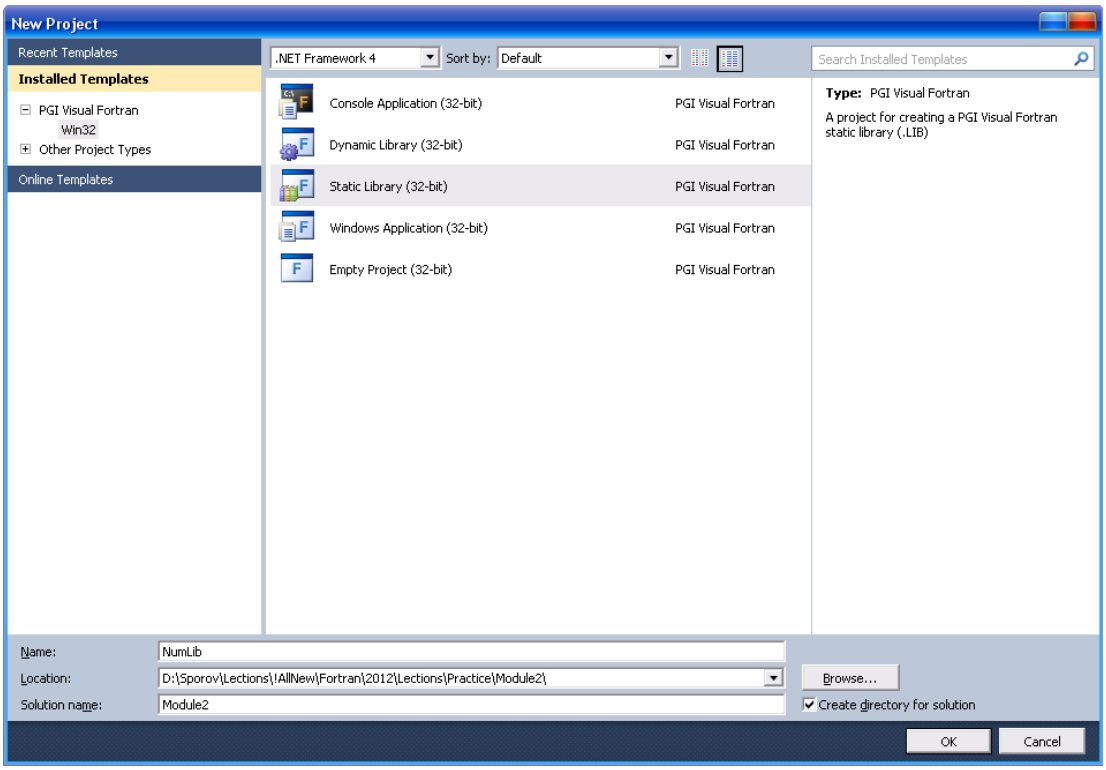
Таким образом, в папке проекта Debug будут находиться файл библиотеки (***.lib**) и файлы откомпилированных модулей (***.mod**). Если бы в исходных файлах с текстами процедур не содержались бы модули, то файлы откомпилированный модулей бы не создавались.

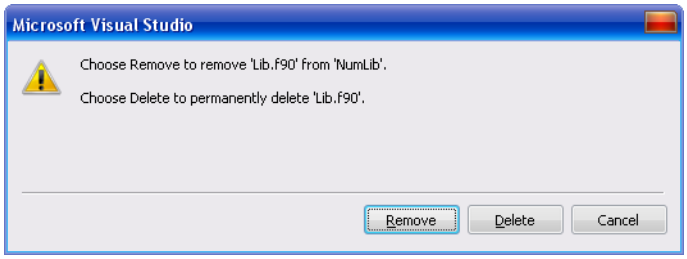
Для проверки работы библиотеки создадим новый консольный проект. Для того, чтобы подключить созданную библиотеку к этому проекту запишем файл библиотеки (***.lib**) и файлы откомпилированных модулей (***.mod**) в папку проекта и добавим эти файлы в проект (*Project | Add To Project | Files...*). После этого можно как обычно пользоваться функциями, находящимися в библиотеке.

Компилятор PGI Visual Fortran 2010

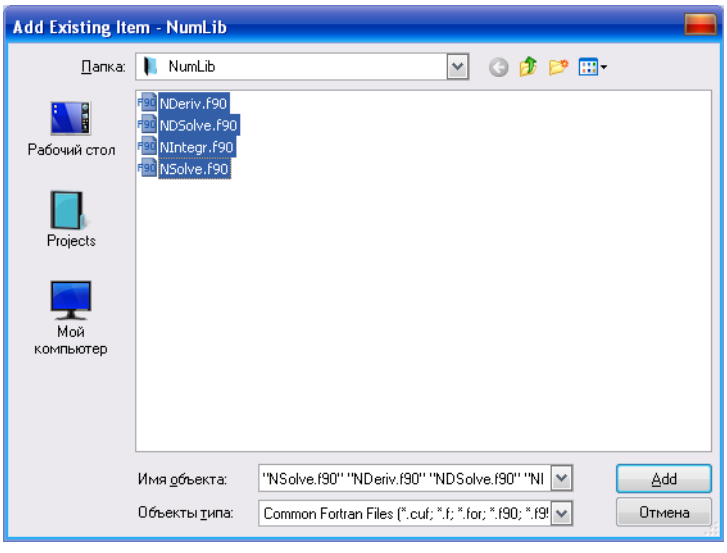
Для создания статической библиотеки нужно создать новый специальный проект. Команда меню *File | New | Project...* (комбинация клавиш ***Ctrl* + Shift + *N***) выведет на экран первое окно мастера создания нового проекта. В данном окне следует выбрать проходящий шаблон проекта (Static Library 32 bit), в текстовом поле *Name:* указать имя проекта (NumLib), в текстовом поле *Location:* стандартным образом (окно выбора папки, вызываемое с помощью кнопки Browse..., расположенной справа от указанного поля) указать месторасположение проекта, и в текстовом поле *Solution name:* указать имя решения (Module2). В данном случае имя решения будет отличаться от имени проекта, так как решение будет содержать много проектов.

После того, как нажата кнопка *OK*, будет создан требуемый проект, включающий пока только стандартный исходный файл (**Lib.f90**) с шаблоном подпрограммы. Удалим данный файл из проекта. Для этого выделим его мышью в окне *обозревателя решений* (*Solution Explorer*) воспользовавшись либо командой *Edit | Remove*, либо клавишей *Del*, либо командой *Remove* контекстного меню. На экран будет выведено диалоговое окно, в котором пользователя попросят уточнить свои действия. Нажатие кнопки *Remove* просто удалит выделенный файл из проекта, оставив его на диске. Кнопка *Delete* предназначена не только для удаления выбранного файла из проекта, но и для удаления его из файловой системы компьютера. Нажимаем кнопку *Delete* для удаления файла. Теперь проект пуст, и в него нужно добавить требуемые исходные файлы.





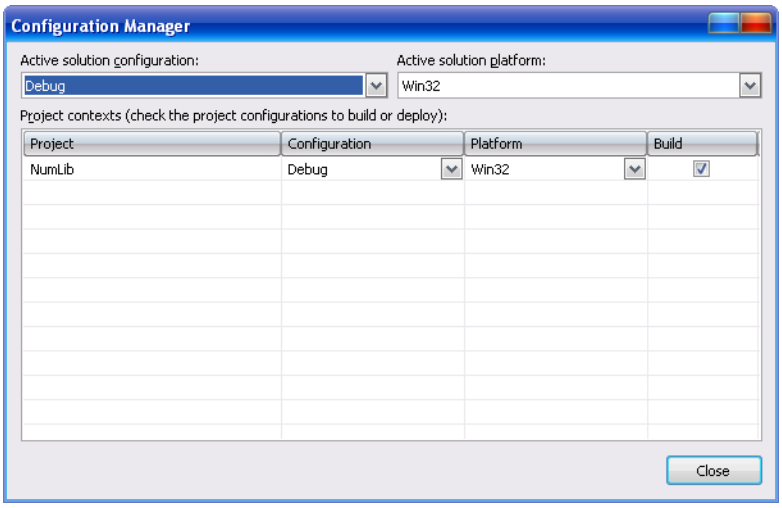
Для этого в папке решения Module2 находим папку нашего проекта (папка NumLib), куда запишем (скопируем) созданные ранее файлы NDeriv.f90, NIntegr.f90, NSolve.f90 и NDSolve.f90. После этого стандартным образом добавим эти файл к нашему проекту (*Project | Add Existing Item... (Shift + Alt + A)*).

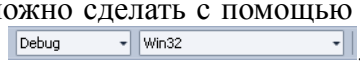


На экран будет выведено диалоговое окно *Add Existing Item*, предназначенное для добавления существующих файлов к проекту. Нужно выбрать файлы NDeriv.f90,

NIntegr.f90, NSolve.f90, NDSolve.f90 и нажать кнопку *OK*. Выбранные файлы будут добавлены к проекту.

Будем делать Debug версию библиотеки. Для этого вначале нужно выбрать нужную конфигурацию проекта. Для этого можно воспользоваться менеджером конфигураций, который вызывается командой меню *Build | Configuration Manager...* . На экран будет выведено диалоговое окно выбора рабочей конфигурации проекта.

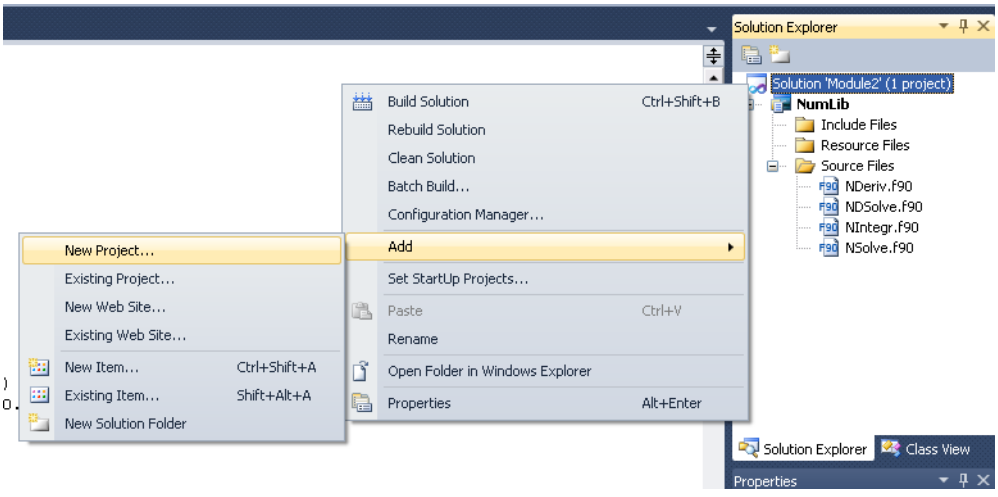


В качестве активной конфигурации укажем конфигурацию Debug (раскрывающийся список *Active solution configuration:*), а в качестве платформы — Win32 (раскрывающийся список *Active solution platform:*). После того, как все настройки выполнены, следует нажать кнопку *OK*. Те же настройки можно сделать с помощью панели инструментов, где расположены два аналогичных списка: 

. Далее можно сначала откомпилировать (*Build | Compile*), а можно сразу собрать проект (*Build | Build NumLib*). Запускать его НЕ НАДО.

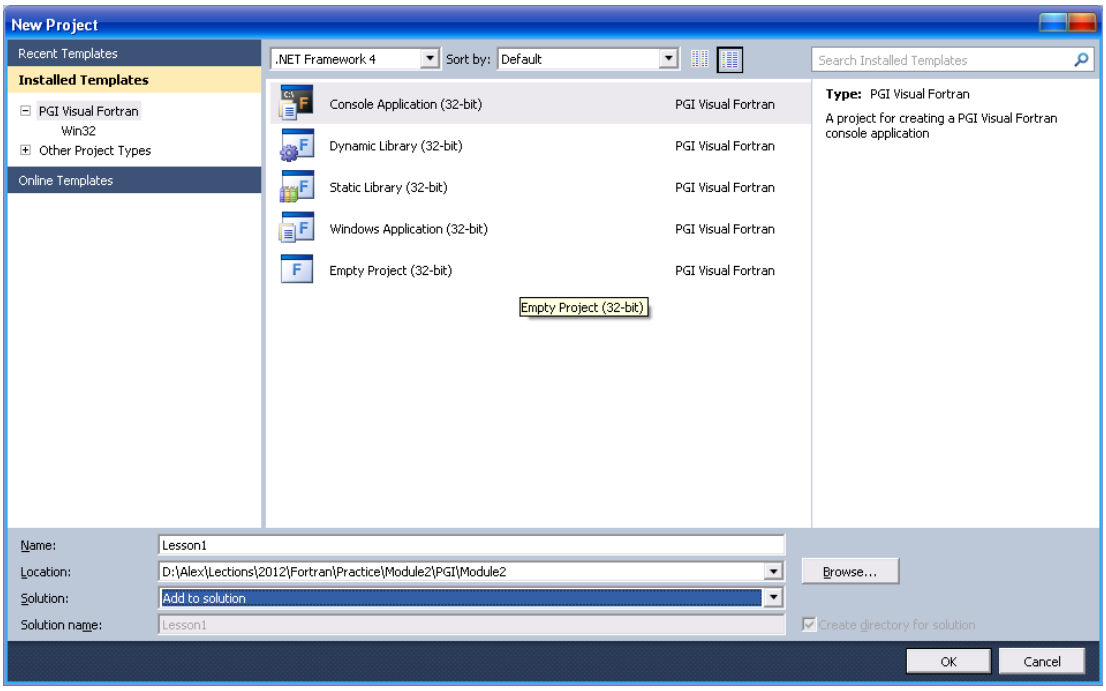
Таким образом, в папке проекта Win32 | Debug будут находиться файл библиотеки (**NumLib.lib**) и файлы откомпилированных модулей (***.mod**). Если бы в исходных файлах с текстами процедур не содержались бы модули, то файлы откомпилированный модулей бы не создавались.

Для проверки работы библиотеки в текущем решении создадим новый консольный проект. Для этого можно воспользоваться контекстным меню решения.



Для его вызова нужно выделить мышью узел решения в древовидной структуре, отображающей структуру решения в окне *Solution Explorer*, щелкнув правой кнопкой

мыши вызвать контекстное меню и в нем выбрать команду *Add | New Project...* . Добавить новый проект к решению можно и другим способом. Для этого воспользуемся командой меню *File | New | Project...* (комбинация клавиш ***Ctrl* + Shift + *N***). Так как данная команда была вызвана из существующего решения, то первое окно мастера создания нового проекта будет иметь немного другой вид.

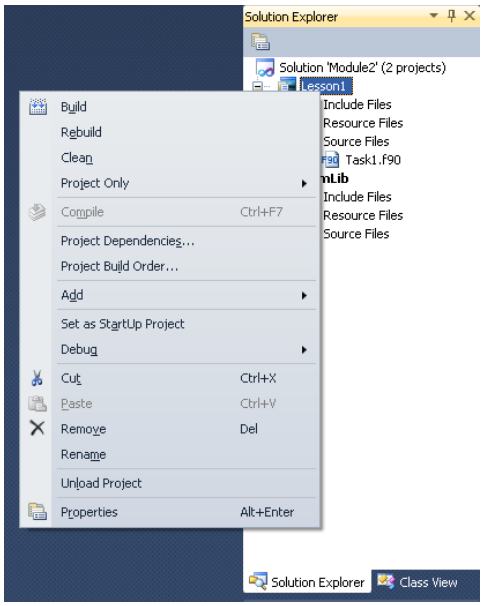


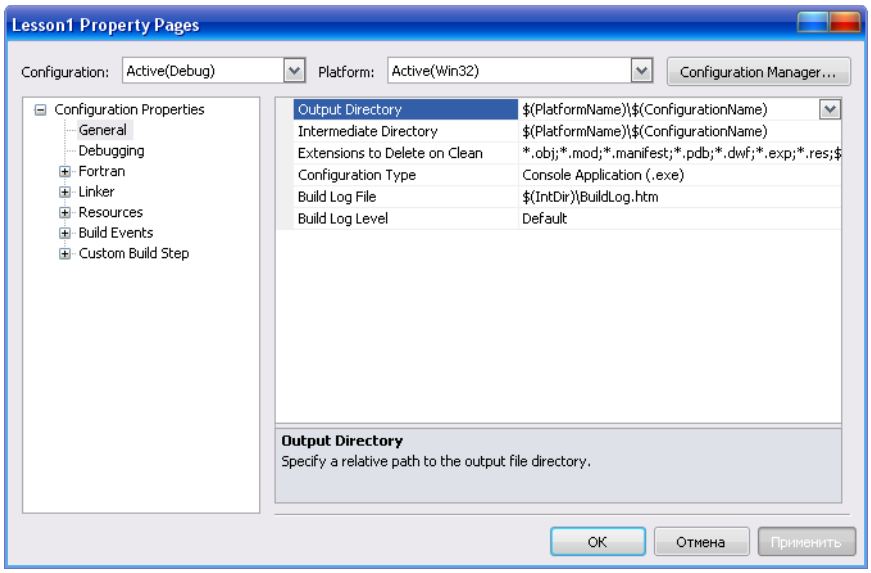
В данном окне следует выбрать тип нового проекта (Console Application (32-bit)), в текстовом поле *Name:* указать имя нового, создаваемого проекта (Lesson1), в поле *Location:* указать месторасположение нового проекта (по умолчанию в этом поле указан полный путь к текущему решению; лучше не менять это значение и тогда новый проект будет размещен в той же папке на жестком диске ПК, что и предыдущий), в новом поле *Solution:* указать, что новый проект нужно добавить к уже существующему решению (из раскрывающегося списка выбрать значение Add to solution). При этом поле *Solution name:* будет не активным. После того, как будет нажата кнопка *OK*, мастер создания нового проекта завершит свою работу и к текущему решению будет добавлен новый проект требуемого типа с указанным именем. Проект будет содержать один пустой исходный файл с именем по умолчанию (ConsoleApp.f90). С помощью контекстного меню переименуем его в Task1.f90.

Так как решение содержит уже несколько проектов, то нужно указать среде разработки какой проект является «запускающим». Для этого вызываем контекстное меню проекта Lesson1 и выбираем команду *Set as StartUp Project*. В результате проектом, который с которого начнется выполнение программы будет проект Lesson1. Название проекта в окне *Solution Explorer* будет выделено полужирным шрифтом.

Для того, чтобы можно было пользоваться только что созданной библиотекой, наш новый проект нужно настроить. Для этого вызываем контекстное меню нашего нового проекта и в нем выбирает команду *Properties*. На экран будет выведено диалоговое окно со всеми свойствами проекта. В данном окне нужно указать компилятору где ему можно будет найти файлы откомпилированных модулей (***.mod**), входящих в библиотеку; указать редуктору связей где в файловой системе ПК расположен файл библиотеки (**NumLib.lib**) и какую собственно библиотеку из этой папки нужно использовать. Кроме того, очень удобно указать существующие зависимости между проектами, что бы при любых изменениях в

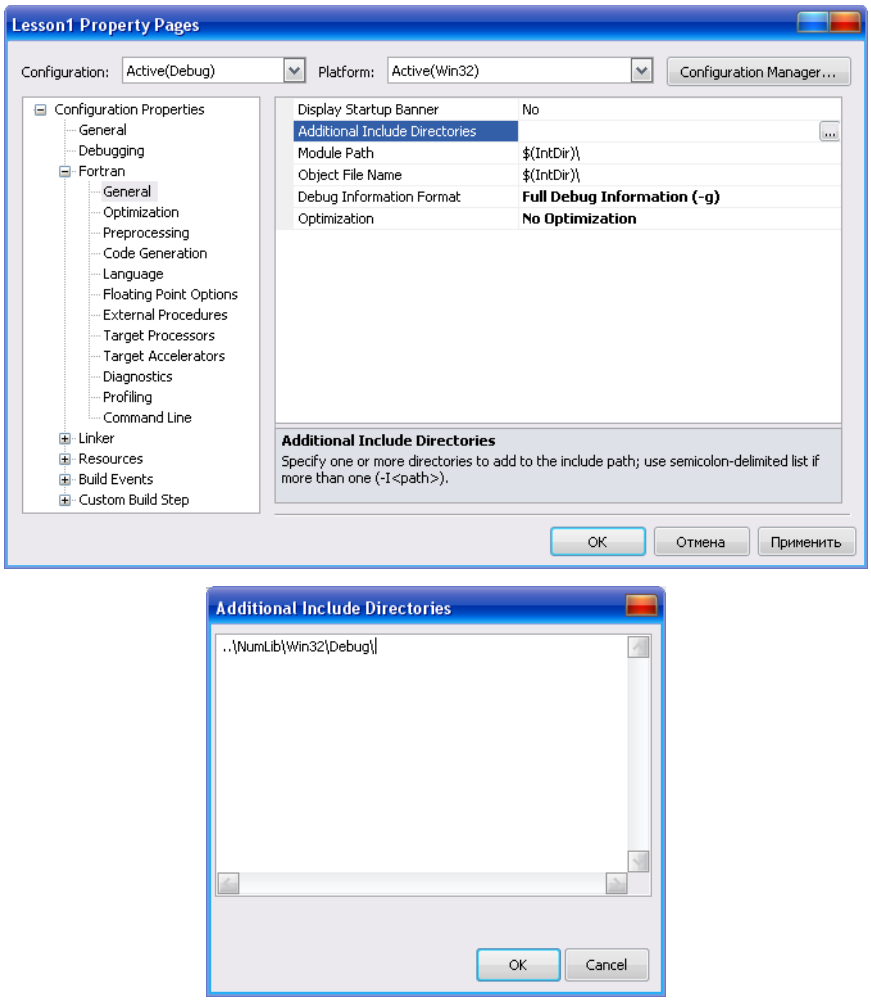
исходных файлах входящих в библиотеку процедур сначала компилировалась и собиралась библиотека, а уже потом с обновленной библиотекой собирался новый консольный проект.



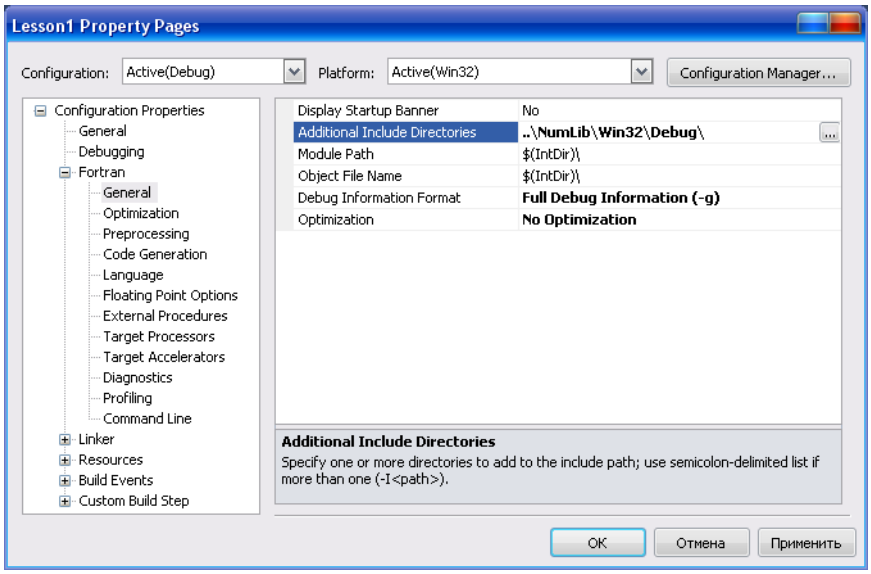


В данном окне (*Lesson1 Property Pages*) нужно указать дополнительные настройки в секциях *Fortran* (настройки компилятора) и *Linker* (настройки редактора связей). Вначале укажем дополнительные настройки компилятора.

Для этого раскроем секцию настроек компилятора *Fortran*. В данной секции нужно перейти в раздел *General*. В данном разделе есть текстовое поле *Additional Include Directories*, в котором нужно указать путь к файлам откомпилированных модулей (***.mod**). Для этого нужно установить текстовый курсор в данное поле и либо сразу ввести нужный путь, либо нажать на кнопку , расположенную справа от текстового поля и в выведенном на окне диалоговом окне ввести нужный путь.

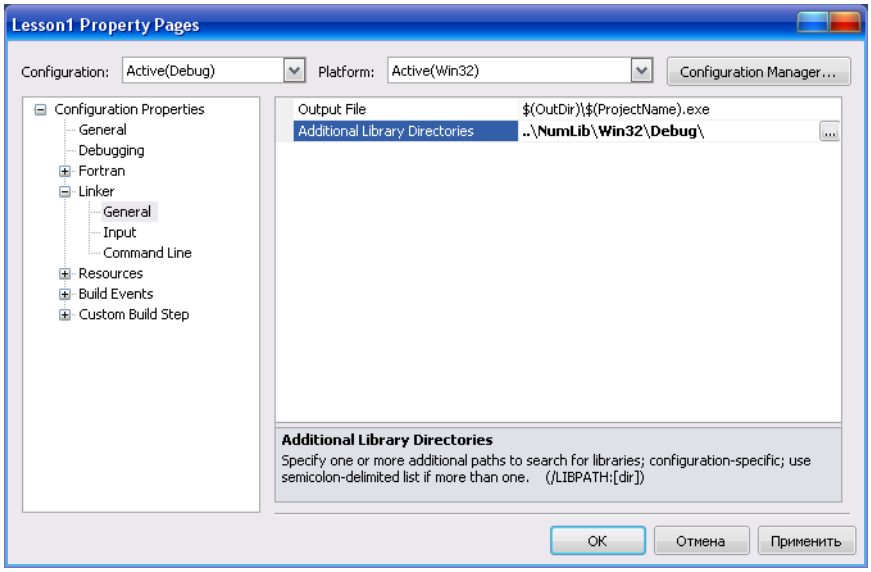


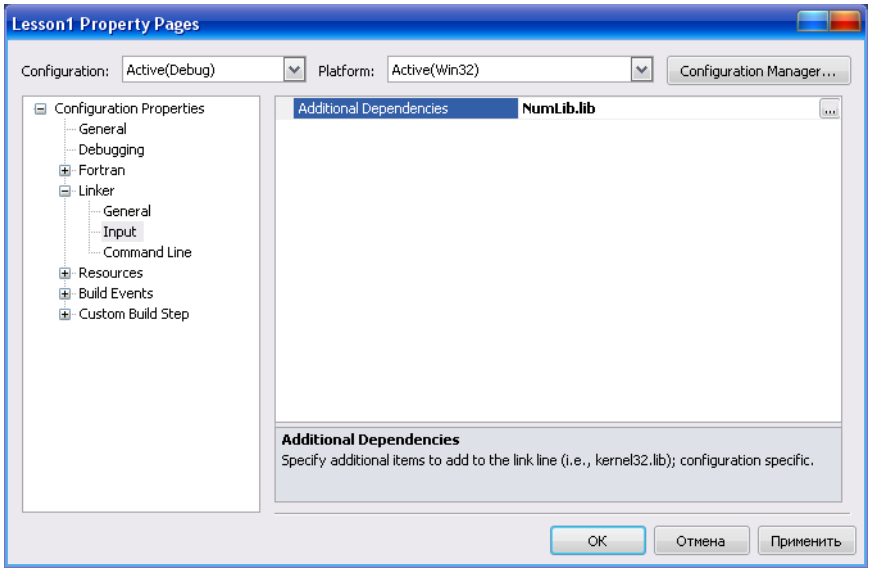
В данном окне можно указать абсолютный (полный) путь к папке, где лежат **mod**–файлы, но лучше указать относительный. Тогда проект можно будет без перенастройки размещать в других папках и переносить на другие компьютеры. В результате диалоговое окно со свойствами проекта будет выглядеть так:



Далее укажем дополнительные настройки редактора связей. Для этого нужно перейти в секцию *Linker*. В данной секции нужно сначала перейти в раздел *General*. В данном

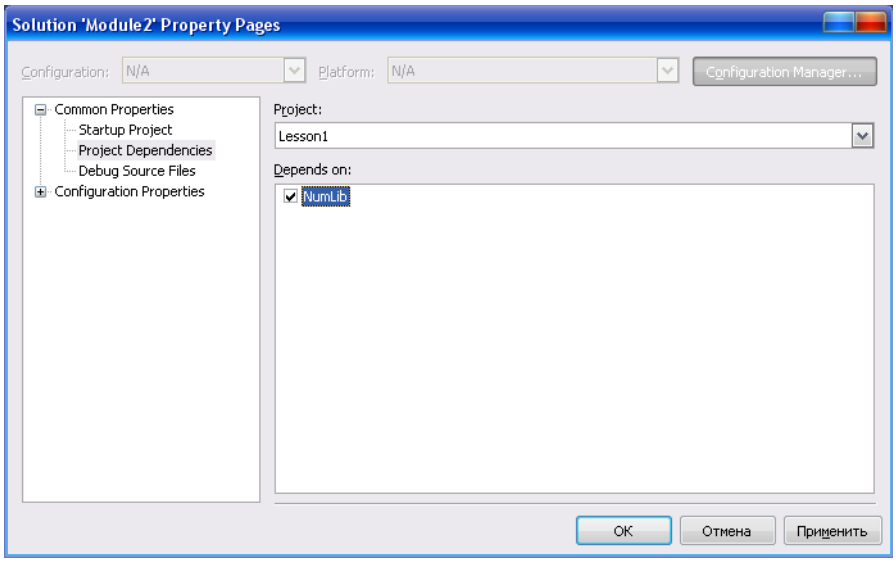
разделе есть текстовое поле *Additional Library Directories*, в котором нужно указать путь к файлам дополнительных библиотек (***.lib**), которые нужно подключить к проекту. Делаем все те же действия, с помощью которых указывали дополнительные настройки компилятора.





Далее нужно указать, какую библиотеку, расположенную в ранее указанном месте, нужно подключить к проекту. Для этого нужно выбрать раздел *Input*, в расположенный в секцию *Linker*. В данном разделе в текстовом поле *Additional Dependencies* нужно указать имя файла библиотеки, которую нужно подключить к проекту (в нашем случае NumLib.lib).

Для удобства дальнейшей работы нужно указать существующие зависимости между проектами (в нашем случае проект Lesson1 зависит от проекта NumLib), что бы при любых изменениях в исходных файлах входящих в библиотеку процедур сначала компилировалась и собиралась библиотека, а уже потом с обновленной библиотекой собирался новый консольный проект. Для этого с помощью контекстного меню нужно вызвать диалоговое окно свойств решения Module2. В этом окне нужно перейти в секцию *Common Properties* и в разделе *Project Dependencies* указать существующие зависимости между проектами. Диалоговое окно Solution 'Module2' Property Pages примет указанный ниже вид.



Исходный код проверочной программы

Для удобства проверки все программные единицы (и модуль *Фортрана*, и головную программу) разместим в одном файле. Необходимо только помнить о том, что если в одном файле указаны и модуль, и процедура, использующая этот модуль, то исходный код модуля должен предшествовать исходному коду процедуры.

Файл **Task1.f90**.

```
module TestFuncts

  private :: f, g
  public testFun

contains

! Функция для дифференцирования
function f(x)

  implicit none

  real(kind=8) :: f
  real(kind=8), intent(in) :: x

  f = sin(x) * exp(-x)

end function f

! Функция для интегрирования
function g(x)

  implicit none

  real(kind=8) :: g
  real(kind=8), intent(in) :: x
```

```
  g = sin(cos(x) ** 2)

end function g

! Функция для тестирования
function testFun(x)

  use NDeriv
  use NIntegr

  implicit none

  real(kind=8) :: testFun
  real(kind=8), intent(in) :: x

  real(kind=8) :: der, int

  der = D(f, x, method=der3ptCenter)
  int = NIntegrate(g, 0.0d0, x)

  testFun = der*int

end function testFun

end module TestFuncts


! Основная программа
program TestLibrary

  use TestFuncts

  implicit none

  real(kind=8) :: xBeg, xEnd, xStep
  real(kind=8) :: res
  integer(kind=4) :: ioStatus

  call getData()

  open(11, file="Result.dat", iostat=ioStatus)
  if (ioStatus /= 0) stop "ERROR while opening file 'Result.dat'..."
  write(11, '(a5, 9x, a6)') "X", "Y"

  do while(xBeg <= xEnd)
    res = testFun(xBeg)
```

```
    print 100, xBeg, res
    write(11, '(e10.4, 4x, e12.6)') xBeg, res
    xBeg = xBeg + xStep
  end do

  close(11)

  100 format(7x, "x = ", e10.4, 4x, "f(x) = ", e12.6)

contains

  subroutine getData()

    xBeg = 0.0d0; xEnd = 6.0d0;

    do
      print '(1x,a,$)', "Type X step value: "
      read *, xStep
      if ((xStep > 0.0d0) .and. (xStep <= 5.0d-1)) exit
    end do

    print "(3x, '1). start: ', e10.4)", xBeg
    print "(3x, '2).  stop: ', e10.4)", xEnd
    print "(3x, '3).  step: ', e10.4)", xStep

    pause 'Press <Enter> to continue the program...'

    xEnd = xEnd + 1.0d-1 * xStep

  end subroutine getData

end program TestLibrary
```

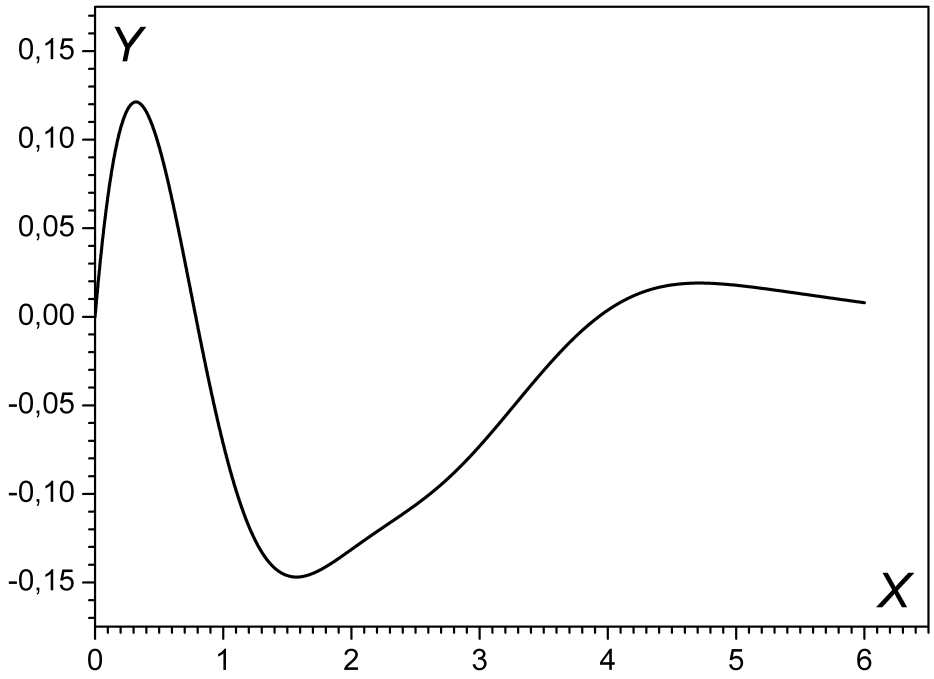
Тестовые результаты

X:	0.0	0.5	0.1	0.15
Y:	0.0	0.0958917	−0.0719088	−0.146121

X:	2.0	2.5	3.0	3.5
Y:	−0.131322	−0.106209	−0.0729194	−0.0301814

X:	4.0	4.5	5.0	5.5
Y:	0.00380313	0.0180324	0.0178168	0.0130693

X:	6.0
Y:	0.00796686



Задание № 2

Для вычисления указанной функции Бесселя в системе *Mathematica* можно воспользоваться функцией BesselJ[m, x].

Для удобства организации программы исключим с сохранением на диске файлы с исходным кодом программы из проекта Lesson1 и в этом проекте создадим новые файлы с исходным кодом программ (Task2.f90 — основная программа; Params.f90 — получение значения m из файла данных; BesselJ.f90 — собственно вычисление функции Бесселя).

При решении данной задачи возникает проблема согласования интерфейсов: библиотечная функция может интегрировать функцию одного вещественного аргумента, а у нас есть еще один параметр — значение m. Для того, чтобы передать значение этого параметра в функцию модуль — получается что–то типа глобальной переменной, но с контролем доступа.

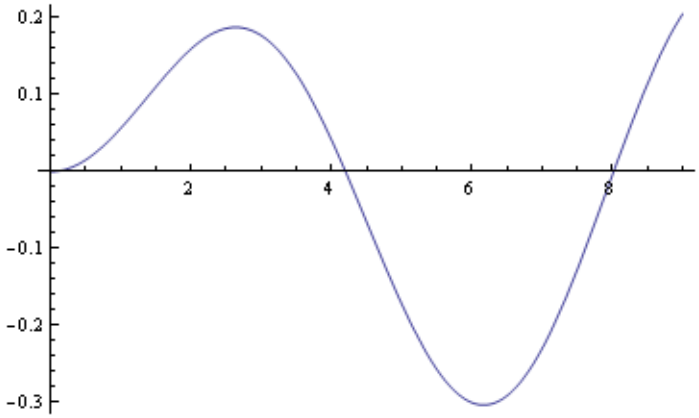
Задание № 3

Для вычисления производной в системе *Mathematica* можно воспользоваться как встроенной функцией D, так и функцией Derivative. Для численного решения уравнения с заданным начальным приближением воспользоваться функцией FindRoot. Для построения графика функции применить функцию Plot. Примерный вид документа *Mathematica* для проверки численного счета приведен на рисунке:

```
m = 3;

f[x_] := Derivative[1][BesselJ[m, #] &][x]

f[x]
```

$$\frac{1}{2}\,(\text{BesselJ}[2, x] - \text{BesselJ}[4, x])$$

```
Plot[f[x], {x, 0, 9}]
```




```
FindRoot[f[x], {x, 4}][[1]]
```

$x \to 4.20119$

```
FindRoot[f[x], {x, 8}][[1]]
```

$x \to 8.01524$

Задание № 4

По результатам расчетов графики можно построить как с помощью системы *GnuPlot*, так и с помощью системы *Origin*.

Для численного решения обыкновенных дифференциальных уравнений в системе *Mathematica* можно воспользоваться как встроенной функцией NDSolve. Для численного решения уравнения с заданным начальным приближением нужно воспользоваться функцией FindRoot. Для численного вычисления интеграла можно использовать функцию NIntegrate. Примерный вид документа *Mathematica* для проверки численного счета (без *Output* ячеек) приведен на рисунке:

Task3

- 1)

```
In[1]:= fun[xArg_, a_] := Module[{ode, init, y, x},
          ode = y'[x] == y[x] Cos[x + y[x]];
          init = y[0] == a;
          y[xArg] /. NDSolve[{ode, init}, y, {x, 0, 30}][[1, 1]]]

In[2]:= der[xArg_, a_] := Module[{ode, init, y, x},
          ode = y'[x] == y[x] Cos[x + y[x]];
          init = y[0] == a;
          y[xArg] Cos[xArg + y[xArg]] /.
           NDSolve[{ode, init}, y, {x, 0, 30}][[1, 1]]]

In[3]:= Plot[{fun[x, 0.1], fun[x, 0.5], fun[x, 3.0], fun[x, 5.0]},
         {x, 0, 30}, PlotRange → All]

In[4]:= Plot[{der[x, 0.1], der[x, 0.5], der[x, 3.0], der[x, 5.0]},
         {x, 0, 30}, PlotRange → All]

In[5]:= TableForm[Table[{x, fun[x, 0.5], der[x, 0.5]}, {x, 0, 30, 1}],
         TableHeadings → {Automatic, {"x", "y(x)", "y'(x)"}}]
```

- 2)

```
In[8]:= Plot[der[x, 0.5], {x, 0, 5}, PlotRange → All]

In[10]:= res = FindRoot[der[x, 0.5], {x, 4, 5}][[1]]

In[12]:= Chop[der[x, 0.5] /. res]

In[16]:= Plot[fun[x, 0.5], {x, 0, x /. res}, PlotRange → All]

In[14]:= NIntegrate[fun[x, 0.5], {x, 0, x /. res}]
```

Тема № 2. Файловый и форматный ввод / вывод

На данном занятии необходимо рассмотреть основные возможности, которые предоставляет программисту язык программирования *Фортран* по работе с файлами

Список заданий

Задание № 1

Напишите программу, которая проверяет, существует ли в рабочей папке программы текстовый файл **Task1.dat**. Если файл не существует, то программа создает его и записывает туда n пар псевдослучайных вещественных чисел (x, y) из диапазона $x \in [a, b]$ $y \in [c, d]$. Если же этот файл уже существует, то программа дописывает в него m пар таких чисел. Величины n (m), a, b, c, d вводятся в программу из тестового файла **Task1.ini** и во время работы программы выводятся на экран.

Замечание: для решения этого и всех следующих заданий удобно создавать по два файла: один файл с текстом основной программы и другой файл с модулем, содержащим основные процедуры, необходимые для решения задания.

Пример организации программы для компилятора *PGI Fortran 2010* приведен в рекомендациях по выполнению заданий.

Задание № 2

Напишите программу, которая позволяет просматривать текстовые файлы (поэкранно выводит содержимое файла), созданные в **Задании № 1**.

Вид экрана во время работы программы показан в рекомендациях по выполнению заданий.

Задание № 3

Напишите программу, которая проверяет, существует ли в рабочей папке программы текстовый файл **Task3.dat**. Если файла нет, то программа должна создать его и заполнить двумя столбцами вещественных чисел x, $f(x)$, где x — псевдослучайное число из диапазона $x \in [-5, 5]$, а $f(x) = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot \exp\left(-\frac{(x-\mu)^2}{2 \cdot \sigma^2}\right)$. Если указанный файл существует, то программа должна вычислить количество строк с данными в этом файле и среднее арифметическое значение функции $f(x)$, а также вывести результат на экран и удалить файл с данными.

Задание № 4

Напишите программу, которая проверяет, существует ли в рабочей папке программы текстовый файл **Task4.dat**. Если файла нет, то программа должна создать его и заполнить псевдослучайными вещественными числами ξ $(\xi \in [a, b])$, записанными в один столбец. Параметры a и b вводятся в программу из тестового файла **Task4.ini** и во время работы программы выводятся на экран. Если указанный файл существует, то программа должна вычислить среднее арифметическое вещественных чисел, содержащихся в файле, и записать в исходном порядке числа из файла **Task4.dat**, большие среднего в файл **Task4_1.dat**, а остальные — в файл **Task4_2.dat**.

Задание № 5

Напишите программу, которая проверяет, существует ли в рабочей папке программы текстовый файл **Task5.dat**. Если указанного файла нет, то программа должна создать его и заполнить псевдослучайными вещественными числами, записанными в один столбец. Если указанный файл существует, то программа должна найти в этом файле второе по величине число.

Задание № 6

Напишите программу, которая проверяет, существует ли в рабочей папке программы текстовый файл **Task6.txt**. В этом файле должны хранятся строки длиной не более 80 символов. Если указанного файла нет, то программа должна создать его и заполнить случайным количеством строк, содержащих случайное количество случайных символов (можно просто цифр). Если указанный файл существует, то программа должна создать новый файл **Task6Proc.txt** содержащий всю информацию из исходного файла без строки с заданным во время работы программы номером. Если строки с указанным номером нет, то программа должна сообщить об этом и новый файл создаваться не должен.

Задание № 7

Напишите программу, которая проверяет, существует ли в рабочей папке программы текстовый файл **Task7.dat**. В этом файле должны хранятся строки длиной не более 100 символов. Если указанного файла нет, то программа должна создать его и заполнить случайным количеством строк, содержащих случайное количество случайных символов (можно просто цифр). Если указанный файл существует, то программа должна создать новый файл **Task7_proc.dat**, в который будет записано первые две строки исходного файла, каждая n -я и последняя строка.

Задание № 8

Даны имена двух файлов (**Task8_1.dat** и **Task8_2.dat**). Известно что один из них (либо первый, либо второй) существует и содержит некоторое количество вещественных чисел, записанных в один столбец, а другой отсутствует. Программа должна проверять существует ли один из указанных файлов, и если ни одного, ни другого файла нет, то должна случайным образом создавать один из них. Если один из указанных файлов существует, то программа должна создать отсутствующий файл и записать в него конечный и начальный элементы существующего файла (в указанном порядке).

Задание № 9

Даны имена двух форматных текстовых файлов (**Task9_1.dat** и **Task9_2.dat**), заполненных целыми числами, которые записаны в один столбец. Программа должна проверять существуют ли указанные файлы и если какого-то файла нет, то должна создавать требуемый, заполняя его случайным количеством псевдослучайных целых чисел. Если оба файла существуют, то программа должна сравнить содержимое файлов и записать в новый файл **Task9_3.dat** числа которые есть во втором файле, но не встречались в первом. Если таких чисел нет, то новый файл создаваться не должен.

Задание № 10

Напишите программу, которая проверяет, существует ли в рабочей папке программы текстовый файл **Task10.dat**. Если этого файла нет, то программа должна создать его и

сохранить в этом файле результат табуляции функции $\sin(x)$ в заданном интервале с заданным шагом. Файл должен иметь следующий вид: первая строка — заголовки двух столбцов: аргумент (x), значение функции ($\sin(x)$). Следующие строки — соответствующие числовые данные. Интервал и шаг табуляции задаются пользователем с клавиатуры. Если указанный файл на диске уже существует, то программа должна дописать в него такое же количество числовых строк, сколько их уже есть в файле. Табуляция должна продолжаться с тем же шагом, что и в исходном файле.

Задание № 11

Задан зашумленный набор экспериментальных данных (x, y), которые располагаются на графике примерно вдоль прямой линии. Требуется найти уравнение прямой $y = k \cdot x + b$, которая наилучшим образом приближает результаты. По полученным коэффициентам построить прямую. Отобразить все на графике.

Основные формулы и результаты приведены в рекомендациях по выполнению заданий.

Рекомендации по выполнению

Задание № 1

Пример организации программы для компилятора *PGI Fortran 2010* (в тексте модуля присутствуют операторы, которые поддерживаются только компиляторами стандарта *Fortran 2003*).

Файл Task1.f90

```
program Task1

  use Task1mod

  implicit none

  if (isFileExists()) then
    print *, 'The adding data to existing file...'
    call prossessFile(1)
  else
    print *, 'The file creation...'
    call prossessFile(0)
  end if

end program Task1
```

Файл Task1mod.f90

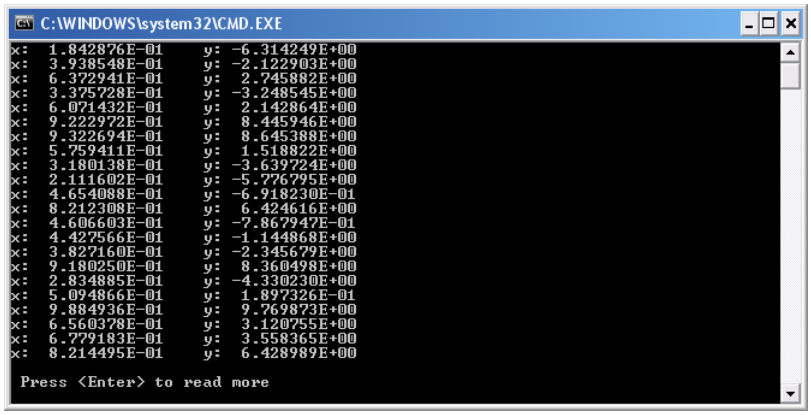
```
module Task1mod

  character(len=9), save :: fileName = "Task1.dat"

  real(kind=4), save :: a, b, c, d
  integer(kind=4), save :: num

  public isFileExists, prossessFile
  private readIniData

contains

  ! ==========================================
  function isFileExists()

    implicit none

    logical(kind=4) :: isFileExists

    inquire(file=fileName, exist = isFileExists)

  end function isFileExists
```

```
! ============================================
subroutine readIniData()

  implicit none

  integer(kind=4) :: status
  character(kind=1, len=80) :: msg

  open(22, file="Task1.ini", status='old', action='read', &
         iostat=status, iomsg=msg)
  if (status /= 0) then
    print *, '->', trim(msg), '<-'
    stop "'Task1.ini' file opening error..."
  end if

  read(22, *) num
  read(22, *) a;  read(22, *) b
  read(22, *) c;  read(22, *) d

  close(22, status='keep')

end subroutine readIniData

! ============================================
subroutine prossessFile(code)

  implicit none

  integer(kind=4), intent(in) :: code
  integer(kind=4) :: i, status
  real(kind=4) :: x, y

  call readIniData()

  select case(code)
    case(0)
      open(11, file=fileName, status='new', action='write', &
              position='rewind', iostat=status)
      if (status /= 0) stop "'Task1.dat' file opening error..."
    case(1)
      open(11, file=fileName, status='old', action='write', &
              position='append', iostat=status)
      if (status /= 0) stop "'Task1.dat' file reopening error..."
    case default
      stop 'Error!!! Wrong code of file operation...'
  end select
```

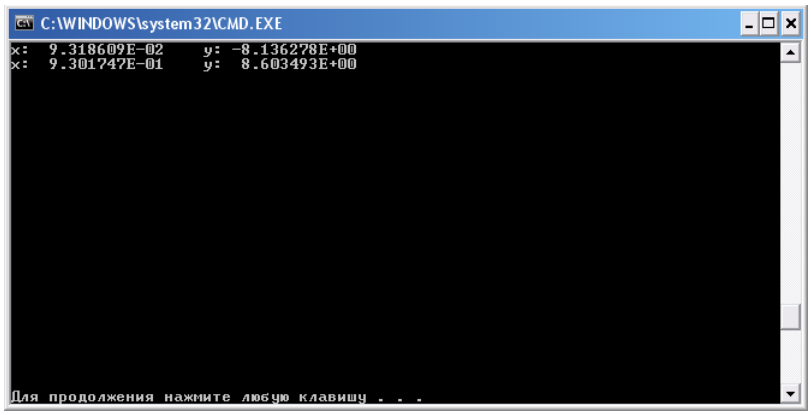
```
    call random_seed()
    do i = 1, num
        call random_number(x)
        x = a + (b - a) * x
        call random_number(x)
        y = c + (d - c) * x
        write(11, 100) x, y
    end do

    close(11, status='keep')

    100 format(es13.6, 4x, es13.6)

  end subroutine prossessFile

end module Task1mod
```

Задание № 2

Рекомендованный вид экрана во время работы программы:
Начальный и следующие экраны (до предпоследнего):



Последний экран вывода:



Задание № 11

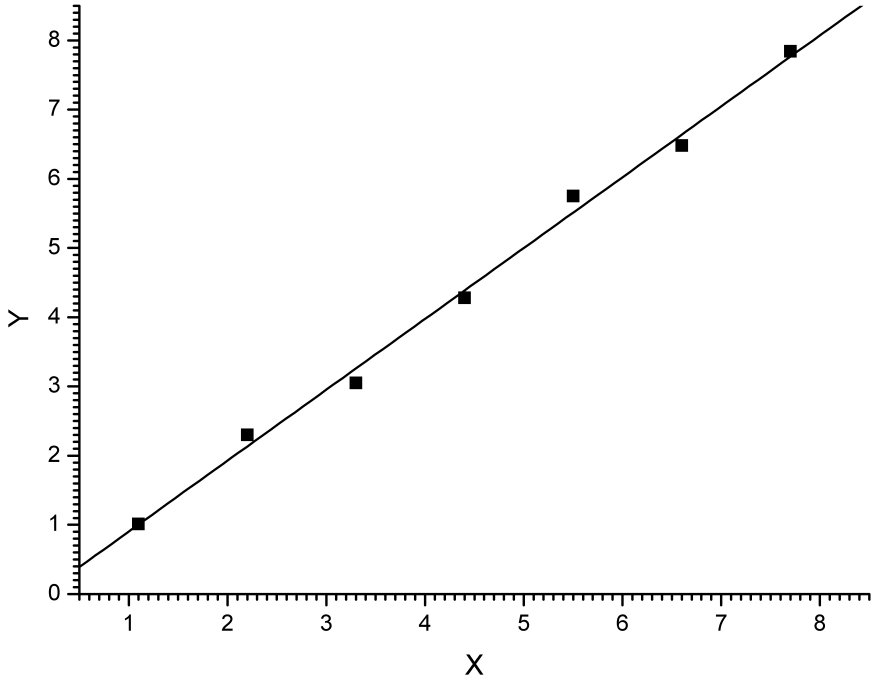
Для проверки правильности работы программы использовать такие данные:

x	y
1.1	1.01
2.2	2.30
3.3	3.05
4.4	4.28
5.5	5.75
6.6	6.48
7.7	7.84

Самый простой и наиболее часто используемый вид регрессии — линейная. Приближение данных (x_i, y_i) осуществляется линейной функцией $y(x)=k\cdot x+b$. Если определить коэффициенты регрессии k и b, то для любого значения x можно предсказать значение y, просто проведя вычисления по формуле. Стандартный метод получения коэффициентов k и b — это метод наименьших квадратов. Метод получил такое название, поскольку коэффициенты k и b вычисляются из условия минимизации суммы квадратов ошибок $|b+k\cdot x_i-y_i|^2$. Расчетные формулы можно записать так:

$$k=\frac{\left(\sum x\cdot y\right)-\left(\sum x\right)\cdot\bar{y}}{\left(\sum x^2\right)-\left(\sum x\right)\cdot\bar{x}},\quad b=\bar{y}-k\,\bar{x}$$

здесь $\bar{x}$, $\bar{y}$ — средние значения соответствующих величин.



Тема № 3 Символы и строки

На данном занятии необходимо рассмотреть основные возможности, которые предоставляет программисту язык программирования *Фортран* по работе со строками

Список заданий

Задание № 1

Напишите программу, которая запрашивает у пользователя имя файла. Если файл с указанным именем существует в рабочей папке программы, то программа определяет, сколько в нем прописных и строчных букв латинского алфавита. После этого программа преобразует файл: если прописных букв в файле больше, чем строчных, то все буквы преобразуются в прописные, и наоборот, если в файле больше строчных букв, чем прописных, то все буквы преобразуются в строчные. Если же строчных и прописных букв в файле поровну, то текстовый файл не преобразуется.

Замечание: для решения этого и всех следующих заданий удобно создавать по два файла: один файл с текстом основной программы и другой файл с модулем, содержащим основные процедуры, необходимые для решения задания.

Пример организации программы для компилятора *PGI Fortran 2010* приведен в рекомендациях по выполнению заданий.

Задание № 2

Напишите модуль, содержащий функцию, предназначенную для преобразования строк с русскими буквами из *Windows* в *DOS* представление. Напишите программу для проверки этого модуля.

Задание № 3

Напишите программу, которая преобразует натуральное число, записанное в римской системе счисления, в обычную арабскую систему счисления. Необходимая информация по римской системе счисления приведена в рекомендациях по выполнению заданий.

Задание № 4

Проверить, является ли заданная пользователем с клавиатуры строка палиндромом, .т е. одинаково ли читается как слева направо, так и справа налево.

Задание № 5

Напишите модуль, содержащий функцию, предназначенную для замены текущего расширения имени файла на **.BAK**. Следует учесть, что в имени файла может быть несколько точек, а также то, что файл может вообще не иметь расширения имени. Напишите программу для проверки этого модуля.

Задание № 6

Напишите программу, которая сначала преобразовывает заданные числа *a* и *b* в символьное представление, а затем выполняет обратное преобразование «строка — число».

Задание № 7

Напишите программу, которая проверяет, одинаковое ли число открывающихся и закрывающихся круглых скобок в данной строке и сбалансированы ли они.

Задание № 8

Напишите модуль, содержащий функцию, которая по словам выводит содержимое строки на экран. Эта функция должна принимать параметр, определяющий что считать разделителем между словами. Напишите программу для проверки этого модуля.

Задание № 9

Напишите модуль, содержащий функции, предназначенные для шифрования и расшифровывания текста на английском языке. Шифрование осуществляется с помощью ключа, содержащего некоторую перестановку 26 больших и 26 малых букв латинского алфавита. Правила шифрования такие: если некоторая буква в строке, предназначенной для шифрования стоит на k-м месте в обычном алфавите, то вместо нее должна быть взята буква, стоящая на k-м месте в ключе. Для дешифровки используется обратная процедура.

Напишите программу, которая тестирует модуль шифрования. Пользователь вводит строку текста, а программа зашифровывает и расшифровывает ее. Предусмотреть возможность сохранения ключа. В качестве дополнительного задания написать программу для шифрования и расшифровывания заданных при работе программы текстовых файлов.

Пример организации программы для компилятора *PGI Fortran 2010* приведен в рекомендациях по выполнению заданий.

Задание № 10

Напишите программу, которая сортирует по возрастанию строки, хранящиеся в заданном текстовом файле. Если требуемого файла не существует, то программа должна автоматически создавать его и заполнять случайными «словами».

Пример организации программы для компилятора *PGI Fortran 2010* приведен в рекомендациях по выполнению заданий.

Рекомендации по выполнению

Для решения заданий на этом практическом занятии нам потребуется базовая таблица кодов *ASCII*, а также кодировка кириллицы в *Windows* и *DOS*. Эти таблицы можно получить, написав программу, определяющую символ по его коду, а можно взять эти таблицы из справочников. Приведем ниже те таблицы, которые понадобятся для решения практических заданий.

Таблица: базовая часть кодовой таблицы *ASCII*

32	пробел	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	

Таблица: кодовая таблица *Windows 1251*

128	Ђ	144	ђ	160		176	°	192	А	208	Р	224	а	240	р
129	Ѓ	145	‘	161	Ў	177	±	193	Б	209	С	225	б	241	с
130	‚	146	’	162	ў	178	І	194	В	210	Т	226	в	242	т
131	ѓ	147	“	163	Ј	179	і	195	Г	211	У	227	г	243	у
132	„	148	”	164	¤	180	ґ	196	Д	212	Ф	228	д	244	ф
133	…	149	•	165	Ґ	181	µ	197	Е	213	Х	229	е	245	х
134	†	150	–	166	¦	182	¶	198	Ж	214	Ц	230	ж	246	ц
135	‡	151	—	167	§	183	·	199	З	215	Ч	231	з	247	ч
136	’	152	’	168	Ё	184	ё	200	И	216	Ш	232	и	248	ш
137	‰	153	™	169	©	185	№	201	Й	217	Щ	233	й	249	щ
138	Љ	154	љ	170	Є	186	є	202	К	218	Ъ	234	к	250	ъ
139	‹	155	›	171	«	187	»	203	Л	219	Ы	235	л	251	ы
140	Њ	156	њ	172	¬	188	ј	204	М	220	Ь	236	м	252	ь
141	Ќ	157	ќ	173	-	189	S	205	Н	221	Э	237	н	253	э
142	Ћ	158	ћ	174	®	190	s	206	О	222	Ю	238	о	254	ю
143	Џ	159	џ	175	Ї	191	ї	207	П	223	Я	239	п	255	я

Таблица: кодовая таблица *DOS 866*

128	А	144	Р	160	а	176	▒	192	└	208	╨	224	р	240	Ё
129	Б	145	С	161	б	177	▓	193	Г	209	╤	225	с	241	ё
130	В	146	Т	162	в	178	▓	194	┬	210	╥	226	т	242	Є
131	Г	147	У	163	г	179	|	195	├	211	╙	227	у	243	є
132	Д	148	Ф	164	д	180	┤	196	—	212	╘	228	ф	244	Ї
133	Е	149	Х	165	е	181	┤	197	┼	213	╒	229	х	245	ї
134	Ж	150	Ц	166	ж	182	┤	198	├	214	╓	230	ц	246	Ў
135	З	151	Ч	167	з	183	╖	199	├	215	╫	231	ч	247	ў
136	И	152	Ш	168	и	184	╖	200	╚	216	╪	232	ш	248	°
137	Й	153	Щ	169	й	185	┤	201	╔	217	┘	233	щ	249	∙
138	К	154	Ъ	170	к	186	|	202	╩	218	┌	234	ъ	250	·
139	Л	155	Ы	171	л	187	╗	203	╦	219	▉	235	ы	251	√
140	М	156	Ь	172	м	188	╝	204	├	220	▄	236	ь	252	№
141	Н	157	Э	173	н	189	╝	205	═	221	▌	237	э	253	¤
142	О	158	Ю	174	о	190	╜	206	╬	222	▐	238	ю	254	■
143	П	159	Я	175	п	191	┐	207	╧	223	▀	239	я	255	

Задание № 1

С помощью псевдокода запишем последовательность действий, которые необходимо выполнить для решения задачи:

```
1. Получить от пользователя имя файла
2. Если (файл не существует) То
       Закончить программу с выдачей диагностического сообщения
    Конец Если
3. Подсчитать количество больших и малых букв в файле
4. Если (количество больших > количества малых букв) То
      Преобразовать малые буквы в файле к большим
   Иначе Если (количество больших < количества малых букв) То
      Преобразовать большие буквы в файле к малым
   Иначе
      Ничего не делать
   Конец Если
```

Для того, чтобы требуемым образом преобразовать заданный файл нужно выполнить такие действия:

```
1. Переписать информацию из заданного файла во временный файл
2. Удалить заданный файл с информацией
3. Создать новый файл с заданным именем
4. Построчно переписать в новый файл с заданным именем все строки,
   выполняя требуемые преобразования символов (к верхнему или к
   нижнему регистру)
5. Удалить временный файл с данными
```

Приведем пример организации программы для компилятора *PGI Fortran 2010* (в тексте модуля присутствуют операторы, которые поддерживаются только компиляторами стандарта *Fortran 2003*). Следует обратить внимание на то, что программа разбита на два файла: **Task1mod.f90** и **Task1.f90**.

! File Task1mod.f90

```
module Task1mod

  integer(kind=4), private, parameter :: STR_WIDTH = 60

  public getFileName, isFileExists, calcNumLetters, processFile
  private isCapitalLetter, isSmallLetter

contains

  subroutine getFileName(fileName)

     implicit none

     character(len=*), intent(out) :: fileName
```

```
    write(*,'(a,i3,a)',advance='no') &
          'Type working file name of maximum ', &
          len(fileName), ' symbols: '
    read '(a)', fileName

end subroutine getFileName

function isFileExists(fileName)

    implicit none

    logical(kind=4) :: isFileExists

    character(len=*), intent(in) :: fileName

    inquire(file=trim(fileName), exist=isFileExists)

end function isFileExists

function isCapitalLetter(symbol)

  implicit none

  logical(kind=4) :: isCapitalLetter
  character(len=1) :: symbol

  integer(kind=4) :: code
  integer(kind=4), parameter :: code_A = 65, code_Z = 90

  code = iachar(symbol)

  isCapitalLetter = .false.
  if ((code >= code_A) .and. (code <= code_Z)) isCapitalLetter = .true.

end function isCapitalLetter

function isSmallLetter(symbol)

  implicit none

  logical(kind=4) :: isSmallLetter
  character(len=1) :: symbol

  integer(kind=4) :: code
  integer(kind=4), parameter :: code_a = 97, code_z = 122

  code = iachar(symbol)
```

```
  isSmallLetter = .false.
  if ((code >= code_a) .and. (code <= code_z)) isSmallLetter = .true.

end function isSmallLetter

subroutine calcNumLetters(fileName, numCapitals, numSmall)

    implicit none

    character(len=*), intent(in) :: fileName
    integer(kind=4), intent(out) :: numCapitals, numSmall

    character(len=STR_WIDTH) :: str
    integer(kind=4) :: status, i

    open(11, file=trim(fileName), status='old', action='read', &
         iostat=status, iomsg=str)
    if (status /= 0) then
       print *, 'File ' // trim(fileName) // ' opening error'
       print *, trim(str)
       stop 'Check Error and restart the program'
    end if

    numCapitals = 0;    numSmall = 0
    do
       read(11,'(a)',iostat=status) str
       if (status /= 0) exit
       do i = 1, len_trim(str)
          if (isCapitalLetter(str(i:i))) numCapitals = numCapitals + 1
          if (isSmallLetter(str(i:i))) numSmall = numSmall + 1
       end do
    end do

    close(11, status='keep')

end subroutine calcNumLetters

subroutine processFile(fileName, toBig)

    implicit none

    character(len=*), intent(in) :: fileName
    logical(kind=4), intent(in) :: toBig

    integer(kind=4) :: status
    character(len=STR_WIDTH) :: str
```

```
    open(33, status='scratch', action='readwrite', &
            iostat=status, iomsg=str)
    if (status /= 0) then
        print *, 'Scratch File opening error'
        print *, trim(str)
        stop 'Check Error and restart the program'
    end if

    call writeTmpFile()
    rewind(33)

    open(44, file=trim(fileName), status='new', action='write', &
            iostat=status, iomsg=str)
    if (status /= 0) then
        print *, 'File ' // trim(fileName) // ' opening error'
        print *, trim(str)
        stop 'Check Error and restart the program'
    end if

    do
        read(33,'(a)',iostat=status) str
        if (status /= 0) exit
        if (toBig) then
            call convertString(str, .true.)
        else
            call convertString(str, .false.)
        end if
        write(44,'(a)') str
    end do

    close(33);    close(44, status='keep')

contains

    subroutine writeTmpFile()

        open(22, file=trim(fileName), status='old', action='read', &
             iostat=status, iomsg=str)
        if (status /= 0) then
            print *, 'File ' // trim(fileName) // ' opening error'
            print *, trim(str)
            stop 'Check Error and restart the program'
        end if

        do
            read(22,'(a)',iostat=status) str
```

```
            if (status /= 0) exit
            write(33,'(a)') trim(str)
        end do

        close(22, status='delete')

    end subroutine writeTmpFile

    subroutine convertString(str, toBigLetters)

        implicit none

        character(len=*), intent(inout) :: str
        logical(kind=4), intent(in) :: toBigLetters

        integer(kind=4) :: i

        do i = 1, len_trim(str)
            if (toBigLetters) then
                if (isSmallLetter(str(i:i))) str(i:i) = achar(iachar(str(i:i))-32)
            else
                if (isCapitalLetter(str(i:i))) str(i:i) = achar(iachar(str(i:i))+32)
            end if
        end do

    end subroutine convertString

  end subroutine processFile


end module Task1mod

! ============================================
```

! File Task1.f90

```
program Task1

    use Task1mod

    implicit none

    character(len=15) :: fileName
    integer(kind=4) :: numCapitalLetters, numSmallLetters

    call getFileName(fileName)

    if (.NOT. isFileExists(fileName)) then
```

```
        print *, 'The file ', trim(fileName), ' is not exists...'
        stop 'Make file and restart program...'
    end if

    call calcNumLetters(fileName, numCapitalLetters, numSmallLetters)
    print *, 'File ' // trim(fileName)
    print *, 'Capital letters: ', numCapitalLetters
    print *, 'Small letters: ', numSmallLetters

    if (numCapitalLetters > numSmallLetters) then
        print *, 'File ' // trim(fileName) // ' is converted to big letters'
        call processFile(fileName, .true.)
    else if (numCapitalLetters < numSmallLetters) then
        print *, 'File ' // trim(fileName) // ' is converted to small letters'
        call processFile(fileName, .false.)
    else
        print *, 'File ' // trim(fileName) // ' is unchanged'
    end if

end program Task1
```

Задание № 3

Системой счисления называют совокупность правил записи чисел. Системы счисления подразделяются на позиционные и непозиционные. Как позиционные, так и непозиционные системы счисления используют определенный набор символов — цифр, последовательное сочетание которых образует число. Непозиционные системы счисления появились раньше позиционных. Они характеризуются тем, что в них символы, обозначающие то или иное число, не меняют своего значения в зависимости от местоположения в записи этого числа. Классическим примером такой системы считается римская. В ней для записи чисел используются буквы латинского алфавита. При этом буква I означает единицу, буква V — пять, X — десять, L — пятьдесят, C — сто, D — пятьсот, M — тысячу.

Для получения количественного эквивалента числа в римской системе счисления необходимо просто просуммировать количественные эквиваленты входящих в него цифр. Исключение из этого правила составляет случай, когда младшая цифра идет перед старшей, — в этом случае нужно не складывать, а вычитать число вхождений этой младшей цифры. Например:

DLXXVII => 500 + 50 + 10 + 10 + 5 + 1 + 1 = 577.

Другой пример:

CDXXIX => 500 - 100 + 10 - 1 + 10 = 429.

Задание № 9

С помощью псевдокода запишем последовательность действий, которые необходимо выполнить для решения задачи:

```
1. Сразу после запуска программа проверяет, существует ли в
рабочей папке программы неформатированный файл с ключом encrypt.bin
2. Если (файл существует) То
       Прочитать из файла алфавит и ключ
    Иначе
       Создать ключ
       Записать в файл алфавит и ключ
    Конец Если
3. С помощью ключа зашифровать строку
4. С помощью ключа расшифровать зашифрованную строку
```

Приведем пример организации модуля (файл **Task9mod.f90**) для компилятора *PGI Fortran 2010* (в тексте модуля присутствуют операторы, которые поддерживаются только компиляторами стандарта *Fortran 2003*). Следует обратить внимание на то, что основную программу (файл **Task9.f90**) нужно написать самостоятельно.

! File Task9mod.f90

```
module Encrypt

  integer(kind=4), parameter :: SIZE = 52
  character(kind=1, len=SIZE) :: alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ&
                              &abcdefghijklmnopqrstuvwxyz"
  character(kind=1, len=SIZE) :: password = "ABCDEFGHIJKLMNOPQRSTUVWXYZ&
                              &abcdefghijklmnopqrstuvwxyz"
  character(kind=1, len=11) :: fileName = "encrypt.bin"

contains

  subroutine showEncryptInfo()
    print *, 'Alphabet: ', alphabet
    print *, 'Password: ', password
  end subroutine showEncryptInfo

  subroutine getPassword()

    implicit none

    logical(kind=4) :: isFileExists
    integer(kind=4) :: status

    inquire(file=fileName, exist=isFileExists)

    if (isFileExists) then
      !print *, 'Read password'
      open(unit=111, file=fileName, status='old', action='read', &
          form='unformatted', iostat=status)
      if (status /= 0) stop 'Password file opening error'
      read(unit=111) alphabet, password
    else
      !print *, 'Write Password'
      open(unit=111, file=fileName, status='new', action='write', &
          form='unformatted', iostat=status)
```

```
    if (status /= 0) stop 'Password file opening error'
    call createPassword()
    write(unit=111) alphabet, password
  end if

  close(unit=111, iostat=status)
  if (status /= 0) stop 'Password file closing error'

end subroutine getPassword


subroutine createPassword()

  implicit none

  integer(kind=4) :: i, j
  real(kind=4) :: x
  character(kind=1, len=1) :: tmp

  call random_seed()
  do i = 1, len_trim(password)
    call random_number(x)
    j = nint(1 + (len_trim(password)-1)*x)
    if (i /= j) then
      tmp = password(i:i)
      password(i:i) = password(j:j)
      password(j:j) = tmp
    end if
  end do

end subroutine createPassword

function encodeChar(ch)

  implicit none

  character(kind=1, len=1) :: encodeChar
  character(kind=1, len=1), intent(in) :: ch

  integer(kind=4) :: ind

  ind = index(alphabet, ch)

  if (ind /= 0) then
    encodeChar = password(ind:ind)
  else
    encodeChar = ch
  end if

end function encodeChar
```

```
  function decodeChar(ch)

    implicit none

    character(kind=1, len=1) :: decodeChar
    character(kind=1, len=1), intent(in) :: ch

    integer(kind=4) :: ind

    ind = index(password, ch)

    if (ind /= 0) then
      decodeChar = alphabet(ind:ind)
    else
      decodeChar = ch
    end if

  end function decodeChar


  function encodeString(str)

    implicit none

    character(kind=1, len=*), intent(in) :: str
    character(kind=1, len=len(str)) :: encodeString

    integer(kind=4) :: i

    do i= 1, len_trim(str)
      encodeString(i:i) = encodeChar(str(i:i))
    end do

  end function encodeString


  function decodeString(str)

    implicit none

    character(kind=1, len=*), intent(in) :: str
    character(kind=1, len=len(str)) :: decodeString

    integer(kind=4) :: i

    do i= 1, len_trim(str)
      decodeString(i:i) = decodeChar(str(i:i))
    end do

  end function decodeString

end module Encrypt
```

Задание № 10

С помощью псевдокода запишем последовательность действий, которые необходимо выполнить для решения задачи:

```
1. Сразу после запуска программа проверяет, существует ли в
рабочей папке программы файл, который нужно отсортировать
2. Если (файл существует) То
       Переписать из него строки в файл прямого доступа и
              удалить исходный файл
       Отсортировать файл прямого доступа
       Переписать из него строки в новый файл с заданным именем и
              удалить временный файл
    Иначе
       Создать файл для сортировки
    Конец Если
```

Приведем пример организации модуля (файл **Task10mod.f90**) для компилятора *PGI Fortran 2010* (в тексте модуля присутствуют операторы, которые поддерживаются только компиляторами стандарта *Fortran 2003*). Следует обратить внимание на то, что основную программу (файл **Task10.f90**) нужно написать самостоятельно.

```
! File Task10mod.f90
module SortFile

  integer(kind=4), private, save :: numLines, a, b
  integer(kind=4), public, parameter :: LINE_LENGTH = 60

  private :: makeIniData

contains

   ! ========================================
  function isFileExists(fileName)

    implicit none

    logical(kind=4) :: isFileExists
    character(len=*), intent(in) :: fileName

    inquire(file=trim(fileName), exist = isFileExists)

  end function isFileExists

  ! ========================================

  subroutine makeIniData()
```

```
    implicit none

    real(kind=4) :: x
    integer(kind=4), parameter :: numMin = 100, numMax = 1000
    integer(kind=4), parameter :: xMin = 97, xMax = 122

    call random_seed()
    call random_number(x);  numLines = int(numMin + (numMax-numMin)*x)
    a = xMin; b = xMax

end subroutine makeIniData

! ========================================
subroutine createNewFile(fileName)

  implicit none

  character(len=*), intent(in) :: fileName

  integer(kind=4) :: i, j, status, numWords, lineLength, wordLength
  real(kind=4) :: x
  character(kind=1,len=60) :: msg
  character(len=LINE_LENGTH) :: str

  call makeIniData()

  open(11, file=fileName, status='new', action='write', &
           iostat=status, iomsg=msg)
  if (status /= 0) then
    print *, '->',trim(msg),'<-'
    stop "ERROR while opening file"
  end if

  call random_seed()
  do i = 1, numLines
     call random_number(x)
     numWords = 5 + int(10*x)
     call random_number(x)
     lineLength = int(0.5*LINE_LENGTH * (1.0 + x))
     wordLength = (lineLength - numWords + 1) / numWords
     str = createRandomWord(wordLength)
     do j = 2, numWords
        str = trim(str) // " " // createRandomWord(wordLength)
        !write(11, '(a)') createRandomWord(5)
        !write(*, '(i5, " word: ",a)') i, createRandomWord(5)
     end do
     write(*, '(i5, " line: ",a)') i, str
```

```
      write(11, '(a)') str
    end do

    close(11, status='keep')

  contains

    function createRandomWord(n) result (res)

      implicit none

      integer(kind=4), intent(in) :: n
      character(len=n) :: res

      integer(kind=4) :: i

      call random_number(x)
      res = achar(int(a + (b - a) * x))
      do i = 1, n
        call random_number(x)
        !print *, int(a + (b - a) * x), x
        res = trim(res) // achar(int(a + (b - a) * x))
      end do

    end function createRandomWord

  end subroutine createNewFile

    subroutine writeToBinFile(fileName)

      implicit none

      character(len=*), intent(in) :: fileName

      integer(kind=4) :: status, num
      character(len=LINE_LENGTH) :: str

      open(unit=10, file=trim(fileName), status='old', action='read', &
           iostat=status)
      if (status /= 0) stop "File opening error..."
      open(unit=20, file="tmp.bin", status='new', action='write', &
           access='direct', form="formatted", recl=LINE_LENGTH, iostat=status)
      if (status /= 0) stop "File opening error..."

      num = 0
      do
         read(10,'(A)', iostat=status) str
```

```
        if (status /= 0) exit
        num = num + 1
        print *, '->', trim(adjustl(str)), '<-'
        write(20, '(A)', rec=num) str
    end do

    close(10, status='delete');  close(20, status='keep')

end subroutine writeToBinFile

subroutine readFromBinFile(fileName)

    implicit none

    character(len=*), intent(in) :: fileName

    integer(kind=4) :: status, num
    character(len=LINE_LENGTH) :: str

    open(unit=20, file="tmp.bin", status='old', action='read', &
         access='direct', form="formatted", recl=LINE_LENGTH, iostat=status)
    if (status /= 0) stop "File opening error..."
    open(unit=40, file=fileName, status='new', action='write', &
           iostat=status)
    if (status /= 0) stop "File opening error..."

    num = 0
    do
        num = num + 1
        read(20, '(A)',rec=num, iostat=status) str
        if (status /= 0) exit
        write(40, '(a)') trim(str)
    end do

    close(20, status='delete');    close(40, status='keep')

end subroutine readFromBinFile

subroutine sortBinFile()

    implicit none

    integer(kind=4) :: status, i, j, numFileLength
    character(len=LINE_LENGTH) :: str, tmp

    open(unit=20, file="tmp.bin", status='old', action='readwrite', &
         access='direct', form="formatted", recl=LINE_LENGTH, iostat=status)
```

```
        if (status /= 0) stop "File opening error..."

        numFileLength = numLines()
        do i = 1, numFileLength-1
           read(20, '(A)',rec=i, iostat=status) str
           if (status /= 0) stop "File reading error..."
           do j = i+1, numFileLength
              read(20, '(A)',rec=j, iostat=status) tmp
              if (llt(tmp, str)) then
                 write(20, '(A)',rec=i, iostat=status) tmp
                 if (status /= 0) stop "File writing error..."
                 write(20, '(A)',rec=j, iostat=status) str
                 if (status /= 0) stop "File writing error..."
                 str = tmp
              end if
           end do
        end do

        close(20, status='keep')

    contains

        function numLines()

           implicit none

           integer(kind=4) :: numLines
           integer(kind=4) :: num

           num = 0
           do
              num = num + 1
              read(20, '(A)', rec=num, iostat=status) str
              if (status /= 0) exit
           end do

           numLines = num-1

        end function numLines

    end subroutine sortBinFile

end module SortFile
```

Тема № 4. Одномерные массивы

На данном занятии следует изучить особенности применения статических и динамических одномерных массивов. Необходимо рассмотреть основные методы работы с массивами — доступ к элементам, присваивание начальных значений, вывод массива на экран и сохранение его в файл, изменение порядка следования элементов в массиве, сортировка и поиск. Кроме того, кратко познакомиться с основами передачи массивов в процедуры и рассмотреть применение некоторых встроенных функций, предназначенных для работы с массивами.

Список заданий

Задание № 1

Написать программу, которая сначала проверяет, существует ли в рабочей папке программы заданный пользователем файл. Если файла нет, то программа создает этот неформатированный файл и записывает в него сначала размер одномерного целочисленного массива, а затем его элементы. Длина массива вводится пользователем во время работы программы. Элементы целочисленного массива либо вводятся с клавиатуры, либо генерируются случайным образом из заданного диапазона. Если указанный пользователем файл существует в рабочей папке программы, то сохраненная информация считывается из него.

Затем в заданном целочисленном массиве чисел $Z(n)$ найти количество чередований знака, то есть количество переходов с минуса на плюс и с плюса на минус. При вычислениях отнести нуль к положительным числам.

Пример организации программы для компилятора *PGI Fortran 2010* приведен в рекомендациях по выполнению заданий.

Задание № 2

Вычислить среднее значение x_{aver} и дисперсию d_x для заданного массива наблюдений $x(k)$ по формулам:

$$s_1=\sum_{i=1}^{k} x_i \qquad s_2=\sum_{i=1}^{k} x_i^2 \qquad x_{aver}=\frac{s_1}{k} \qquad d_x=\frac{s_2}{k-1}-\frac{s_1^2}{k(k-1)}$$

Задание № 3

В массиве $A(n)$ найти и напечатать номера (индексы) локальных максимумов, то есть таких элементов массива a_i, для которых выполняется условие $a_{i-1}<a_i>a_{i+1}$.

Задание № 4

Каждый из элементов x_i массива $X(n)$ заменить средним значением первых i элементов этого массива.

Замечание: при решении задачи следует обратить внимание на то, что бы при вычислении соответствующих средних значений использовались исходные, не измененные значения элементов заданного массива $X(n)$.

Задание № 5

Каждый элемент одномерного массива $A(n)$ (кроме 2-х крайних) заменить выражением ($i=2,3,\ldots,n-1$):

$$\acute{a}_i=\frac{a_{i-1}+2a_i+a_{i+1}}{4},$$

а крайние элементы — выражениями:

$$\acute{a}_1=\frac{a_1+a_2}{2} \qquad \acute{a}_n=\frac{a_{n-1}+a_n}{2}.$$

Замечание: при решении задачи следует обратить внимание на то, что бы при вычислении новых, «сглаженных» значений элементов массива использовались бы исходные, не измененные значения элементов заданного массива $A(n)$.

Задание № 6

Пусть даны координаты n точек на плоскости: (x_1, y_1), (x_2, y_2), …, (x_n, y_n). Найдите две точки, расстояние между которыми наибольшее (считайте, что такая пара точек единственная).

Замечание: программу следует организовать так, чтобы данные о точках сохранялись в файле; а при запуске программы проверялось бы наличие файла с данными, и если он есть, то информация о точках считывалась бы из него.

Задание № 7

Задан одномерный целочисленный массив. Необходимо удалить элементы массива, расположенные между его минимальным и максимальным элементами (не включая минимальный и максимальный элементы).

Замечание: пример модульной процедуры изменения размера массива приведен в приведен в рекомендациях по выполнению заданий.

Задание № 8

Написать программу, которая, преобразовывает массив таким образом, чтобы в первой его части располагались все нечетные элементы, а после них — все четные.

Замечание: пример модульной процедуры преобразования массива приведен в приведен в рекомендациях по выполнению заданий.

Задание № 9

Написать программу, которая заданным методом (методом прямого выбора либо методом обмена — «*пузырька*») сортирует по заданному критерию одномерный массив.

Замечание: пример модульной процедуры сортировки массива приведен в приведен в рекомендациях по выполнению заданий.

Задание № 10

Написать программу, которая, используя метод бинарного поиска, выполняет поиск в упорядоченном по возрастанию массиве. Написать нерекурсивную и рекурсивную версии программы.

Рекомендации по выполнению

Задание № 1

С помощью псевдокода запишем последовательность действий, которые необходимо выполнить для решения задачи:

```
1. Получить от пользователя имя файла
2. Если (файл существует) То
       Прочитать длину массива
       Выделить для элементов массива память
       Прочитать из файла значения элементов массива
    Иначе
       Узнать у пользователя размер массива
       Выяснить каким образом присвоить элементам массива значения
       Требуемым образом присвоить элементам массива значения
       Сохранить информацию в файл
    Конец Если
3. Вывести элементы массива на экран
4. Просчитать и вывести на экран количество смен знака
```

Приведем пример организации программы для компилятора *PGI Fortran 2010* (в тексте модуля присутствуют операторы, которые поддерживаются только компиляторами стандарта *Fortran 2003*). Следует обратить внимание на то, что программа разбита на два файла: **Task1mod.f90** и **Task1.f90**.

```
! File Task1mod.f90
module Task1mod

   integer(kind=4), allocatable :: a(:)
   integer(kind=4) :: num

   private constructArray, destroyArray

contains

   subroutine constructArray(n)

      implicit none

      integer(kind=4), intent(in) :: n
      integer(kind=4) :: error

      allocate(a(1:n), stat=error)
      if (error /= 0) stop 'Allocation memory error'

   end subroutine constructArray


   subroutine destroyArray()
```

```
    implicit none

    integer(kind=4) :: error

    deallocate(a, stat=error)
    if (error /= 0) stop 'Deallocation memory error'

end subroutine destroyArray


subroutine readArray(fileName)

    implicit none

    character(len=*), intent(in) :: fileName

    integer(kind=4) :: i, error
    character(len=80) :: msg

    open(33, file=trim(fileName), form='unformatted', status='old', &
        action='read', iostat=error, iomsg=msg)
    if (error /= 0) then
        print *, 'IOError while opening file ' // trim(fileName)
        print *, trim(msg)
        stop 'IOERROR!!!'
    end if
    read(33) num
    call constructArray(num)
    do i = 1, num
        read(33) a(i)
    end do
    close(33, status='keep')

end subroutine readArray


subroutine makeArray(fileName)

    implicit none

    character(len=*), intent(in) :: fileName

    character(len=1) :: ans
    integer(kind=4) :: i, error
    character(len=80) :: msg
    real(kind=4) :: x
```

```
    integer(kind=4), parameter :: LEFT = -10, RIGHT = 10

    do
       print '(a,$)', 'Array size: '
       read *, num
       if (num > 0) exit
       print *, 'Size of array must be positive'
    end do
    call constructArray(num)
    print *, 'Type y or Y to manual input or any other to automatic'
    print '(3x,a,$)', '-> '
    read *, ans
    if ((ans == 'y') .OR. (ans == 'Y')) then
       print *, 'Manual'
       print '("Type ",i3, " array values")', num
       do i = 1, num
          print '(5x,"a[", i2, "]: ", $)', i
          read *, a(i)
       end do
    else
       print *, 'Automatic'
       call random_seed()
       do i = 1, num
          call random_number(x)
          a(i) = nint(LEFT + (RIGHT-LEFT)*x)
       end do
    end if
    open(22, file=trim(fileName), form='unformatted', status='new', &
       action='write', iostat=error, iomsg=msg)
    if (error /= 0) then
       print *, 'IOError while opening file ' // trim(fileName)
       print *, trim(msg)
       stop 'IOERROR!!!'
    end if
    write(22) num
    do i = 1, num
       write(22) a(i)
    end do
    close(22, status='keep')

end subroutine makeArray


subroutine processArray(fileName)

    implicit none
```

```
    character(len=*), intent(in) :: fileName

    if (isFileExists(trim(fileName))) then
        print *, 'Read array from file'
        call readArray(fileName)
    else
        print *, 'Create array and write to file'
        call makeArray(fileName)
    end if

    call printArray()

    print '(3x,a,i3,a,a)', 'There are ', znakChanges(), ' sign ', &
          'changes in the array'

    call destroyArray()

contains

    subroutine printArray()

        integer(kind=4) :: i

        print '(3x,a)', 'Array'
        do i = 1, num
            print '(1x,i3,$)', a(i)
        end do
        print *

    end subroutine printArray

end subroutine processArray

function znakChanges()

    implicit none

    integer(kind=4) :: znakChanges
    integer(kind=4) :: i, pred, curr, res

    res = 0
    pred = znak(a(1))
    do i = 2, num
        curr = znak(a(i))
        if (pred /= curr) then
            res = res + 1
            pred = curr
```

```
            end if
        end do

        znakChanges = res

    end function znakChanges


    function znak(a)

        implicit none

        integer(kind=4) :: znak
        integer(kind=4), intent(in) :: a

        if (a >= 0) then
            znak = 1
        else
            znak = -1
        end if

    end function znak


    function isFileExists(fileName)

        implicit none

        character(len=*), intent(in) :: fileName
        logical(kind=4) :: isFileExists

        inquire(file=fileName, exist=isFileExists)

    end function isFileExists

end module Task1mod

! File Task1.f90
program Task1

    use Task1mod

    implicit none

    character(len=20) :: fileName

    !call processArray('test.bin')
```

```
    print *, 'Type Name of File: '
    print '(3x,a,$)', '-> '
    read '(a)', fileName

    call processArray(trim(fileName))


end program Task1
```

Задание № 7

Для решения задания сначала нужно сделать копию временную исходного массива, удалить из памяти исходный массив, вычислить новый размер массива, выделить требуемый объем памяти, заполнить нужными значениями и удалить временный массив из памяти.

В данном фрагменте переменные posMin и posMax — это индексы минимального и максимального элементов массива (переменные модуля), которые до вызова подпрограммы должны быть определены.

```
subroutine reallocateArray()

        implicit none

        integer(kind=4), allocatable :: tmp(:)
        integer(kind=4) :: tmpNum

        integer(kind=4) :: i, first, second, error

        call makeCopy()

        if (posMin < posMax) then
           first = posMin;  second = posMax
        else
           first = posMax;  second = posMin
        end if

        num = tmpNum - (second - first - 1)
        call createArray(num)
        do i = 1, first
           arr(i) = tmp(i)
        end do
        do i = second, tmpNum
           arr(i- (second - first - 1)) = tmp(i)
        end do

        deallocate(tmp, stat=error)
        if (error /= 0) stop 'Deallocation memory error'
        tmpNum = 0
```

```
    contains

        subroutine makeCopy()
            tmpNum = num
            allocate(tmp(1:tmpNum), stat=error)
            if (error /= 0) stop 'Allocation memory error'
            do i = 1, num
                tmp(i) = arr(i)
            end do
            call destroyArray()
        end subroutine makeCopy

end subroutine reallocateArray
```

Задание № 8

Смысл перестановки элементов заключается в следующем: с двух сторон идем к центру массива, проверяя на своем ли месте стоят крайние элементы. Если оба крайних элемента стоят не на своем месте, то они обмениваются значениями.

```
subroutine processArray()

        implicit none

        integer(kind=4) :: i, j, tmp

        i = 1;  j = num
        do while (i < j)
          if (isOdd(arr(i))) Then
            i = i + 1
          else if (isOdd(arr(j))) Then
            tmp = arr(i)
            arr(i) = arr(j)
            arr(j) = tmp
            i = i + 1
            j = j - 1
          else
            j = j - 1
          end if
        end do

    contains

        function isOdd(x) ! NeChetnost

            logical(kind=4) :: isOdd
            integer(kind=4), intent(in) :: x
```

```
    isOdd = mod(x, 2) == 1

  end function isOdd

end subroutine processArray
```

Задание № 9

Алгоритм метода сортировки состоит в повторяющихся проходах по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован, либо выполнено размер_массива - 1 проход. Кроме того, по мере выполнения сортировки в конце массива гарантировано окажутся требуемые элементы, т. е. на каждом просмотре попарное сравнение элементов выполняется до размер_массива - i-го, где i –номер просмотра.

Алгоритм сортировки массива по возрастанию методом прямого выбора может быть представлен так: просматривая массив от первого элемента, найти минимальный элемент и поместить его на место первого элемента, а первый — на место минимального; затем просматривая массив от второго элемента, найти минимальный элемент и поместить его на место второго элемента, а второй — на место минимального; и так далее до предпоследнего элемента.

Для удобства работы логическая функция–критерий, указывающая в правильном ли порядке расположены соседние элементы массива передается в процедуру как процедура–параметр.

```
subroutine bubbleSort(crit)

  implicit none

  interface
    function crit(a1, a2)
      implicit none
      logical(kind=4) :: crit
      integer(kind=4), intent(in) :: a1, a2
    end function crit
  end interface

  integer(kind=4) :: i, j, tmp
  logical(kind=4) :: obmen

  i = 1
  do
    obmen = .false.
    do j = 1, num-i
      if (.not. crit(arr(j), arr(j+1))) then
        obmen = .true.
        tmp = arr(j)
        arr(j) = arr(j+1)
```

```
                arr(j+1) = tmp
            end if
         end do
        i = i + 1
        if ((i == num) .Or. (.Not. obmen)) exit
      end do

end subroutine bubbleSort
```

Примерный вариант модуля с процедурами–критериями приведен ниже:

```
module Criters

contains

   function increase(a1, a2)

      implicit none

      logical(kind=4) :: increase
      integer(kind=4), intent(in) :: a1, a2

      increase = a1 < a2

   end function increase

   function decrease(a1, a2)

      implicit none

      logical(kind=4) :: decrease
      integer(kind=4), intent(in) :: a1, a2

      decrease = a1 > a2

   end function decrease

end module Criters
```

Задание № 10

На практике довольно часто производится поиск в массиве, элементы которого упорядочены по некоторому критерию (такие массивы называются упорядоченными). Пусть есть упорядоченный по возрастанию массив целых чисел. Нужно определить, содержит ли этот массив некоторое число (образец).

Метод бинарного поиска может быть реализован так: сначала образец сравнивается со средним (по номеру) элементом массива. Если образец равен среднему элементу, то задача решена. Если образец больше среднего элемента, то это значит, что искомый элемент расположен выше среднего элемента и новый поиск осуществляется между элементами с

номерами sred+1 и up. Если образец меньше среднего элемента, то это значит, что искомый элемент расположен ниже среднего элемента и новый поиск осуществляется между элементами с номерами low и sred-1. После того как определена часть массива, в которой может находиться искомый элемент, вычисляется новое значение sred и поиск продолжается.

Пример модульной процедуры итерационного бинарного поиска приведен ниже:

```
subroutine binSearch(obr, sred, found)

    implicit none

    integer(kind=4), intent(in) :: obr
    integer(kind=4), intent(out) :: sred
    logical(kind=4), intent(out) :: found

    integer(kind=4) :: low, up

    low = 1;  up = num; found = .false.
    do
       sred = (up - low) / 2 + low
      if (arr(sred) == obr) Then
         found = .true.
      else
         if (obr < arr(sred)) then
            up = sred - 1
         else
            low = sred + 1
         end if
      end if
      if ((low > up) .or. found) exit
    end do

end subroutine binSearch
```

Тема № 5. Двумерные массивы

На данном занятии следует изучить особенности применения статических и динамических двумерных массивов. Необходимо рассмотреть основные методы работы с массивами — доступ к элементам, присваивание начальных значений, вывод двумерного массива на экран и сохранение его в файл, изменение порядка следования элементов в массиве, сортировка, поиск, а также познакомится с методами численного дифференцирования таблично заданной функции и изучить методы интерполяции. Кроме того, кратко познакомиться с основами передачи массивов в процедуры и рассмотреть применение некоторых встроенных функций, предназначенных для работы с массивами.

Список заданий

Задание № 1

Написать программу, которая сначала проверяет, существует ли в рабочей папке программы заданный пользователем файл. Если файла нет, то программа создает этот форматный файл и записывает в него количество столбцов и строк двумерного целочисленного массива, а затем все его элементы. Размеры двумерного массива вводятся пользователем во время работы программы. Элементы двумерного целочисленного массива могут либо вводиться с клавиатуры, либо генерироваться случайным образом из заданного диапазона. Если указанный пользователем файл существует в рабочей папке программы, то сохраненная информация считывается из него.

Затем в заданной целочисленной матрице найти элемент, являющийся максимальным в своей строке и минимальным в своем столбце. Если в матрице такого элемента нет, то вывести соответствующее сообщение.

Пример организации программы для компилятора *PGI Fortran 2010* приведен в рекомендациях по выполнению заданий.

Задание № 2

Написать программу, которая в квадратной матрице M в строках с отрицательным элементом на главной диагонали находит наибольший элемент из всех элементов строки и сумму всех таких максимальных элементов.

Пример процедуры, выполняющей заданные действия приведен в рекомендациях к данному заданию.

Задание № 3

Задана целочисленная квадратная матрица A порядка order. Следует зеркально отразить ее элементы относительно побочной диагонали. (при этом элементы, находящиеся на побочной диагонали остаются на прежнем месте; так, например элемент $A_{1,1}$ меняется местами с элементом $A_{\text{order},\text{order}}$, элемент $A_{1,2}$ — с $A_{\text{order}-1,\text{order}}$ и т. д.).

Пример процедуры, выполняющей заданные действия приведен в рекомендациях к данному заданию.

Задание № 4

Задана целочисленная квадратная матрица A порядка order . Найти сумму элементов каждой ее диагонали, параллельной главной (начиная с диагонали $A_{1,\text{order}}$, состоящей из одного элемента).

Пример процедуры, выполняющей заданные действия приведен в рекомендациях к данному заданию.

Задание № 5

В двумерном массиве X все числа различны. В каждой строке находится минимальный элемент, затем среди этих чисел выбирается максимальное. Напечатать номера строки и столбца массива X, на пересечении которых расположено выбранное число.

Пример программы, предназначенной для решения данного задания приведен в рекомендациях. Следует обратить внимание на способ, с помощью которого массив передается как параметр в процедуры программы.

Задание № 6

Задана целочисленная матрица размером $m \times n$. Найти матрицу, получающуюся из данной:

a) перестановкой столбцов — первого с последним, второго с предпоследним и т.д.
b) перестановкой строк — первой с последней, второй с предпоследней и т.д.

Пример процедуры, выполняющей заданные действия приведен в рекомендациях к данному заданию.

Задание № 7

Задана матрица вещественных чисел размером $n \times m$. Упорядочить строки матрицы по убыванию сумм элементов строк.

Пример процедуры, выполняющей заданные действия приведен в рекомендациях к данному заданию.

Задание № 8

Задана матрица A . Первая строка этой матрицы — представляет собой многочлен $P_n(x)$, заданный набором своих коэффициентов. Вычислить остальные строки этой матрицы — массивы коэффициентов производных этого многочлена (вторая строка матрицы — первая производная многочлена, третья строка — вторая производная многочлена и т.д.). Вычислить значение многочлена и его производных для заданного значения x .

Пример организации программы приведен в рекомендациях по выполнению заданий.

Задание № 9

Задан целочисленный двумерный массив A размером $\text{row} \times \text{col}$. Следует удалить из него k –ую строку и l -ый столбец ($k \le \text{row}$, $l \le \text{col}$).

Пример модельной процедуры, выполняющей заданные действия приведен в рекомендациях к данному заданию.

Задание № 10

Построить таблицу значений функции $f(x) = \exp(-x) \cdot \sin(x)$ в заданных пользователем интервалах с заданным шагом (для начала взять интервал $[0; 2\pi]$). Полученную таблично заданную функцию численно продифференцировать и вычислить

значение производной в произвольной заданной точке внутри интервала табуляции.

Основные расчетные формулы и пример организации программы приведен в рекомендациях по выполнению заданий.

В качестве дополнительного задания следует так изменить программу, чтобы табулируемую функцию можно было передавать как параметр в процедуру табуляции. Кроме того, следует написать процедуру получения от пользователя точки, в которой следует вычислить производную таблично заданной функции.

Рекомендации по выполнению

Задание № 1

Организация программы аналогична программе, предназначенной для решения **Задания № 1** из **Темы № 4**. С помощью псевдокода запишем последовательность действий, которые необходимо выполнить для решения задачи:

```
1. Получить от пользователя имя файла
2. Если (файл существует) То
       Прочитать размеры массива
       Выделить для элементов массива память
       Прочитать из файла значения элементов массива
    Иначе
       Узнать у пользователя размеры массива
       Выяснить каким образом присвоить элементам массива значения
       Требуемым образом присвоить элементам массива значения
       Сохранить информацию в файл
    Конец Если
3. Вывести элементы массива на экран
4. Найти элемент являющийся максимальным в своей строке и
   минимальным в своем столбце
5. Если (элемент найден) То
       Вывести элемент и расположение на экран
   Иначе
       Вывести сообщение об отсутствии требуемого элемента
   Конец
```

Приведем пример организации программы для компилятора *PGI Fortran 2010* (в тексте модуля присутствуют операторы, которые поддерживаются только компиляторами стандарта *Fortran 2003*). Следует обратить внимание на то, что программа разбита на два файла: **Task1mod.f90** и **Task1.f90**.

```
! File Task1mod.f90
module Task1mod

   integer(kind=4), allocatable, private :: m(:,:)
   integer(kind=4), save, private :: row, col

contains

   ! ============================================
   function isFileExists(fileName)
```

```
    implicit none

    character(len=*), intent(in) :: fileName
    logical(kind=4) :: isFileExists

    inquire(file=trim(fileName), exist=isFileExists)

end function isFileExists

! ============================================
subroutine getArray(fileName)

    implicit none

    character(len=*), intent(in) :: fileName

    integer(kind=4) :: i, j, error
    character(len=80) :: msg

    if (isFileExists(trim(fileName))) then
        print *, 'Read array from file: ' // trim(fileName)
        call readArray()
    else
        print *, 'Create array and write to file: ' // trim(fileName)
        call makeArray()
        call writeArray()
    end if

contains

    subroutine readArray()

        open(33, file=trim(fileName), form='unformatted', &
                status='old', action='read', &
                iostat=error, iomsg=msg)
        if (error /= 0) then
            print *, 'IOError while opening file ' // trim(fileName)
            print *, trim(msg)
            stop 'IOERROR!!!'
        end if
        read(33) row, col
        call constructArray()
        read(33) m

        close(33, status='keep')
```

```
end subroutine readArray

subroutine makeArray()

    character(len=1) :: ans
    integer(kind=4), parameter :: LEFT = -50, RIGHT = 50
    real(kind=4) :: x

    do
        print '(a,$)', 'Number of rows: '
        read *, row
        if (row > 0) exit
        print *, 'Number of rows must be positive'
    end do
    do
        print '(a,$)', 'Number of cols: '
        read *, col
        if (col > 0) exit
        print *, 'Number of cols must be positive'
    end do

    call constructArray()

    print *, 'Type y or Y to manual input or any other to automatic'
    print '(3x,a,$)', '-> '
    read *, ans

    if ((ans == 'y') .OR. (ans == 'Y')) then
        print *, 'Manual'
        print '("Type ",i3, " array rows")', row
        print '("Type ", i3, " elements and press <Enter> ")', col
        do i = 1, ROW
            print '(5x,"row ", i2, ": ", $)', i
            read *, (m(i,j), j=1,COL)
        end do
    else
        print *, 'Automatic'
        call random_seed()
        do i = 1, row
            do j = 1, col
                call random_number(x)
                m(i,j) = nint(LEFT + (RIGHT-LEFT)*x)
            end do
        end do
    end if

end subroutine makeArray
```

```
    subroutine writeArray()

        open(22, file=trim(fileName), form='unformatted', &
                status='new', action='write', &
                iostat=error, iomsg=msg)
        if (error /= 0) then
            print *, 'IOError while opening file ' // trim(fileName)
            print *, trim(msg)
            stop 'IOERROR!!!'
        end if
        write(22) row, col
        write(22) m
        close(22, status='keep')

    end subroutine writeArray

end subroutine getArray

! ============================================
subroutine constructArray()

    implicit none

    integer(kind=4) :: error

    allocate(m(1:row, 1:col), stat=error)
    if (error /= 0) stop 'Allocation memory error'

end subroutine constructArray

! ============================================
subroutine destroyArray()

    implicit none

    integer(kind=4) :: error

    deallocate(m, stat=error)
    if (error /= 0) stop 'Deallocation memory error'

end subroutine destroyArray

! ============================================
subroutine printArray()

    implicit none
```

```
        integer(kind=4) :: i, j

        do i = 1, ROW
           do j = 1, COL
              print '(i4,$)', m(i,j)
           end do
           print *
        end do

    end subroutine printArray

    ! ============================================
    subroutine findSaddlePoints()

        implicit none

        logical(kind=4) :: isSaddlePoint
        integer(kind=4) :: i, j, maxJ, minI

        isSaddlePoint = .false.;
        do i = 1, ROW
           ! Let find max element in row i
           maxJ = 1
           do j = 2, COL
              if (m(i,maxJ) < m(i,j)) maxJ = j
           end do
           ! m[i][maxJ] - maximum in row i
           ! Let find min element in column maxJ
           minI = 1;
           do j = 2, ROW
              if (m(minI,maxJ) > m(j,maxJ)) minI = j;
           end do
           ! a[minI][maxJ] - mainumum in col maxJ
           if (i == minI) then
              isSaddlePoint = .true.;
              print '("Saddle Point: m[", i2, "][", i2, "] = ", i3)', &
                    minI, maxJ, m(minI,maxJ);
           endif
        end do

        if (.not. isSaddlePoint) print *, "There is no Saddle Point in matrix";

    end subroutine findSaddlePoints

end module Task1mod
```

```
! File Task1.f90
program Task1

    use Task1mod

    implicit none

    character(len=20) :: fileName

    print *, 'Type Name of File: '
    print '(3x,a,$)', '-> '
    read '(a)', fileName

    call getArray(trim(fileName))
    call printArray()
    call findSaddlePoints()
    call destroyArray()


end program Task1
```

Задание № 2

Программа, предназначенная для решения этого задания имеет ту же самую структуру, что и программа из **Задания № 1**. Модульная подпрограмма, собственно предназначенная для решения задания, приведена ниже:

```
subroutine processArray()

        implicit none

        logical(kind=4) :: isNegativeElement
        integer(kind=4) :: i, j, ind, sum

        isNegativeElement = .false.;
        sum = 0
        do i = 1, order
           if (m(i,i) < 0) then
              ! Let find max element in row
              ind = 1
              do j = 1, order
                 if (m(i,ind) < m(i,j)) ind = j
              end do
              print '(3x,"Row ", i2, " max: ", i3)', i, m(i,ind)
              sum = sum + m(i,ind)
              isNegativeElement = .true.;
           end if
        end do
```

```
    if (.not. isNegativeElement) then
       print *, "There is no negative &
              &elements on basic diagonal in matrix";
    else
       print *, "Sum: ", sum
    end if

 end subroutine processArray
```

Задание № 3

Программа, предназначенная для решения этого задания имеет ту же самую структуру, что и программа из **Задания № 2**. Модульная подпрограмма, собственно предназначенная для решения задания, приведена ниже:

```
subroutine mirrorArray()

    implicit none

    integer(kind=4) :: i, j, tmp

    do i = 1, order
       do j = 1, order-i
          tmp = m(i,j)
          m(i,j) = m(order-j+1,order-i+1)
          m(order-j+1,order-i+1) = tmp
       end do
    end do

end subroutine mirrorArray
```

Задание № 4

Программа, предназначенная для решения этого задания имеет ту же самую структуру, что и программа из **Задания № 3**. Модульная подпрограмма, собственно предназначенная для решения задания, приведена ниже:

```
subroutine processArray()

    implicit none

    integer(kind=4) :: i, k, sum

    do k = order - 1, -order+1, -1
       sum = 0
       do i = 1, order - abs(k)
          if (k >= 0) then
             sum = sum + m(i,i+k)
          else
```

```
                sum = sum + m(i+abs(k),i)
            end if
          end do
          print *, sum
        end do

    end subroutine processArray
```

Задание № 5

При анализе программы особое внимание следует обратить на то, как массивы передаются в процедуры в виде параметров.

```
program Task5

    implicit none

    integer row, col, error
    integer, allocatable :: matr(:,:)

    call getArraySize(row, col)

    allocate(matr(1:row, 1:col), stat=error)
    if (error /= 0) stop 'Allocation Memory Error'

    call setDifferentData(matr, row, col)
    print *, 'Working Array'
    call printMatrix(matr, row, col)
    print *
    call solveProblem(matr, row, col)

    deallocate(matr, stat=error)
    if (error /= 0) stop 'Deallocation Memory Error'

contains

  subroutine getArraySize(row, col)

      integer, intent(out) :: row, col

      do
         print '(5x,a,$)', 'Input Number of Array Rows: '
         read *, row
         if (row > 0) exit
         print '(1x,a)', 'ERROR!! Number of array rows must be positive'
      end do
      do
         print '(5x,a,$)', 'Input Number of Array Cols: '
         read *, col
```

```
        if (col > 0) exit
        print '(1x,a)', 'ERROR!! Number of array cols must be positive'
      end do

  end subroutine getArraySize

end program Task5


subroutine setDifferentData(m, row, col)

  implicit none

  integer, intent(in) :: row, col
  integer, intent(out) :: m(row, col)

  integer i, j, tmp, posI, posJ
  real x

  tmp = 1
  do i = 1, row
     do j = 1, col
        m(i, j) = tmp
        tmp = tmp + 1
     end do
  end do

  call random_seed()
  do i = 1, row
      do j = 1, col
         call random_number(x)
         posI = int(1 + (row-1) * x)
         call random_number(x)
         posJ = int(1 + (col-1) * x)
         if ((posI /= i) .or. (posJ /= j)) then
            tmp = m(i, j);  m(i, j) = m(posI, posJ)
            m(posI, posJ) = tmp
         end if
      end do
  end do

end subroutine setDifferentData


subroutine printMatrix(m, row, col)

  implicit none
```

```
  integer, intent(in) :: row, col
  integer, intent(in) :: m(row, col)

  integer i, j

  do i = 1, row
     do j = 1, col
        print '(i4,$)', m(i, j)
     end do
     print *
  end do

end subroutine printMatrix



subroutine solveProblem(m, row, col)

  implicit none

  integer, intent(in) :: row, col
  integer, intent(in) :: m(row, col)

  integer i, j

  integer posJMin, posIMax, posJMax

  do i = 1, row
     posJMin = 1
     do j = 2, col
        if (m(i, posJMin) > m(i, j)) posJMin = j
     end do
     print '(a,i4,a,i6)', 'Row:', i, ' Min element:', m(i,posJMin)
     if (i == 1) then
        posIMax = i; posJMax = posJMin
     else if (m(posIMax, posJMax) < m(i, posJMin)) then
        posIMax = i; posJMax = posJMin
     end if
  end do

  print *
  print '(a,i6)', 'Max of Min:', m(posIMax, posJMax)
  print '(4x,a,i4)', 'Row:', posIMax
  print '(4x,a,i4)', 'Col:', posJMax

end subroutine solveProblem
```

Задание № 6

Программа, предназначенная для решения этого задания имеет ту же самую структуру, что и программа из **Задания № 5**. Пример подпрограмм, которые собственно и предназначены для решения задания, приведен ниже:

```
subroutine ChangeCols(m, row, col)

  implicit none

  integer, intent(in) :: row, col
  integer, intent(inout) :: m(row, col)

  integer endVal, colBeg, colEnd, i, tmp

  if (mod(col, 2) == 0) then
    endVal = 1
  else
    endVal = 2
  end if
  colBeg = 0;  colEnd = col + 1
  do
    colBeg = colBeg + 1
    colEnd = colEnd - 1
    do i = 1, row
      tmp = m(i, colBeg)
      m(i, colBeg) = m(i, colEnd)
      m(i, colEnd) = tmp
    end do
    if ((colEnd - colBeg) == endVal) exit
  end do

end subroutine ChangeCols


subroutine ChangeRows(m, row, col)

  implicit none

  integer, intent(in) :: row, col
  integer, intent(inout) :: m(row, col)

  integer endVal, rowBeg, rowEnd, j, tmp

  if (mod(row, 2) == 0) then
    endVal = 1
  else
    endVal = 2
  end if
```

```
  rowBeg = 0;  rowEnd = row + 1
  do
    rowBeg = rowBeg + 1
    rowEnd = rowEnd - 1
    do j = 1, col
      tmp = m(rowBeg, j)
      m(rowBeg, j) = m(rowEnd, j)
      m(rowEnd, j) = tmp
    end do
    if ((rowEnd - rowBeg) == endVal) exit
  end do

end subroutine ChangeRows
```

Задание № 7

Программа, предназначенная для решения этого задания имеет ту же самую структуру, что и программа из **Задания № 6**. Пример подпрограмм, которые собственно и предназначены для решения задания, приведен ниже:

```
subroutine SortMatrix(m, row, col)

  implicit none

  integer, intent(in) :: row, col
  integer, intent(inout) :: m(row, col)

  integer posIn, posOut, min, tmp

  do posOut = 1, row - 1
    min = posOut
    do posIn = posout + 1, row
      if (sumRow(posIn) < sumRow(min)) min = posIn
    end do
    if (min /= posOut) then
      do posIn = 1, col
        tmp = m(posOut, posIn)
        m(posOut, posIn) = m(min, posIn)
        m(min, posIn) = tmp
      end do
    end if
  end do

contains

  function sumRow(numRow)

    integer sumRow
    integer, intent(in) :: numRow
```

```
    integer j, tmp

    tmp = 0
    do j = 1, col
      tmp = tmp + m(numRow, j)
    end do

    sumRow = tmp

  end function sumRow

end subroutine SortMatrix
```

Задание № 8

Программа, предназначенная для решения этого задания имеет ту же самую структуру, что и программа из **Задания № 1**. Пример программы приведен ниже:

```
! File Task8mod.f90
module polynomial

  integer, save, private :: row, col
  real, save, private, allocatable :: matr(:,:)

  public create, destroy, setCoeffs, printMatrix, calc
  private setDerivatives

contains

  subroutine create()

    implicit none
    integer error

    do
      print '(5x,a,$)', 'Input Max Power of the Polynomial: '
      read *, col
      if (col > 0) exit
      print '(1x,a)', 'ERROR!! Max power of the polynomial must be positive'
    end do
    row = col

    allocate(matr(0:row, 0:col), stat=error)
    if (error /= 0) stop 'Allocation Memory Error'

  end subroutine create

  subroutine destroy()
```

```
  implicit none
  integer error

   deallocate(matr, stat=error)
   if (error /= 0) stop 'Deallocation Memory Error'

end subroutine destroy

subroutine setCoeffs()

  implicit none
  integer j

  do j = 0, col
    print '(a,i1,a,$)', 'coef at x^',j,': '
    read *, matr(0,j)
  end do

  call setDerivatives()

end subroutine setCoeffs

subroutine setDerivatives()

  implicit none
  integer i, j

  do i = 1, row
     do j = 0, col
       matr(i, j) = j*matr(i-1,j)
     end do
     matr(i, :) = EOSHIFT(matr(i, :), 1)
  end do
end subroutine setDerivatives

subroutine printMatrix()

  implicit none
  integer i, j

  do i = 0, row
    do j = 0, col
      print '(f6.2,$)', matr(i, j)
    end do
    print *
  end do
```

```
  end subroutine printMatrix

  subroutine printPolynomial()

    implicit none
    integer i, j

    do i = 0, row
      print '(i1, a, $)', i, ' derivative: '
      do j = 0, col-1
        print '(f6.2,a,i1,a,$)', matr(i, j), ' x^', j, ' + '
      end do
      print '(f6.2,a,i1)', matr(i, col), ' x^', col
    end do

  end subroutine printPolynomial

  subroutine calc()

    implicit none
    real x, res
    integer i, j

    print '(a,$)', 'Input x value: '
    read *, x

    do i = 0, row
      res = matr(i, 0)
      do j = 1, col
        res = res + matr(i, j)* x**j
      end do
      print '(i1,a,e9.3)', i, ' derivative: ', res
    end do

  end subroutine calc

end module polynomial

! File Task8.f90
program Task8

  use polynomial

  implicit none

  call create()
  call setCoeffs()
```

```
  call printMatrix()
  call printPolynomial()
  call calc()
  call destroy()

end program Task8
```

Задание № 9

Алгоритм решения данного задания аналогичен алгоритму решения **Задания №7** из **Темы № 4**, а структура программы аналогична программе **Задания № 1** этой темы. Для решения задания сначала нужно сделать копию временную исходного массива, удалить из памяти исходный массив, вычислить новые размеры массива, выделить требуемый объем памяти, заполнить новый массив нужными значениями и удалить временный массив из памяти. Пример модульной процедуры, которая собственно и предназначена для решения задания, приведен ниже:

```
subroutine deleteRowCol(numCol, numRow)

    implicit none

    integer(kind=4), intent(in) :: numCol, numRow

    integer(kind=4) :: i, j, error
    integer(kind=4), allocatable :: a(:,:)

    call makeArrayCopy()

    call destroyArray()

    row = row - 1;  col = col - 1

    !print *, "row: ", row, "    col: ", col

    allocate(m(1:row, 1:col), stat=error)
    if (error /= 0) stop 'Allocation memory error'

    do i = 1, numRow-1
       do j = 1, col
          if (j < numCol) then
             m(i,j) = a(i,j)
          else
             m(i,j) = a(i,j+1)
          end if
       end do
    end do

    do i = numRow, row
       do j = 1, col
```

```
            if (j < numCol) then
               m(i,j) = a(i+1,j)
            else
               m(i,j) = a(i+1,j+1)
            end if
         end do
      end do

      call destroyArrayCopy()

   contains

      subroutine makeArrayCopy()

         allocate(a(1:row, 1:col), stat=error)
         if (error /= 0) stop 'Allocation memory error'

         do i = 1, row
            do j = 1, col
               a(i,j) = m(i,j)
            end do
         end do

      end subroutine makeArrayCopy

      subroutine destroyArrayCopy()

         deallocate(a, stat=error)
         if (error /= 0) stop 'Deallocation memory error'

      end subroutine destroyArrayCopy

   end subroutine deleteRowCol
```

Задание № 10

Для вычисления производной воспользоваться конечно – разностными формулами численного дифференцирования по трем точкам:

Конечно – разностная формула дифференцирования вперед

$$f'(x_i)=\frac{f(x_{i+1})-f(x_i)}{h}$$

Конечно – разностная формула дифференцирования назад

$$f'(x_i)=\frac{f(x_i)-f(x_{i-1})}{h}$$

Центральная конечно – разностная формула дифференцирования

$$f'(x_i)=\frac{f(x_{i+1})-f(x_{i-1})}{2h}$$

Здесь h — шаг, с которым построена исходная таблица значений.

Для вычисления значения функции, заданной таблично в точках между узлами таблицы можно воспользоваться методами интерполяции. Задача интерполяции часто формулируется так: пусть функция $y=f(x)$ известна в $N+1$ точке $(x_{0,}y_0)$, $(x_{1,}y_1)$, (x_n,y_n), где $y_k=f(x_k)$, а значения x_k принадлежат интервалу $[a,b]$ и удовлетворяют условиям:

$$a\leqslant x_0<x_1<\ldots<x_N\leqslant b\,.$$

Требуется вычислить значение функции $f(x)$ на интервале $[a,b]$ в промежутках между точками табуляции.

Если заданные точки (x_k,y_k) известны с высокой степенью точности, то можно построить интерполяционный полином $P(x)$, который проходит через них. Когда вычисляется значение интерполяционного полинома $P(x)$ в точке $x\in[x_{0,}x_N]$, то приближение $P(x)$ называется *значением интерполяции*.

Существует много способов построения интерполяционного полинома $P(x)$. Рассмотрим **интерполяционный полином Лагранжа**.

Французский математик *Жозеф Луи Лагранж* (1736 – 1813) предложил использовать для нахождения интерполяционного полинома $P(x)$ следующий метод. Полином, проходящий через $N+1$ точку $(x_{0,}y_0)$, $(x_{1,}y_1)$, … , (x_n,y_n), степени не большей, чем N может быть записан в виде:

$$P_N(x)=\sum_{k=0}^{N}y_k\,L_{N,k}(x)\,,$$

где $L_{N,k}(x)$ — коэффициенты полинома Лагранжа, основанного на этих узлах. Выражения для коэффициентов $L_{N,k}(x)$ могут быть записаны в виде:

$$L_{N,k}(x)=\frac{(x-x_0)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_N)}{(x_k-x_0)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_N)}=\frac{\prod\limits_{j=0,\,j\neq k}^{N}(x-x_j)}{\prod\limits_{j=0,\,j\neq k}^{N}(x_k-x_j)}$$

Для каждого фиксированного k коэффициенты полинома Лагранжа $L_{N,k}(x)$ обладают свойствами:

$$\begin{array}{ll}L_{N,k}(x)=1 & \text{, когда } j=k\,,\\ L_{N,k}(x)=0 & \text{, когда } j\neq k\,.\end{array}$$

Пример организации программы приведен ниже:

```
! File Task10mod1.f90
module DataTable

   real(kind=8), allocatable, private :: dataPoints(:,:)
   integer(kind=4), private :: num
   ! dataPoints(1,i) - x coordinate of point i
   ! dataPoints(2,i) - y coordinate of point i


   public constructDataTable
```

```
    private isFileExists, makeDataTable

contains

    subroutine finalize()

        implicit none

        integer(kind=4) :: error

        deallocate(dataPoints, stat=error)
        if (error /= 0) stop 'Deallocation memory ERROR'
        num = 0

    end subroutine finalize

    ! ============================================
    function isFileExists(fileName)

        implicit none

        character(len=*), intent(in) :: fileName
        logical(kind=4) :: isFileExists

        inquire(file=trim(fileName), exist=isFileExists)

    end function isFileExists

    ! ============================================
    subroutine constructDataTable(fileName)

        implicit none

        character(len=*), intent(in) :: fileName
        character(len=len(fileName)+5) :: newFileName


        if (.not. isFileExists(trim(fileName))) then
            call makeDataTable(fileName)
        end if

        call changeFileName(fileName, newFileName)

        if (.not. isFileExists(trim(newFileName))) then
            call makeDerFile(fileName, newFileName)
        end if
```

```
    call readDataTable(newFileName)

end subroutine constructDataTable


! ========================================
subroutine readDataTable(fileName)

    implicit none

    character(len=*), intent(in) :: fileName
    integer(kind=4) :: error, i
    character(len=1) :: str
    real(kind=8) :: x, y

    num =  numDataLines(trim(fileName))
    allocate(dataPoints(1:2,1:num), stat=error)
    if (error /= 0) stop 'Allocation memory ERROR'

    open(25, file=trim(fileName), status='old', iostat=error)
    if (error /= 0) then
        print 'Error while opening ' // trim(fileName)
        stop  'readDataTable: Program Termination'
    end if

    read(25,*) str
    i = 0
    do
        i = i + 1
        !read(25,*,iostat=error) dataPoints(1,i), dataPoints(2,i)
        !if (error < 0) exit
        read(25,*,iostat=error) x, y
        if (error == 0) then
            dataPoints(1,i) = x;  dataPoints(2,i) = y
        else
          exit
        end if
    end do

    close(25)

end subroutine readDataTable


function numDataLines(fileName)

    implicit none
```

```
        integer(kind=4) :: numDataLines
        character(len=*), intent(in) :: fileName

        integer(kind=4) :: error, num
        character(len=1) :: str
        real(kind=8) :: x

        open(21, file=trim(fileName), status='old', iostat=error)
        if (error /= 0) then
            print *, 'Error while opening ' // trim(fileName)
            stop  'numDataLines: Program Termination'
        end if

        read(21,*) str

        num = 0
        do
            read(21,*,iostat=error) x
            if (error /= 0) exit
            num = num + 1
        end do

        close(21)

        numDataLines = num

    end function numDataLines

    subroutine makeDataTable(fileName)

        implicit none

        character(len=*), intent(in) :: fileName

        real(kind=8) :: xStart, xStop, xStep

        call getData()
        call createFileTable()

    contains

        function tblFunc(x)

            real(kind=8) :: tblFunc
            real(kind=8), intent(in) :: x
```

```
        tblFunc = exp(-x)*sin(x)

    end function tblFunc

    subroutine getData()

        print *, 'Type initial data'
        print '(3x,a,$)', 'start: '
        read *, xStart
        print '(3x,a,$)', 'stop : '
        read *, xStop
        print '(3x,a,$)', 'step : '
        read *, xStep

    end subroutine getData

    subroutine createFileTable()

        integer(kind=4) :: error
        real(kind=8) :: x

        open(21, file=trim(fileName), iostat=error)
        if (error /= 0) then
            print *, 'Error while opening ' // trim(fileName)
            stop  'createFileTable: Program Termination'
        end if

        write(21,'(7x,a1,14x,a4)'), 'x', 'f(x)'
        x = xStart
        do while(x <= xStop + 0.1d0*xStep)
            !write(21,*) sngl(x), sngl(tblFunc(x))
            write(21,'(es14.6,2x,es14.6)') sngl(x), sngl(tblFunc(x))
            x = x + xStep
        end do

        close(21)

    end subroutine createFileTable

end subroutine makeDataTable

! ========================================
function isInRange(testX)

  implicit none

  logical(kind=4) :: isInRange
```

```
  real(kind=8), intent(in) :: testX

  isInRange = (testX >= minval(dataPoints(1,:))) .and. &
              (testX <= maxval(dataPoints(1,:)))

end function isInRange

! ========================================
function getPosition(testX)

  implicit none

  integer(kind=4) :: getPosition
  real(kind=8), intent(in) :: testX

  integer(kind=4) :: i
  logical(kind=4) :: found

  i = 1
  do while (i < num)
    if ((testX >= dataPoints(1,i)) .and. (testX < dataPoints(1,i+1))) then
      found = .true.
      exit
    end if
      i = i + 1
  end do
  if ((.not. found) .and. testX == dataPoints(1,num)) then
    found = .true.
    i = num
  end if

  if (found) then
    getPosition = i
  else
    getPosition = -1
  end if

end function getPosition

! ========================================
function startNumber(testX, ptNum)

  implicit none

  integer(kind=4) :: startNumber
  real(kind=8), intent(in) :: testX
  integer(kind=4), intent(in) :: ptNum
```

```
  integer(kind=4) :: findNum

  if (isInRange(testX)) then
    findNum = getPosition(testX)
    if (findNum < 1+ptNum/2) then
      startNumber = 1
    else if (findNum > (num - ptNum + 2*(ptNum/2))) then
      startNumber = num - ptNum
    else
      startNumber = findNum - ptNum/2
    end if
  else
    print '(3x,f4.2,a,f6.2,a,f6.2,a)', testX, ' is not in [', &
          minval(dataPoints(1,:)), ', ', maxval(dataPoints(1,:)), ']'
    stop 'Program Terminated'
  end if

end function startNumber

! ========================================
function lagrInterp(testX, ptNum)

    implicit none

    real(kind=8) :: lagrInterp

    real(kind=8), intent(in) :: testX
    integer(kind=4), intent(in) :: ptNum

    integer(kind=4) :: j, k, start
    real(kind=8) :: numer, denom

    start = startNumber(testX, ptNum)

    lagrInterp = 0.0d0
    do k = start, start + ptNum
      numer = 1.0d0
      denom = 1.0d0
      do j = start, start + ptNum
        if (k /= j) then
          numer = numer * (testX - dataPoints(1,j))
          denom = denom * (dataPoints(1,k) - dataPoints(1,j))
        end if
      end do
      lagrInterp = lagrInterp + dataPoints(2,k)*numer/denom
    end do
```

```
    end function lagrInterp


    ! ========================================
    subroutine makeDerFile(fileNameRead, fileNameWrite)

        implicit none

        character(len=*), intent(in) :: fileNameRead, fileNameWrite

        integer(kind=4) :: num, i, error
        real(kind=8) :: x1, f1, x2, f2, x3, f3
        character(len=1) :: str

        num = numDataLines(trim(fileNameRead))

        open(21, file=trim(fileNameRead), status='old', iostat=error)
        if (error /= 0) then
            print *, 'Error while opening ' // fileNameRead
            stop  'makeDerFile: Program Termination'
        end if
        open(22, file=trim(fileNameWrite), iostat=error)
        if (error /= 0) then
            print *, 'Error while opening ' // fileNameWrite
            stop  'makeDerFile: Program Termination'
        end if

        read(21,*) str
        write(22,'(7x,a1,13x,a5)'), 'x', "f'(x)"

        read(21,*) x1, f1
        read(21,*) x2, f2
        write(22,*) sngl(x1), sngl((f2 - f1)/(x2 - x1))
        do i = 3, num
          read(21,*,iostat=error) x3, f3
          if(error /= 0) stop 'Error while reading the file'
          write(22,*) sngl(x2), sngl((f3 - f1)/2.0d0/(x2 - x1))
          x1 = x2; f1 = f2
          x2 = x3; f2 = f3
        end do
        write(22,*) sngl(x3), sngl((f2 - f1)/(x2 - x1))

        close(21); close(22)

    end subroutine makeDerFile
```

```
  subroutine changeFileName(fileName, newFileName)

      implicit none

      character(len=*), intent(in) :: fileName
      character(len=*), intent(out) :: newFileName

      integer(kind=4) :: posLastPoint

      posLastPoint = index(fileName, '.', .true.)

      if (posLastPoint == 0) then
         newFileName = trim(fileName) // 'Deriv'
      else
         newFileName= fileName(1:posLastPoint-1) // &
                      'Deriv' // fileName(posLastPoint:len_trim(fileName))
      end if

   end subroutine changeFileName

end module DataTable


! File Task10.f90
program Task10

   use DataTable

   implicit none

   character(len=20) :: fileName
   real(kind=8) :: x

   print *, 'Type Name of File: '
   print '(3x,a,$)', '-> '
   read '(a)', fileName

   call constructDataTable(trim(fileName))

   x = 3.766d0
   !x = 1.5d0
   print *, lagrInterp(x, 3)
   print *, analytDeriv(x)

   call finalize()
```

```
contains


    function analytDeriv(x)

        real(kind=8) :: analytDeriv
        real(kind=8), intent(in) :: x

        analytDeriv = (cos(x) - sin(x))*exp(-x)

    end function analytDeriv

end program Task10
```