Введение

“Современный Фортран” – в последнее время вышло немало книг именно с этими словами в названии. Спрашивается, почему? Оказывается, появился спрос на этот Фортран. Какой же он, этот Фортран?

- это стандартизованный язык Фортран, версии 66, 77, **90, 95,** 03, 08;
- это язык сегодняшнего дня, современный;
- в Фортране Вы можете писать программы на любой версии языка, например, кое-где преподают даже Фортран-77;
- это язык, для которого есть современные компиляторы, например MS Fortran, Intel Fortran Compiler, GNU-Fortran, Plato IDE;
- это язык, для которого налажено обучение;
- это версия языка, которая уже достаточно хорошо освоена;
- это язык, для которого есть возможность автоматически строить параллельные приложения, например, в Intel Fortran Compiler;
- это язык, на котором пишут реальные программы, в частности, для современных компьютеров и суперкомпьютеров;
- итак, это **Фортран-95**.

Традиционно задачу на компьютере решают таким образом: используя библиотечные процедуры, составить алгоритм, продумать ввод и вывод, написать программу, скомпилировать, собрать, отладить, выполнить и получить результаты. В Фортране предлагается так много нового, что даже не верится, что `Fortran` - это первый язык программирования. Название языка произошло от **For***mula* **Tran***slator* – переводчик формул. В истории программирования Фортран – это живой классический язык. Не одно поколение программистов выросло на Фортране. Если кому-то кажется, что Фортран - из прошлого, то посмотрите на его новейшие возможности. Например, то, что предлагается для суперкомпьютеров, - это как раз из будущего. Фортран критиковали за «примитивность», но именно простота помогает ему жить и развиваться, сохраняя преемственность и проходя стандартизацию:

- 1954, Фортран – первый язык программирования, Джон Бэкус, IBM;
- Ф66, первый стандарт языка – универсальность и преемственность;
- Ф77, не первым, но присоединился к структурному программированию;
- Ф90/Ф95, *современный Фортран* стал вехой в развитии языка:
  - новое описание массивов, функции редукции, конструкторы, секции, конформность массивов, выражений, ветвлений и циклов;
  - модульное программирование, механизмы передачи данных;
- Ф03, Ф08 – объектно-ориентированное программирование, *coarrays*.

Фxx – Фортран по годам выхода стандартов, это язык программирования, стабильный, развивающийся, нацеленный в будущее, только в нём:

- история насчитывает 58 лет, чего попросту нет у других языков;
- автоматизировано построение параллельных приложений;
- сочетается строгая классика в изучении основ программирования и конструирование сверхсложных проектов при помощи модулей;

- действия над векторами и матрицами внешне выглядят так же, как со скалярами, с комплексными числами - так же, как с вещественными;
- накоплены численные библиотеки (IMSL, NAG, LAPACK, BLAS, Intel MKL), с высокопроизводительными вычислениями (MPI, PVM), с графическими интерфейсами (Quickwin, FORTRAN/TK);
- наиболее популярные компиляторы Фортрана: *по стандарту* Ф95, Ф03 – это *Intel Fortran Compiler,* версии ifc-9, ifc-10, ifc-11, ifc-12 для многоядерных компьютеров и суперкомпьютеров, свободно распространяется для Linux; Plato Silverfrost Ltd. Studio; *по стандарту* Ф90-Ф95 – Fortran Power Station 4.0 и 5, 6 фирмы Compaq; GNU Fortran.

О Фортране в Интернете:

- www.fortran.com,
- www.npac.syr.edu ,
- netlib.org – набор библиотек,
- www.j3 – fortran.org – стандарт языка,
- наш сайт – twcad.ifmo.ru,
- parallel.ru/tech/tech_dev/newfortran.html,
- free compilers –www.thefreecountry.com/compilers/fortran.shtml.

Практикум подразумевает лабораторные, задания, вопросы, кратко теория.

1. Инструкции для работы в IDE Studio

Integrated Developer Studio (IDE) – это интегрированная студия разработки программ. Обычно это универсальная среда, рассчитанная на разработку программ в разных языковых системах. Она включает хороший текстовый редактор с подсветкой ключевых слов, компоновщик и отладчик, поддерживает программы в старой фиксированной форме *.for, а также в свободной форме - расширение *.f90 или *.f03.

Приведём краткое описание перечисленных ниже трёх сред для начинающего разработчика.

MS Developer Studio, 1996г – компактная (всего 110 Мб) среда разработки программ Fortran, C++. Устанавливается за 5 минут и требует всего 110 Мб – это общий объём Developer Studio, компилятора для языка Fortran, трёх учебников и помощи. Недостатки: устарел отладчик, Debug работает только в среде Windows-XP. В других, более новых ОС приходится работать без Debug, только в Release. Среда не обеспечивает построения параллельного приложения. MS Developer пошла по рукам и стала менять хозяев: DEC, HP.

MS Visual Studio, 2010г – некомпактная среда разработки программ. Отладчик Debug работает в среде Windows-7 и Vista. Сначала устанавливают Visual Studio (400 Мб) и, если надо, компиляторы C++ и C#, затем компилятор фирмы Intel. Intel Fortran Compiler поставляется в версиях ifc-9, ifc-10, ifc-11, ifc-12 (до 500 Мб.). Среды ifc-9, ifc-10, ifc-11, ifc-12 обеспечивают уникальное автоматическое построение параллельного приложения. Есть версии компилятора и для Windows, и для Linux.

Plato Silverfrost Ltd. Studio, 2011г – это среда разработки программ. Поддерживается Fortran, C++. Отладчик Debug работает в среде Windows-7. Строится на удивление компактное приложение. Компилятор не позволяет пользоваться кириллицей не только в именах файлов и папок, но даже в командах вывода и комментариях.

GNU Fortran – это свободно распространяемый Fortran. Есть версии и для Windows, и для Linux.

Для размещения проектов рекомендуется создать каталог дня Ваших проектов, например, *d:/PROJ,* и при этом не пользоваться кириллицей в именах файлов и папок.

1.1. Инструкция по работе в DevStudio, FPS 4.0

Компактная (110 Мб) среда разработки программ MS Developer Studio (*DevStudio*) используется для обучения Fortran Power Station (*FPS 4.0*) и Си++.

Студия *DevStudio* снабжена текстовым редактором, компиляторами, сборщиком, отладчиком, который работает только в Windows-XP. В чистовом варианте *Release* компилятор работает под Windows-Vista, Windows-7. *Debug|Release* переключает варианты создания приложения:

- *Release* – чистовой вариант приложения (выполнение <Ctrl>+<F5>);
- *Debug* – отладочный вариант приложения (выполнение <F5>);
- переключитесь в режим *Release*, потому что режим по умолчанию *Debug* устарел и работает только в среде Windows-XP.

Работа начинается с создания проекта, в именах папок и файлов не пользуйтесь кириллицей. Создайте новый проект *d:/PROJ/myProj* через *File|New|Project WorkSpace|Console Application*:

- задайте сверху справа имя проекта myProj*;*
- внизу через <*Browse*> выберите место для размещения проекта *myProj* в *d:/PROJ*;
- нажмите кнопку <*Create*> – создать проект, затем сохраните проект по кнопке <*Save all*>;
- чтобы закончить работу с программой и сохранить всю информацию, достаточно закрыть приложение, щелкнув по крестику в верхнем правом углу окна приложения.

Варианты включения программы в состав проекта d:/*PROJ/myProj*:

1. включить в проект ранее написанную программу *lab.f90* через меню *Insert | File into project* ==> выбрать путь d:/PROJ/*myProj*;
2. создать заново программу *Lab.f90* в *DevStudio*, включив её в проект:
   - выбрать в меню *File|New|Text*;
   - набрать текст программы;
   - *File|Save as* ==> задать имя программы *Lab1.f90*;
   - щелкнуть на панели инструментов по кнопке *Compile* для первой компиляции, и на вопрос «Включать ли программу в проект?» - дать утвердительный ответ.

При повторном входе продолжается последний проект, для которого Вы можете создать этикетку для запуска проекта с рабочего стола.

Удобно помимо программы включать в проект следующие тексты с расширением *.txt:

- исходные данные создать, как новый текст через меню *Insert | File into project* ==> выбрать *myProj/in.txt*, тогда при выполнении программы данные читаются из файла в текущем каталоге проекта с именем, указанным в операторе *open*;
- текст результатов под именем, указанным в операторе *open.*

Тексты программы и результатов составят распечатку для отчета.

Вызов FPS40 – четырехцветный ярлычок *MS Developer Studio.*

Варианты запуска программы на выполнение без отладки (любая Windows):

- по этикетке Вашего приложения на рабочем столе;
- <Ctrl>+<F5>, либо через меню *Build\execute Lab1.exe* с паузой в конце для просмотра консоли.

Варианты запуска программы на выполнение в отладке (в Windows-XP):

- после коррекции щелкните по <F5>, и *FPS40 компилирует, компонует и выполняет программу* в отладочном режиме;
- то же, что <F5>, на панели инструментов «стрелка вниз» (всплывающая надпись на кнопке «go»).

Дополнительные возможности по отладке (в Windows-XP):

- <F8> – пошаговая отладка,
- <F10> – вместо <F8>, чтобы не «нырять» в процедуру,
- <F7> – идти в отладке до места, указанного курсором,
- чередование <F7> , <F8>, <F10> при «хитрой» отладке,
- мышку навести на переменную – появится её значение.

1.2. Инструкция по работе в Visual Studio

Новая среда для разработки программ MS Visual Studio (600 Мб-1000 Мб) используется для обучения Fortran, Си++ и C#. Это MS Visual Studio, C++, C# плюс Intel Fortran compiler (500 Мб).

Среда Visual Studio снабжена текстовым редактором, компиляторами, сборщиком, отладчиком. Работа начинается с создания проекта:

- создайте новый проект в каталоге с именем *File|New|New Project|Intel Visual Fortran|Console Application*;

- задайте внизу имя проекта в поле *Name;*

- внизу через <*Browse*> выберите место *Location* для размещения проекта *myProj*;

- нажмите кнопку <*Ok*> – создать проект.

Варианты включения программы в состав проекта по правой кнопке мыши:

1. включить в проект ранее написанную программу *lab.f90* через меню *Source|File project* ==> выбрать *lab.f90*;
2. создать текст программы *lab.f90* в *Visual Studio* и включить в проект.

Создание текста программы *lab.f90* в *Visual Studio:*

- выбрать правой кнопкой мыши в меню *Source|New|Text*;
- набрать текст программы;
- задать имя программы *lab1.f90* и выбрать место для размещения программы *myProj*;
- щелкнув правой кнопкой мыши, выбрать в контекстном меню *Compile* для компиляции.

Удобно помимо программы включать в проект следующие тексты:

- исходные данные, которые можно создать, как новый текст через меню *Insert|File into project,* чтобы *myProj/in.txt* при выполнении программы читались, как данные из файла в папке проекта с именем, указанным в операторе *open*;
- текст результатов под именем, указанным в операторе *open.*

Всем текстам, кроме программы, давайте имена, используя расширение *txt*, чтобы они были доступны в любом текстовом редакторе.

Тексты программы и результатов составят распечатку для отчета.

Вызов среды проектирования - это ярлычок MS Visual Studio. Чтобы закончить работу с программой и сохранить всю информацию, достаточно закрыть приложение, щелкнув по крестику в верхнем правом углу окна приложения.

При повторном входе надо открыть последний проект.

Варианты создания приложения – переключатель *Debug|Release*:

- *Debug* – отладочный вариант приложения (выполнение <F5>) ;
- *Release* – чистовой вариант приложения (выполнение <Ctrl>+<F5>).

Варианты запуска программы на выполнение:

- после коррекции щелкните на панели инструментов по “стрелке вправо” – *компилируется, компонуется и выполняется программа* в отладочном режиме;
- то же самое, что “стрелка вправо”, <F5> на панели инструментов.

Дополнительные возможности по отладке программы:

- *Start Debug* – продолжить программу (выполнение <F5>) ;
- <F10> – пошаговая отладка, без «входа» в процедуры;
- <F11> – вместо <F10>, чтобы не «нырять» в процедуры;
- по правой кнопке мыши *Run to Cursor* – идти в отладке до места, указанного курсором;
- чередуя <F10> , *Run to Cursor*, <F11>, <F5> при «хитрой» отладке;
- мышку навести на переменную – появится её значение;

Отладочный вариант приложения (*Debug*) работает под Windows-Vista и Windows-7.

Дополнительные возможности без отладки:

- выполнение <Ctrl>+<F5>;
- либо через меню *Build|execute lab1.exe* с паузой для просмотра консоли.

1.3. Инструкция по работе в Plato IDE Studio

Среда разработки программ Plato IDE Studio для обучения Fortran. Студия Plato IDE снабжена текстовым редактором, компиляторами, сборщиком, отладчиком. Работа начинается с создания проекта:

- создайте папку *d:/PROJ* для хранения проектов;
- в папке проектов *d:/PROJ* создайте папку *myProj* под новый проект;
- в одной папке может быть лишь один проект.

Создайте новый проект в каталоге с именем *myProj* *File|New Project|Fortran Application*:

- задайте снизу имя проекта в поле *Name;*
- через <*Browse*> выберите место *Location* для размещения проекта *myProj* в папке *d:/PROJ* и нажмите кнопку <*Open*>, чтобы создать проект.

Варианты включения программы в состав проекта:

1) создать новый текст программы *Lab.f90* в Plato IDE Studio и включить в проект по правой кнопке мыши;
2) включить в проект ранее написанную программу *labac.f90* через меню *Source* Files | *Add existing item* ==> выбрать *labac.f90*.

Создание нового текста программы *Lab.f90* по правой кнопкой мыши в *Project Explorer* пункт *Source* Files | *New Item:*

- задайте снизу имя программы в поле *Name;*
- через <*Browse*> выберите место *Location* для размещения программы *lab.f90* в папке *d:/PROJ/myProj* и нажмите кнопку *Open*, чтобы создать программу;
- набрать новый текст программы;
- щелкнуть по правой кнопке мыши – и по кнопке *Compile* выполнить компиляцию.

Удобно помимо программы включать в проект следующие тексты:

- исходные данные через меню Source Files|Add existing item ==> выбрать *myProject/in.txt*, тогда при выполнении программы данные будут читаться из файла в текущем каталоге проекта с именем, указанным в операторе *open*;
- текст результатов – под именем, указанным в операторе *open.*

Все тексты, кроме программы, именуйте, используя расширение *txt*, чтобы они были доступны в любом текстовом редакторе. Тексты программы и результатов составят распечатку для отчета.

Вызов – ярлычок Plato IDE. Чтобы закончить работу с программой и сохранить всю информацию, достаточно закрыть приложение, щелкнув по крестику в верхнем правом углу окна приложения. При повторном входе надо загрузить последний проект.

Варианты создания приложения – переключатель *Debug|Release*:

- *Debug* – отладочный вариант приложения, выполнение <F7>;
- *Release* – чистовой вариант приложения, выполнение <Ctrl>+<F5>.

Варианты запуска программы на выполнение:

- *Step Info, отладка в пошаговом режиме,* после коррекции программы щелкните по клавише <F7>, чтобы *скомпилировать, скомпоновать и выполнить программу* в отладочном режиме, в отдельном окне;
- *Start run*, или, что то же самое <Ctrl>+<F5>, чтобы запустить программу без отладки, в чистовом варианте;

Дополнительные возможности по отладке программы:

- <F6> – *Continue,* продолжить программу;
- <F7> – *Step Info,* пошаговая отладка,
- <F8> – *Step Over,* пошагово, не «ныряя» в процедуры,
- <Ctrl>+<F10> – *Goto Cursor,* идти в отладке до места, указанного курсором,
- чередуя <F7> , <F8>, <Ctrl>+<F10> при «хитрой» отладке,

- отладочный вариант приложения (*Debug*) работает под Windows-Vista и Windows-7.

2. Лабораторные работы

2.1. Вычисления по формулам

Задание

Составить программу, которая:

1. вычисляет значения двух эквивалентных пар числовых формул $y_1 \sim y_2$ и $z_1 \sim z_2$, с указанными в варианте индивидуального задания значениями исходных данных.
2. выводит в файл *out.txt* исходные данные и результаты вычислений.

Примечания:

a) *математически* две числовые формулы эквивалентны, если для всех возможных значений переменных их значения равны;

b) *эквивалентными на компьютере* будем считать значения, совпадающие до 6-7 десятичного знака с одинарной точностью `real`;

c) функции в формулах задания нельзя заменять другими, кроме тех, которых нет среди встроенных функций, как *sec x*;

d) формулы задания нельзя упрощать, но рекомендуется вводить вспомогательные (промежуточные) переменные.

Содержание отчета

1. Название работы и номер варианта индивидуального задания.
2. Фамилия, имя, отчество и номер группы студента.
3. Текст задания, формулы в том виде, как они приведены в варианте индивидуального задания.
4. Области допустимых значений (ОДЗ) переменных для y_1, y_2.
5. Математические формулы для промежуточных переменных (если они используются в программе).

6. Распечатки текста программы и результатов работы программы с тремя комплектами исходных данных.

   При подготовке к защите ответить на контрольные вопросы.

Справочная информация

1. Порядок выполнения операций в соответствии их старшинством (приоритетом) показан в Табл.1.

Таблица 1.

Порядок выполнения операций в соответствии их старшинством

Операции	**Знаки операций**	**Старшинство**
Вычисление функций	Ссылки на функции	1
Возведение в степень	**	2
Умножение, деление	*, /	3
Сложение, вычитание	+, –	4

Примечание: порядок выполнения операций изменяют скобки (круглые).

2. Реализация некоторых математических выражений средствами Фортрана показана в Табл. 2; *x - вещественный* аргумент **Sqrt***(x)* и тригонометрических функций.

Таблица 2.

Реализация некоторых математических выражений средствами Фортрана

В формуле	**В Фортране**	**В формуле**	**В Фортране**	**В формуле**	**В Фортране**
sin *x*	**Sin**(x)	Tg *x*	**Tan**(x)	$\sqrt{2}$	Sqrt(2.)
cos *x*	**Cos**(x)	Ctg *x*	**Cotan**(x)	$\sqrt{x}$	при *x*>0 **Sqrt**(x) при *x*<0 **Sqrt**(**cmplx**(x))
sec *x*	1/**Cos**(x)	arcsin x	**Asin**(x)	$\sqrt[n]{x}$	при *x*>0 x**(1./n) при *x*<0 и *n* нечетн. -(-x)**(1./n) при *x*<0 и *n четном* **cmplx**(x)**(1./n)
|*b*|	**Abs**(b)	π	2***Asin**(1.)		

3. Тип результата числовой операции (целочисленный или вещественный) выбирается автоматически по типу операндов, особо важно понимание этого механизма для операции деления:

a) если операнды *целые,* в результате взятия целой части частного получается целое число (для 7/4 => 1);
b) если операнды *вещественные,* в результате получается вещественное число (для 7./4. => 1.75);
c) если операнды *разных типов*, например, целого и вещественного, перед выполнением операции они приводятся к типу, который соответствует наиболее широкому классу (диапазону) чисел (для 7./4 => 7./4. – к вещественным числам); затем выполняется операция для этого типа с соответствующим результатом: (7./4 => 7./4. => 1.75).

4. Выполнение операции a**b – возведения в степень a^b зависит от типа показателя *b*:
   a) для `integer` *b* выполняется *b*-кратное умножение основания *a*;
   b) для *вещественного* значения *b* при *a*>0 $a^b = e^{b\cdot\ln a}$.

Внимание! Отрицательное основание не возводят в вещественную степень.

Комментарии к заданию

1. Имена переменных предпочтительно выбирать со смыслом (например, длина окружности $C=2\pi R$: `Circle=2*Pi*Radius`).
2. Громоздкие формулы рекомендуется упрощать, вводя промежуточные переменные для частей формулы: повторяющиеся части формулы, числители и знаменатели дробей и так далее.
3. Восстановив формулу по выражению в программе, найдете ошибки.
4. *Типичные ошибки в арифметических выражениях* – *см. раздел 8.3.*

Пример

Исходное значение аргумента из ОДЗ: x=3.3

$$y_1 = \frac{x^2 + 2x - 3 + (x+1)\sqrt{x^2-9}}{x^2 - 2x - 3 + (x-1)\sqrt{x^2-9}}; \quad y_2 = \frac{\sqrt{x+3}}{\sqrt{x-3}}$$

Область допустимых значений (ОДЗ)

$x > 3$

Промежуточные переменные

Повторяющаяся часть формулы: $s1 = \sqrt{x^2-9}$

Знаменатель формулы y_1: $yd = x^2 - 2x - 3 + (x-1) * s1$

Программа

```
Program  Formula ! тема: вычисления по формулам
! студент (фамилия, имя)  группа №  работа №  вариант №
Implicit None ! переменные должны быть объявлены без умолчаний
Real::x=3.3, y1,y2,yd,s1  ! [pro1] инициализация переменной  x (из задания)
        ! <== потом z1, z2 - объявить здесь же
```

```
Open (6,file='result.txt') ! закомментировать «open» - вывод на экран
![pro2] диалог по консоли: пригласить Write(*,*)'x=?' читать Read(*,*)x
! [pro3] Open(1,file='input.txt') – для чтения x из файла №1 Read(1,*) x
 Write(6,*) 'at x=',x
 s1 = Sqrt(x**2-9) ! повторяющаяся часть формулы
 yd = x**2 - 2*x - 3 + (x-1)*s1 ! знаменатель y1
 y1 = (x**2 + 2*x - 3 + (x+1)*s1) / yd
 Write(6,*) 'y1=', y1
 y2 = Sqrt(x+3) / Sqrt(x-3)
 Write(6,*)'y2=', y2
 ! OK: в y1 и y2 совпали 6 – 7 значащих цифр
End Program Formula !<== далее – добавить вычисление z1,z2
```

Последовательность работы над программой

Проект [pro1] с `Real`:: x=3.3 приведен в программе, а [pro2], [pro3], показаны комментариями.

Составить три проекта:

- [pro1] - предъявить на ПК;
- [pro2] - предъявить на ПК с данными, введенными в диалоге;
- [pro3] - распечатать проект с данными из файла (3 комплекта чисел);

Определить область допустимых значений (ОДЗ) исходных данных. Отладить программу с данными, указанными в задании [pro1].

[pro1] Отладка завершена, если в *y1* и *y2* совпали 6 - 7 цифр.

[pro2] Организовать диалог по вводу исходных данных:

a) пригласить к вводу данных `Write(*,*) 'x>3 x=?'`
b) добавить в программу ввод данных с консоли `Read`(*,*) x
c) вычислить и вывести результирующие значения *y1* и *y2*.

[pro3] Взять данные из ОДЗ, подготовив файл *in.txt* :

a) открыть файл *in.txt* для чтения данных `Open`(1,file='in.txt');
b) используя цикл, ввести данные и получить 3 комплекта ответов

```
do k=1,3
```

- ввести один комплект данных из файла *in.txt*;
- вычислить *y1* и *y2*, *z1* и *z2*;
- вывести результаты с указанием имен переменных;

```
enddo
```

Контрольные вопросы к защите работы

1. Что называется программой на алгоритмическом языке?
   а) файл на диске; б) файл, внутри которого есть слово *program*;
   в) дайте свой развернутый вариант ответа.
   Примечание. Вы просматриваете на экране текстовый файл. Можете ли Вы утверждать, что это файл с программой?
2. Вы работаете с компилятором, опишите что должно обязательно присутствовать в имени файла с программой на Фортране 90:

а) хотя бы одна буква; б) слово *fortran*; в) свой вариант ответа – приведите примеры имен файлов с программами на Фортране 90.

3. Как записать комментарий в программе?
   а) начать его с красной строки; б) начать его с «//» ; в) дайте свой вариант ответа. Приведите примеры комментариев.
4. Какие числовые типы данных используют в Фортране? Приведите примеры числовых констант.
5. Можно ли вещественную константу 6420 записать на Фортране без десятичной точки? Если можно, то как это сделать?
6. Как записать на Фортране комплексное число *19+4i* ?
7. Какие имена переменных в программе записаны верно, а какие нет?
   а) *F1* б) *Y(X)* в) *X_1* г) *Б5* д) *Z.8* е) *3J* ж) *β4*
8. Как правильно вызвать функцию *sinx* ?
   а) sinX б) sin*x* в) sin(x)
9. Определите значение переменной *M* в результате вычисления:
```
Integer :: N=1, M;    M = 1/((2*N+1)*(2*N+2))
```
10. В каком порядке выполняются операции в числовом выражении?
    а) со скобками; б) без скобок. Разъясните оба случая.
11. Определите значение переменной *B* в результате вычисления:
```
real:: A=2.0, B;      B = -A**2
```
12. Исправьте ошибки:
    a) $\sqrt{4}$ записан как **sqrt**(4);
    b) вещественный корень $\sqrt{4}$ записан как **sqrt** (4,0) ;
    c) комплексный корень $\sqrt{-4}$ записан как **sqrt** (-4);
    d) комплексный корень $\sqrt{-4}$ записан как **sqrt** (-4.0);
    e) комплексный корень $\sqrt{-4}$ записан как **sqrt** (-4,0) .
13. Отрицательное число не возводят в вещественную степень, почему?
14. Записать в Фортране известные величины – дюжина, число π, число *e*, скорость света $c = 3.10^8$:
    а) в виде констант; б) как значения, точные для компьютера.
15. Определите значения переменных *B* и *C* в результате вычисления:
```
real:: A=2.0, B,C;  B = 1/2*A;    C = 1/(2*A)
```
16. Как формулу $\sqrt[3]{-1/8}$ записать в Фортране?

Указание. Правильность ответов на вопросы: (9), (11), (15), (16) проверьте на компьютере.

Пример контрольного задания

1. Расположите константы в порядке возрастания (представить в одинаковой форме): `24.0;  2.4E+2;  0.24E-3`
2. *b*=4.0 – вещественная переменная. Что получим в результате вычисления: *1/2*b; 1/(2*b); b**(1/2); 1/b*2* .

3. Формула $\sqrt[5]{x}$ была записана как `x**1/5` – исправьте ошибки.
4. Запишите на Фортране формулу $\sqrt[-5]{3\sin x + 4\cos^2 x^2 - \frac{1}{2x}}$
5. Чему будут равны значения переменных m, n, k, c, c1 после выполнения программы? В ответах учесть тип переменных.

```
integer:: m, n, k;     real:: a=7.2, b=1.8, c, c1
m=a;  n=b;  k=a/b+b;  c = a/b+b;  c1 = m/n+b
```

Таблица 3.

Варианты индивидуальных заданий «Формулы»

№	Данные	Формулы
1	a=8.6 b=1.3 c=3.3 α=0.75	$y_1 = \left(a^2 - b^2 - c^2 + 2bc\right) : \frac{a+b-c}{a+b+c}$; $y_2 = (a+c)^2 - b^2$; $z_1 = 2\sin^2(3\pi - 2\alpha)\cdot\cos^2(5\pi + 2\alpha)$; $z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$
2	a=3.5 b=-2.1 α=0.1	$y_1 = \frac{a^2 - b^2}{a - b} - \frac{a^3 - b^3}{a^2 - b^2}$; $y_2 = \frac{ab}{a+b}$; $z_1 = \frac{\cos 2\alpha}{ctg^2\alpha - tg^2\alpha}$; $z_2 = \frac{1}{4}\sin^2 2\alpha$
3	a=3.5 b=0.72 α=0.62	$y_1 = \frac{\left(\sqrt{a^2 + a\sqrt{a^2 - b^2}} - \sqrt{a^2 - a\sqrt{a^2 - b^2}}\right)^2}{2\sqrt{a^3 b}\left(\sqrt{\frac{a}{b}} + \sqrt{\frac{b}{a}} - 2\right)}$; $y_2 = \frac{\left(\sqrt{a} + \sqrt{b}\right)^2}{a - b}$; $z_1 = 1 - \frac{1}{4}\sin^2 2\alpha + \cos 2\alpha$; $z_2 = \cos^2\alpha + \cos^4\alpha$
4	a=4.3 α=0.43	$y_1 = \frac{a^3 - 3a^2 + 4 + (a^2 - 4)\sqrt{a^2 - 1}}{a^3 + 3a^2 - 4 + (a^2 - 4)\sqrt{a^2 - 1}}$; $y_2 = \frac{(a-2)\sqrt{a+1}}{(a+2)\sqrt{a-1}}$; $z_1 = \cos\alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha$; $z_2 = 4\cos\frac{\alpha}{2}\cdot\cos\frac{5}{2}\alpha\cdot\cos 4\alpha$

№	Данные	Формулы
5	m=0.4 n=2.1 α=0.43	$y_1 = \frac{(m\text{-}1)\sqrt{m} - (n-1)\sqrt{n}}{\sqrt{m^3 n} + mn + m^2 - m}$; $y_2 = \frac{\sqrt{m} - \sqrt{n}}{m}$; $z_1 = \frac{\operatorname{tg}\alpha - \sec\alpha}{\cos\alpha - \operatorname{ctg}\alpha}$; $z_2 = \operatorname{tg}\alpha \cdot \sec\alpha$
6	a=15.1 α=1.23	$y_1 = \left(\frac{1+a+a^2}{2a+a^2} + 2 - \frac{1\text{-}a+a^2}{2a\text{-}a^2}\right)^{-1}\left(5-2a^2\right)$; $y_2 = \frac{4-a^2}{2}$; $z_1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha$; $z_2 = 2\sqrt{2}\cos\alpha \cdot \sin\left(\frac{\pi}{4} + 2\alpha\right)$
7	a=12.3 α=0.43	$y_1 = \left(\frac{a+2}{\sqrt{2a}} - \frac{a}{\sqrt{2a}+2} + \frac{2}{a-\sqrt{2a}}\right)\cdot\frac{\sqrt{a}-\sqrt{2}}{a+2}$; $y_2 = \frac{1}{\sqrt{a}+\sqrt{2}}$; $z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha + 1 - 2\sin^2 2\alpha}$; $z_2 = 2\sin\alpha$
8	x=3.2, y=0.8 α=0.81	$y_1 = \frac{x}{x^2+y^2} - \frac{y(x-y)^2}{x^4-y^4}$; $y_2 = \frac{1}{x+y}$; $z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\alpha}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\alpha}{4}\right)$; $z_2 = \frac{\sqrt{2}}{2}\sin\frac{\alpha}{2}$
9	x=1.4 y=2.8 α=0.66 β=0.82	$y_1 = \frac{x^{\frac{2}{3}} + 2\cdot\sqrt[3]{xy} + 4y^{\frac{2}{3}}}{\left(\sqrt[3]{x^4} - 8y\cdot\sqrt[3]{x}\right):\sqrt[3]{xy}}$; $y_2 = \frac{1}{\sqrt[3]{\frac{x}{y}} - 2}$; $z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2$; $z_2 = -4\sin^2\frac{\alpha-\beta}{2}\cdot\cos(\alpha+\beta)$
10	a=2.3 α=0.75	$y_1 = \frac{\sqrt{(2a+1)^3} + \sqrt{(2a-1)^3}}{\sqrt{4a + 2\sqrt{4a^2-1}}}$; $y_2 = 4a - \sqrt{4a^2-1}$; $z_1 = \frac{\sin\left(\frac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)}$; $z_2 = \operatorname{ctg}\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right)$

№	Данные	Формулы
11	a=0.7 x=0.44 y=0.82	$y_1=\frac{1}{2\left(1+\sqrt{a}\right)}+\frac{1}{2\left(1-\sqrt{a}\right)}-\frac{a^2+2}{1-a^3}$; $y_2=\frac{-1}{a^2+a+1}$; $z_1=\cos^4 x+\sin^2 y+\frac{1}{4}\sin^2 2x-1$; $z_2=\sin(y+x)\cdot\sin(y-x)$
12	a=5.1 α=0.1	$y_1=\frac{\sqrt{a}+1}{a\sqrt{a}+a+\sqrt{a}}:\frac{1}{a^2-\sqrt{a}}$; $y_2=a-1$; $z_1=\frac{\sin\alpha-\sin3\alpha+\sin5\alpha}{\cos\alpha-\cos3\alpha+\cos5\alpha}$; $z_2=\mathrm{tg}3\alpha$
13	a=5.3 b=2.1 α=0.75	$y_1=\frac{\left(a^2-b^2\right)\cdot\left(\sqrt[3]{a}-\sqrt[3]{b}\right)}{\sqrt[3]{a^4}+\sqrt[3]{ab^3}-\sqrt[3]{a^3b}-\sqrt[3]{b^4}}$; $y_2=a-b$; $z_1=\cos4\alpha-\sin4\alpha\cdot\mathrm{ctg}2\alpha$; $z_2=\cos2\alpha-2\cos^2\alpha$
14	a=1.7 b=2.8 α=0.22	$y_1=\frac{a^{\frac{1}{2}}+ab^{-1}}{a^{-\frac{1}{3}}-a^{-\frac{1}{6}}b^{-\frac{1}{3}}+b^{-\frac{2}{3}}}-\frac{a}{\sqrt[3]{b}}$; $y_2=a^{\frac{5}{6}}$; $z_1=\cos4\alpha\cdot\mathrm{tg}2\alpha-\sin4\alpha$; $z_2=\frac{2\mathrm{tg}\alpha}{\mathrm{tg}^2\alpha-1}$
15	m=1.8 α=0.43 β=0.58	$y_1=\frac{4m\left(m+\sqrt{m^2-1}\right)^2}{\left(m+\sqrt{m^2-1}\right)^4-1}$; $y_2=\frac{1}{\sqrt{m^2-1}}$; $z_1=\frac{\mathrm{tg}2\alpha+\mathrm{ctg}3\beta}{\mathrm{ctg}2\alpha+\mathrm{tg}3\beta}$; $z_2=\frac{\mathrm{tg}2\alpha}{\mathrm{tg}3\beta}$
16	x=5.3 α=0.3 β=0.1	$y_1=\frac{x+\sqrt{x^2-4x}}{x-\sqrt{x^2-4x}}-\frac{x-\sqrt{x^2-4x}}{x+\sqrt{x^2-4x}}$; $y_2=\sqrt{x^2-4x}$; $z_1=\sin(\alpha+\beta)\cdot\sin(\alpha-\beta)\cdot\sec^2\alpha\cdot\sec^2\beta$; $z_2=\mathrm{tg}^2\alpha-\mathrm{tg}^2\beta$
17	b=4.8 α=0.23	$y_1=\frac{b^2-3b-(b-1)\sqrt{b^2-4}+2}{b^2+3b-(b+1)\sqrt{b^2-4}+2}\cdot\sqrt{\frac{b+2}{b-2}}$; $y_2=\frac{1-b}{1+b}$; $z_1=\frac{\cos4\alpha+1}{\mathrm{ctg}\alpha-\mathrm{tg}\alpha}$; $z_2=\frac{1}{2}\sin4\alpha$

№	Данные	Формулы
18	x_1=2.8 x_2=4.8 α=0.97	$y_1=\sqrt{\frac{4}{x}+\frac{1}{4x^{-1}}-2}+\sqrt{\frac{1}{4x^{-1}}+\frac{1}{4x}+\frac{1}{2}}$; при $x \le 4$ $y_1 \sim y_2$; при $x>4$ $y_1 \sim y_3$ $y_2=\frac{5}{2\sqrt{x}}$; $y_3=\frac{2x-3}{2\sqrt{x}}$ $z_1=\sin^2\left(\frac{7}{8}\pi-2\alpha\right)-\sin^2\left(\frac{9}{8}\pi-2\alpha\right)$; $z_2=\frac{\sin 4\alpha}{\sqrt{2}}$
19	x=1.4 y=2.8 α=0.5 β=0.34	$y_1=\frac{\sqrt{x^3}+\sqrt{xy^2}-\sqrt{x^2y}-\sqrt{y^3}}{\sqrt[4]{y^5}+\sqrt[4]{x^4y}-\sqrt[4]{xy^4}-\sqrt[4]{x^5}}$; $y_2=-\left(\sqrt[4]{x}+\sqrt[4]{y}\right)$; $z_1=\operatorname{ctg}^2\alpha-\operatorname{ctg}^2\beta$; $z_2=\frac{\cos^2\alpha-\cos^2\beta}{\sin^2\alpha\cdot\sin^2\beta}$
20	a=5.1 α=0.3	$y_1=\left(\frac{1+\sqrt{a}}{\sqrt{1+a}}-\frac{\sqrt{1+a}}{1+\sqrt{a}}\right)^2-\left(\frac{1-\sqrt{a}}{\sqrt{1+a}}-\frac{\sqrt{1+a}}{1-\sqrt{a}}\right)^2$; $y_2=\frac{16a\sqrt{a}}{\left(1-a^2\right)\cdot(a-1)}$; $z_1=(1+\sec 2\alpha+\operatorname{tg}2\alpha)\cdot(1-\sec 2\alpha+\operatorname{tg}2\alpha)$; $z_2=2\operatorname{tg}2\alpha$
21	x=0.3 α=0.77	$y_1=\left(\sqrt{1-x^2}+1\right):\left(\frac{1}{\sqrt{1+x}}+\sqrt{1-x}\right)$; $y_2=\sqrt{1+x}$; $z_1=\frac{\cos(3\pi-2\alpha)}{2\sin^2\left(\frac{5}{4}\pi+\alpha\right)}$; $z_2=\operatorname{tg}\left(\alpha-\frac{5}{4}\pi\right)$
22	a=12.3 α=0.24	$y_1=\left(\frac{\sqrt{a}}{2}-\frac{1}{2\sqrt{a}}\right)^2\left(\frac{\sqrt{a}-1}{\sqrt{a}+1}-\frac{\sqrt{a}+1}{\sqrt{a}-1}\right)$; $y_2=\frac{1\text{-}a}{\sqrt{a}}$; $z_1=\frac{\sin 2\alpha-\sin 3\alpha+\sin 4\alpha}{\cos 2\alpha-\cos 3\alpha+\cos 4\alpha}$; $z_2=\operatorname{tg}3\alpha$
23	a=2.3 b=1.89 α=0.23	$y_1=\frac{(2a-b)^2+2b^2\text{-}3ab}{2a^{-1}+b^2}:\frac{4a^2-3ab}{2+ab^2}$; $y_2=a\text{-}b$; $z_1=\sin^6\left(\frac{\alpha}{2}\right)-\cos^6\left(\frac{\alpha}{2}\right)$; $z_2=\frac{1}{4}\left(\sin^2\alpha-4\right)\cos\alpha$

№	Данные	Формулы
24	b=3.8 α=0.28	$y_1=\dfrac{\sqrt{2b+2\sqrt{b^2-4}}}{\sqrt{b^2-4}+b+2}$; $y_2=\dfrac{1}{\sqrt{b+2}}$; $z_1=\sin^2\left(\dfrac{15}{8}\pi-2\alpha\right)-\cos^2\left(\dfrac{17}{8}\pi-2\alpha\right)$; $z_2=\dfrac{-\cos 4\alpha}{\sqrt{2}}$
25	p=0.7 α=0.54	$y_1=\left(\left(1-p^2\right)^{-\frac{1}{2}}-\left(1+p^2\right)^{-\frac{1}{2}}\right)^2+2\left(1-p^4\right)^{-\frac{1}{2}}$; $y_2=\dfrac{2}{\left(1-p^4\right)}$; $z_1=\operatorname{tg}\alpha+\operatorname{ctg}\alpha+\operatorname{tg}3\alpha+\operatorname{ctg}3\alpha$; $z_2=\dfrac{8\cos^2 2\alpha}{\sin 6\alpha}$
26	m_1=0.47 m_2=2.47 α=0.1	$y_1=\dfrac{\sqrt{(2m+3)^2-24m}}{2\sqrt{m}-\dfrac{3}{\sqrt{m}}}$; при $m \le 1.5$ $y_1 \sim y_2$; при $m > 1.5$ $y_1 \sim y_3$ $y_2=-\sqrt{m}$; $y_3=\sqrt{m}$ $z_1=\dfrac{\cos\alpha+\sin\alpha}{\cos\alpha-\sin\alpha}$; $z_2=\operatorname{tg}2\alpha+\sec 2\alpha$
27	x=4.3 α=1.23	$y_1=\dfrac{x^4-x^3-x+1}{x^3-5x^2+7x-3}\lvert x-3\rvert$; $y_2=x^2+x+1$; $z_1=\dfrac{\sin 4\alpha}{1+\cos 4\alpha}\cdot\dfrac{\cos 2\alpha}{1+\cos 2\alpha}$; $z_2=\operatorname{ctg}\left(\dfrac{3}{2}\pi-\alpha\right)$
28	m=2.3 α=0.23 β=1.2	$y_1=\dfrac{m^2-m-6-(m+3)\sqrt{m^2-4}}{m^2+m-6-(m-3)\sqrt{m^2-4}}$; $y_2=\dfrac{-\sqrt{m+2}}{\sqrt{m-2}}$; $z_1=(\cos\alpha-\cos\beta)^2+(\sin\alpha-\sin\beta)^2$; $z_2=4\sin^2\dfrac{\alpha-\beta}{2}$
29	a=6.3 α=0.1 β=0.7	$y_1=\dfrac{1+2a^{\frac{1}{4}}-a^{\frac{1}{2}}}{1-a+4a^{\frac{3}{4}}-4a^{\frac{1}{2}}}+\dfrac{a^{\frac{1}{4}}-2}{\left(a^{\frac{1}{4}}-1\right)^2}$; $y_2=\dfrac{1}{\sqrt[4]{a}-1}$; $z_1=\dfrac{\sin\alpha+\cos(2\beta-\alpha)}{\cos\alpha-\sin(2\beta-\alpha)}$; $z_2=\dfrac{1+\sin 2\beta}{\cos 2\beta}$

№	Данные	Формулы
30	m_1=0.65 m_2=1.65 α=1.43	$y_1 = \dfrac{m^5 + m^4 \cdot \sqrt[3]{2} + \sqrt[3]{4m^9}}{\left\|m^3 - 1\right\| - 1}$; при $m>1$ $y_1 \sim y_2$; при $m \le 1$ $y_1 \sim y_3$ $y_2 = \dfrac{m^3}{m - \sqrt{2}}$ $y_3 = -\left(m^2 + m\sqrt[3]{2} + \sqrt[3]{4}\right)$; $z_1 = \dfrac{1 - 2\sin^2\alpha}{1 + \sin 2\alpha}$; $z_2 = \dfrac{1 - \mathrm{tg}\alpha}{1 + \mathrm{tg}\alpha}$

2.2. Ветвления If и циклы Do

Задание

Заштрихованная часть рисунка в дальнейшем называется *областью*. Задача состоит в проверке попадания точек *(x,y)* в заданную область.

Задание содержит рисунок, состоящий из геометрических фигур.

1. Написать программу, которая:
   a) для точек каждой контурной линии рисунка составляет таблицу соответствия координат в файлах с расширением *.txt*;
   b) генерирует точки в прямоугольнике, перекрывающем рисунок на 10-20% с каждой стороны, и записывает координаты этих точек в один из двух файлов в зависимости от их нахождения в *области*.
2. Для графической интерпретации результатов использовать программу *Agrapher.*

Последовательность выполнения работы

1. Графики контурных линий должны повторять рисунок индивидуального задания:
   a) для каждой i-ой линии контура задать область определения и уравнение $y_i=f_i(x);$ проверить уравнения в *Agrapher*;
   b) написать *программу* табуляции $f_i(x);$ строки *<аргумент><значение функции>* записать в файлы с расширением *.txt;*
   c) импортировать эти файлы в *Agrapher* для тестирования; выполнить настройки: *линии* – красным цветом, *точки* удалить;
   d) сохранить график в файле *Contur.agr* для дальнейшего использования.
2. Описать математически *систему неравенств,* обеспечивающую попадание произвольной точки в область. Проверить правильность системы неравенств в *AGrapher*.
3. Рисунок индивидуального задания накрыть прямоугольной сеткой,

в узлах которой находятся точки для проверки системы неравенств:

a) прямоугольник должен на 10-20% перекрывать заданный график с каждой из четырех сторон;
b) рекомендуемое число узлов сетки – 20-40 по каждой оси.

Определить *параметры сетки*: начальное и конечное значения *x* и *y*, шаг изменения *x* и *y*.

3. Дополнить *программу*:
   a) генерировать узлы сетки, используя ее параметры;
   b) проверить каждую точку на соответствие с системой неравенств, используя *единственный* оператор *IF*;
   c) записать координаты точки (*x*, *y*) в один из файлов для *Agrapher:*
      – если точка в пределах области – в файл *in.txt,*
      – иначе – в файл *out.txt.*
4. Дополнить график в *Agrapher*:
   a) прочитать график контурных линий из файла *Contur.agr*;
   b) импортировать таблицы из файлов *in.txt* и *out.txt*;
   c) установить для точек из *in.txt* и *out.txt* разные цвета; удалить линии, соединяющие точки;
   d) предъявить преподавателю график на экране.

Содержание отчета

1. Название работы, номер варианта.
2. Фамилия, имя, отчество и номер группы студента.
3. Рисунок в том виде, как он приведен в варианте задания.
4. Таблица участков графиков контурных линий (смотри пример), в которой для каждого участка указаны границы, уравнения, шаг табуляции, количество выполнений цикла.
5. Система неравенств для всех точек области.
6. Блок – схема и распечатка текста программы.

Справочная информация

Порядок действий в логических выражениях определяется приоритетом используемых операций. В приведенной ниже таблице 5 L1, L2 – логические выражения; a, b – числа или строки. В операциях отношения можно сравнивать не только числа, но и строки – они сравниваются, как в словаре (в соответствии с алфавитом).

Таблица 4.

Операции отношения и логические операции

Операция		**Фортран-77**	**Фортран-90**	**Приоритет**
Вычислить значения a,b		Арифметические или строковые операции.		1
Операции отношения	Больше чем	a.**GT**.b	a>b	2
	Больше или равно	a.**GE**.b	a>=b	2
	Меньше чем	a.**LT**.b	a<b	2
	Меньше или равно	a.**LE**.b	a<=b	2
	Равенство	a.**EQ**.b	a= =b	2
	Неравенство	a.**NE**.b	a/=b	2
Логические операции	Инверсия	.**NOT**.L1		3
	Логическое умножение	L1.**AND**.L2		4
	Логическое сложение	L1.**OR**.L2		5
	Эквивалентность	L1.**EQV**.L2		6
	Неэквивалентность	L1.**NEQV**.L2		6

Комментарии к заданию

1. Приложение *Advanced Grapher* распространяется бесплатно, адрес сайта http://www.alentum.com/agrapher/.
2. Напишите уравнение и выберите шаг табуляции каждой линии контура, заполните *таблицу*. На криволинейном участке с целью улучшения изображения в *AGrapher* шаг выбирается более мелкий. Для прямой выводятся две точки (без цикла).
3. Если аргумент тригонометрической функции задан в градусах, следует в программе использовать функции `Sind(x)`, `Cosd(x)`, `Tand(x)`.
4. Для табуляции линий использовать оператор цикла по переменной

```
DO x=xn,xk,step
   . . .
enddo
```

Здесь *x* – переменная цикла, *xn*, *xk* – начальное и конечное значения *x*, *step* - шаг изменения *x*.

5. Количество повторений цикла рассчитывается по формуле:

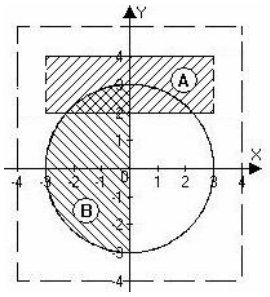
$$M = \mathrm{Max}\left(0, \mathrm{Int}\left(\frac{xk - xn + step}{step}\right)\right) \qquad \text{или} \qquad M = \mathrm{Max}\left(0, \mathrm{Int}\left(\frac{xk - xn}{step} + 1\right)\right)$$

6. В системе неравенств показать *объединение* квадратной скобкой, *пересечение* – фигурной скобкой.
7. Пунктирная контурная линия на рисунке означает, что точки, лежащие на ней, не принадлежат области.
8. При составлении системы неравенств рекомендуется рассматривать область по отдельным фрагментам, что поможет в решении задачи. Система неравенств в этом случае представляется *объединением* групп неравенств, соответствующих этим фрагментам.
9. Рекомендуется воспользоваться симметрией заданной области, иногда это помогает уменьшить количество неравенств.
10. Генерируя точки – узлы сетки, воспользуйтесь вложенными циклами для задания их координат.

Пример

В Таблице 5 на рисунке показана область и ограничивающий ее прямоугольник. Область разбита на два фрагмента – A и B; точка находится в пределах заданной области, если она принадлежит хотя бы одному из них.

Таблица 5. Заданная область и ограничивающий ее прямоугольник



Система неравенств:

$$\left[\begin{cases} y \ge 2 \\ y \le 4 \\ |x| < 3 \end{cases} \quad (\textit{фрагмент} \quad A)\right.$$

$$\left[\begin{cases} x^2 + y^2 \le 9 \\ x \le 0 \end{cases} \quad (\textit{фрагмент} \quad B)\right.$$

Границы прямоугольника, содержащего точки: $x \in [-4, +4]$, $y \in [-4, +5]$.
Шаг изменения координат *x* и *y* равен 0,4.

Таблица 6. Графики контурных линий к рисунку из Табл. 5

№	Границы участка	Уравнение	Шаг цикла	Количество точек	Примечание
1	$x = [-3, 3]$	$y = 4$ $y = 2$	цикла нет	2 2	Горизонтальные стороны прямоугольника – сплошные.
2	$y = [2, 4]$	$x = -3$ $x = 3$	цикла нет	2 2	Вертикальные стороны прямоугольника – пунктир.
3	$\alpha \in [0, 360^{\circ}]$	$x = 3\cos\alpha$ $y = 3\sin\alpha$	Step=3	Точек цикла 121	Параметрические уравнения окружности радиуса *R*=3 с центром в начале координат.

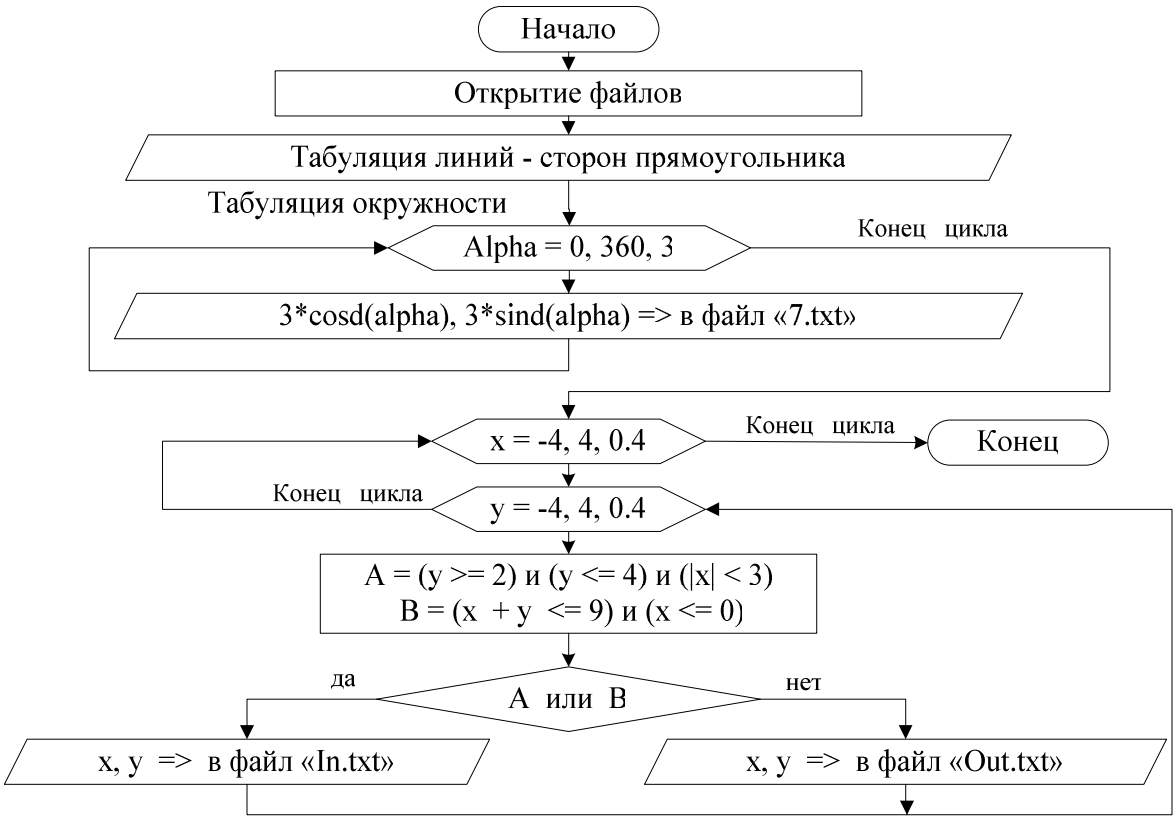


Рис.1. Блок-схема программы

Программа.

```
Program Region     ! Попадание точки в область плоскости
! студент (фамилия, имя)  группа №  работа №  вариант №
 Implicit none
  Real x, y
  Integer alpha  ! alpha–целый, Real(alpha)- вещественный
  Integer,parameter::inside=1,outside=2 ! номера устройств
  Logical A, B
!-------выполняемые операторы
Open(3,FILE='3.txt') ! координаты точек сторон прямоугольника
Open(4,FILE='4.txt')
Open(5,FILE='5.txt')
Open(6,FILE='6.txt')
Open(7,FILE='7.txt') ! координаты точек окружности
Open(inside,FILE='in.txt') ! координаты точек внутри области
Open(outside,FILE='out.txt')! координаты точек вне области
!------табуляция сторон прямоугольника (файлы на устройствах 3 - 6)
Write(3,*)-3,4; Write(3,*)3,4 ! верхняя сторона прямоугольника
Write(4,*)-3,2; Write(4,*)3,2  ! нижняя сторона прямоугольника
Write(5,*)-3,4; Write(5,*)-3,2 ! левая сторона прямоугольника
Write(6,*)3,4; Write(6,*)3,2   ! правая сторона прямоугольника
```

```
!------окружность (файл на устройстве 7)
Do alpha=0,360,3 ! аргумент функций Sind, Cosd - в градусах, целый
  x=3*Cosd(real(alpha)); y=3*Sind(real(alpha)) ! real()–вещ.
  Write(7,*)x,y;
Enddo

! цикл по Y внутри цикла по X - перебор точек прямоугольника
Do x = -4, 4, 0.4
  Do y = -4, 5, 0.4
    A = y>=2 .and. y<=4 .and. abs(x)<3 ! фрагмент A
    B = x*x + y*y <= 9 .and. x<=0      ! фрагмент B
    If (A .or. B) then
      Write(inside, *) x, y
    Else
      Write(outside, *) x, y
    End If
  End Do
End Do
End Program Region
```

Контрольные вопросы

1. Что называется логическим выражением, бывает ли оно смешанным? Могут ли в него входить числа?
2. Как в логическом выражении распределяются приоритеты между операциями арифметическими, логическими, отношения?
3. Пять логических операций (три основные и две дополнительные) с таблицами истинности и примерами.
4. Приоритеты выполнения логических операций в логическом выражении.
5. Шесть операций отношения с примерами.
6. Запись на Фортране логических констант. Как логическую константу “истина” задать с именем *ON*?
7.
```
Integer :: B=27, C=3
L1=B**1/3==C
```
   Как объявить тип переменной *L1* (оператор)? Вычислите ее значение.
8. Что такое цикл в программе, управляющее выражение, параметр цикла, тело цикла?
9. Приведите стандартную блок-схему и пример
   - по переменной;
   - бесконечного цикла;
   - итеративного цикла.

10. Объясните формулу расчета числа повторений цикла по переменной.
11. Приведите подробную блок-схему цикла по переменной.
12. Приведите блок-схемы и примеры условных операторов
    - с двумя блоками;
    - с одним блоком;
    - без блоков.

Примеры задач контрольного задания

1. Рассчитайте число повторений цикла `Do X=3.15,2.12,1.3`
2. Напишите оператор цикла по переменной для вывода на экран 11 пар значений *x* и *tg x*, если *x* изменяется от -100^o до -200^o. Приведите блок-схему этого оператора.
3. Подсчитайте количество выведенных строк

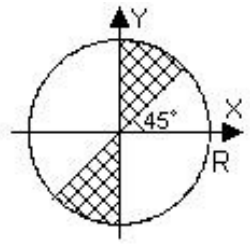
```
Do a = 22, 9, -8
  Do b = 3, 19, 11
    write(*, *) a, b
  enddo
enddo
```

   Какие значения *a* и *b* будут выведены?

4. Дан фрагмент программы:

```
A = x>0;  B = y<0;  C = y**2<-x+1
L1 = B.or.A.and.C
```

   – Объявите тип всех переменных (операторы Фортрана).
   – Приведите систему неравенств и графическую интерпретацию всех логических переменных.

5. 

   При каких значениях координат точка [*x, y*] находится в заштрихованной области плоскости? Опишите эти условия:
   a) в виде системы неравенств;
   b) в виде логического выражения (без промежуточных переменных)

6. Напишите условный оператор для вычисления переменной Y:

$$Y = \begin{cases} \frac{2}{9}x^3 & \text{при} \quad x < 2, \\ sec^2\frac{1}{x} & \text{при} \quad 2 \le x < 4, \\ \sqrt[3]{e^x + 2} & \text{при} \quad x \ge 4 \end{cases}$$

7. Напишите по блок-схеме фрагмент программы на Фортране.

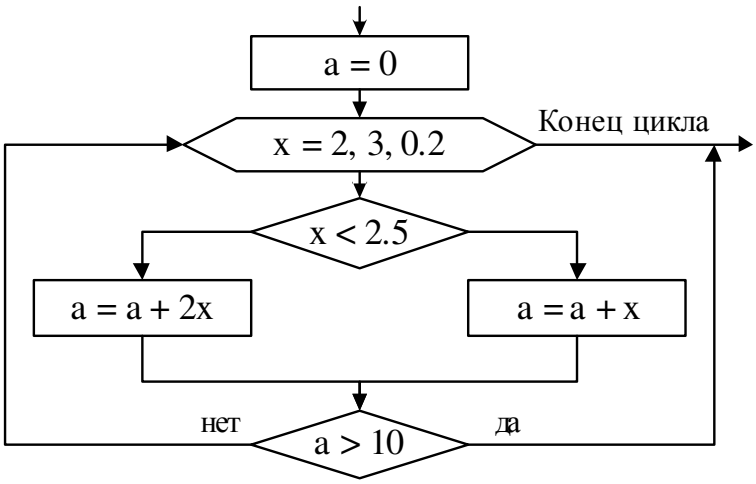
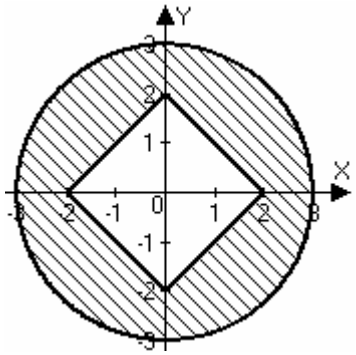
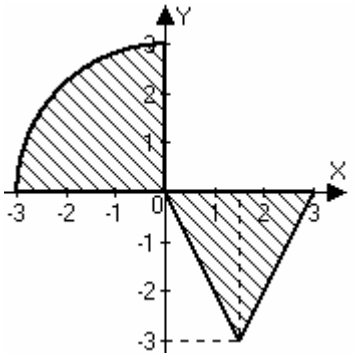
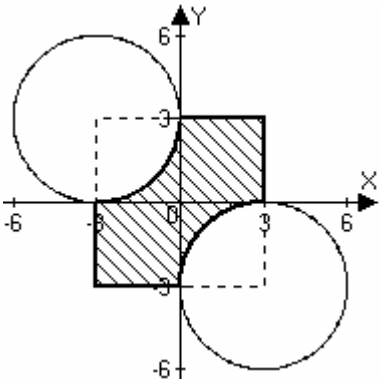
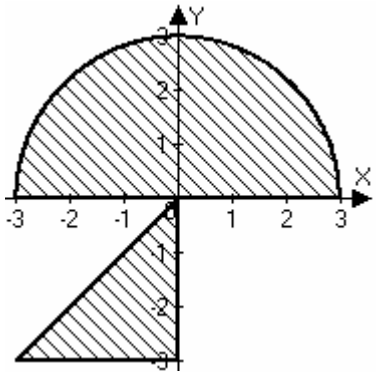
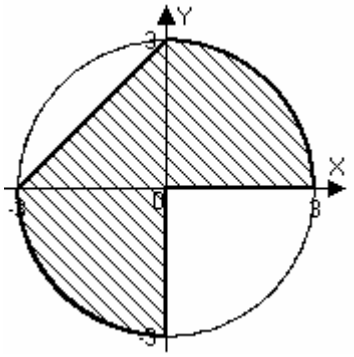
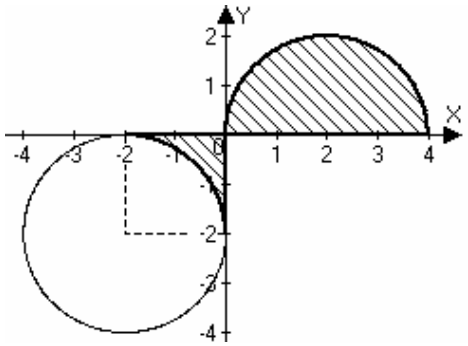
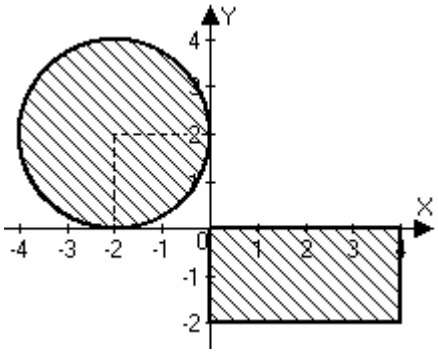
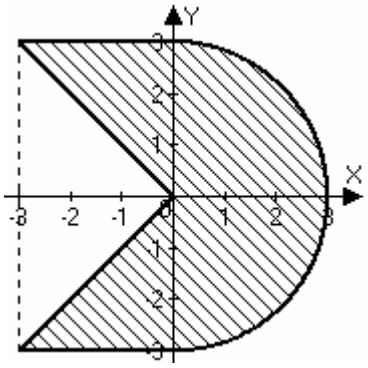
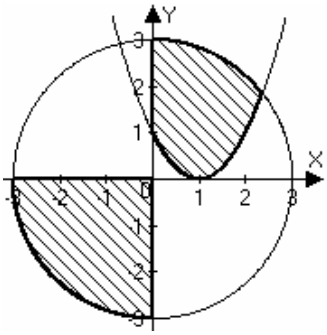
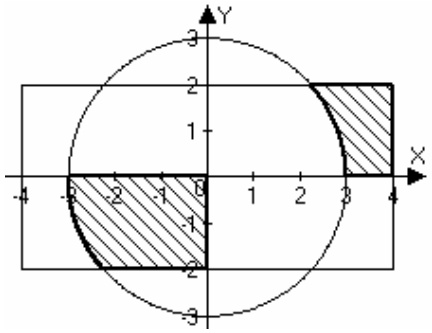
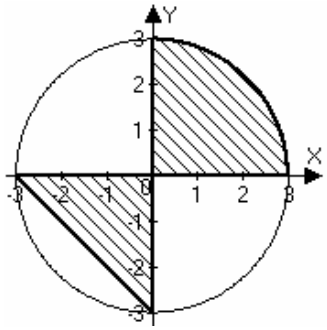
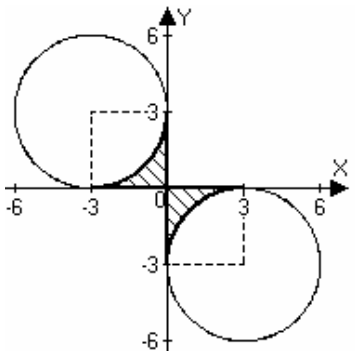
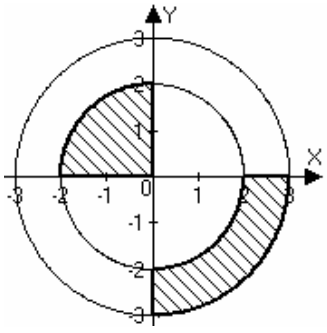
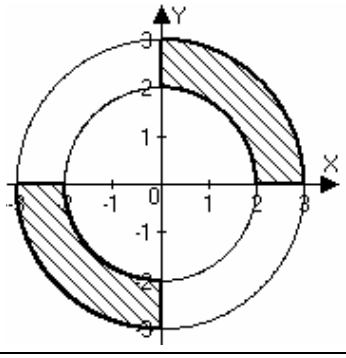


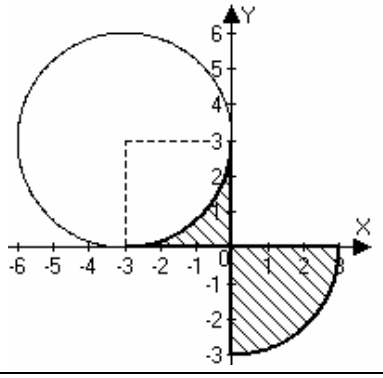
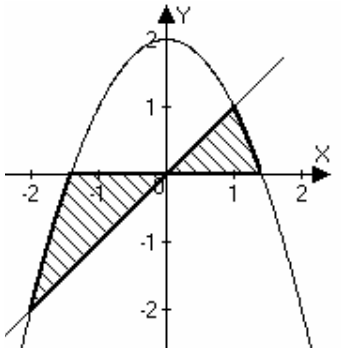
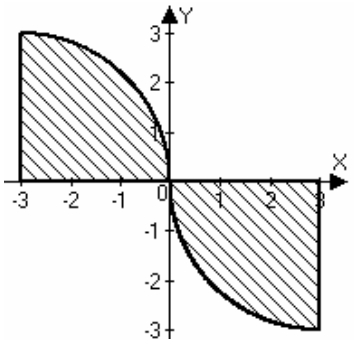
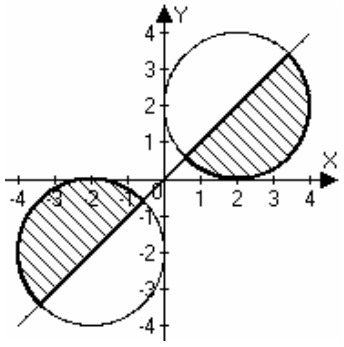
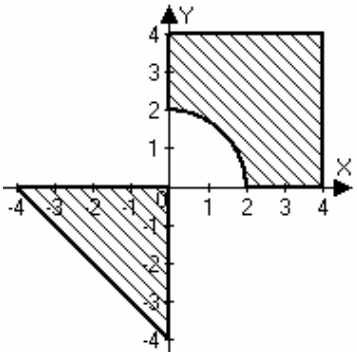
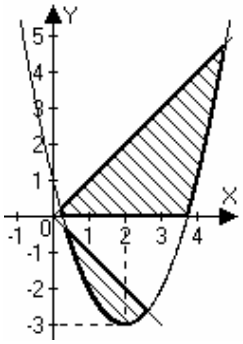
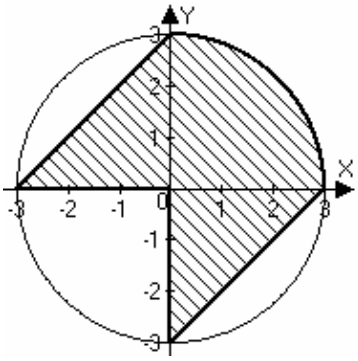
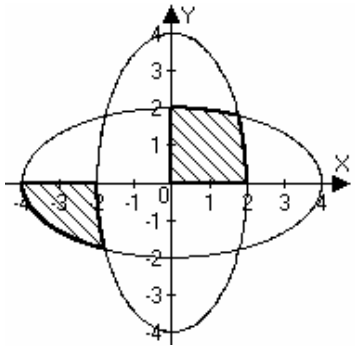
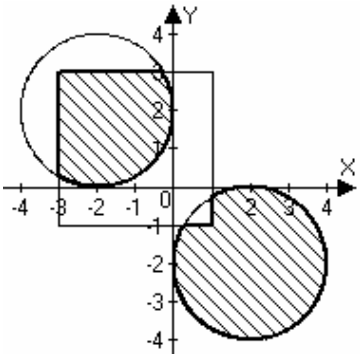
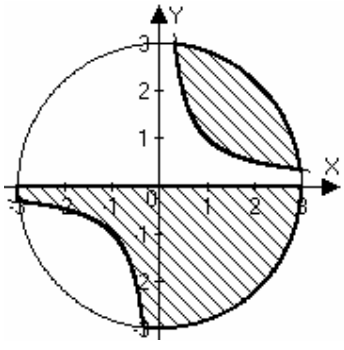
Рис.2. Блок-схема фрагмента программы

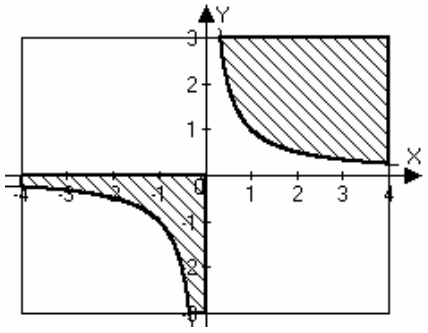
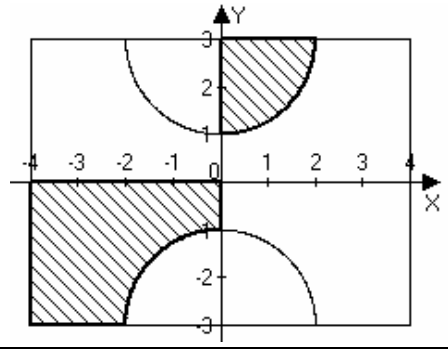
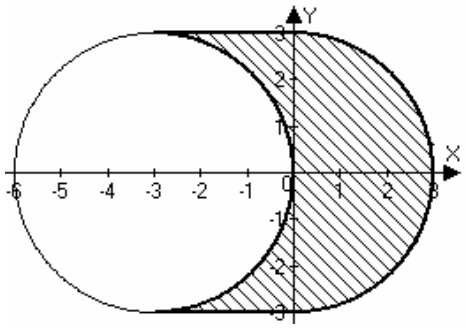
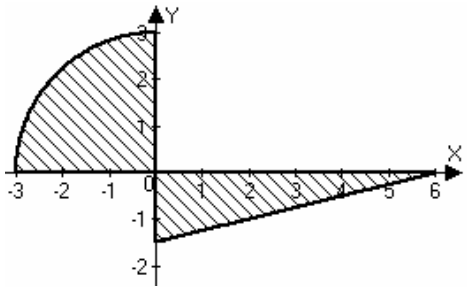
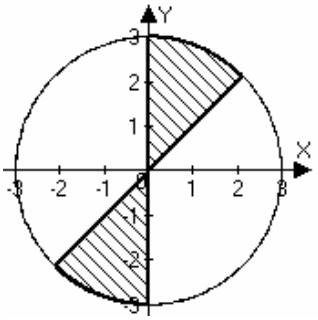
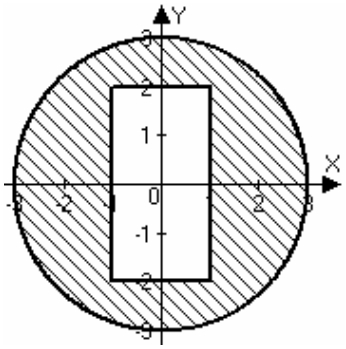
Таблица 7.

Варианты индивидуальных заданий «Ветвления и циклы»

№	Графики	№	Графики
1		2	
3		4	

№	Графики	№	Графики
5		6	
7		8	
9		10	
11		12	
13		14	

№	Графики	№	Графики
15		16	
17		18	
19		20	
21		22	
23		24	

№	Графики	№	Графики
25		26	
27		28	
29		30	

2.3. Сумма степенного ряда - приближенное вычисление функции

Задание

1. Составить программу *Pro1*, которая для рекомендованного значения аргумента $x = x_0$ и заданного значения точности ε:
   a) вычисляет значение функции $f(x_0)$ со встроенной функцией;
   b) вычисляет приближенное значение $S_n(x_0)$, суммируя столько членов ряда Тейлора, сколько требуется для заданной точности ε;
   c) формирует три текстовых файла для пакета *AGrapher*:
      – зависимость членов ряда a_n от n;
      – зависимость *частичных* сумм ряда S_n от n;
      – значение встроенной функции $f(x_0)$, одно и то же при всех n;
      для построения графиков принять $\varepsilon = 10^{-3}$;
   d) выводит в файл *Out.txt* результаты с пояснениями, включая:
      – значение ε;
      – значение аргумента $x = x_0$;
      – $f(x_0)$, вычисленное по стандартной программе;

- S_n - сумма ряда, как приближение функции;
- n - количество членов ряда в частичной сумме ряда;
- модуль разности $f(x_0)$ и суммы ряда S_n.

2. С помощью пакета *AGrapher* построить графики зависимостей по текстовым файлам *An.txt, Sn.txt, f.txt,* созданным в программе *Pro1*.
3. Составить программу *Pro2,* модифицировав программу *Pro1*:
   a) исключить формирование текстовых файлов для графиков;
   b) результаты вычислений оформить в виде таблицы (каждая строка таблицы соответствует одному значению x и содержит форматированные результаты);
   c) значение аргумента x функции $f(x)$ изменять от x_n до x_k с шагом Δx из расчета 10 – 15 строк в таблице.
4. Выполнить программу *Pro2* дважды с разной степенью точности ε:
   a) различимой визуально на графике;
   b) достижимой для *Real* - одинарной точности.

Содержание отчета

1. Название работы, номер варианта индивидуального задания.
2. Фамилия, имя, отчество и номер группы студента.
3. Вариант индивидуального задания (функция, её разложение в ряд Тейлора и область определения аргумента функции).
4. Рекуррентная формула и расчет коэффициента рекурсии с последующей его проверкой для двух членов ряда.
5. Блок-схемы и распечатки текстов программ *Pro1* и *Pro2*.
6. Распечатка результатов работы программы *Pro1*.
7. Рисунок или распечатка графиков, построенных в *AGrapher* по трем файлам, созданным в *Pro1*.
8. Распечатки двух таблиц с результатами программы *Pro2*.

При подготовке к защите ответить на контрольные вопросы.

Справочная информация

1. Запись бесконечного ряда Тейлора:

   $\sum_{n=0}^{\infty} a_n = a_0 + a_1 + a_2 + ... + a_n + ...$, здесь a_n – *общий член* ряда;

   $S_0, S_1, \ldots S_n, \ldots$ называют *частичными суммами* бесконечного ряда,

   $S_0 = a_0,\ S_1 = a_0 + a_1,\ \ldots,\ S_n = \sum_{k=0}^{n} a_k = a_0 + a_1 + a_2 + ... + a_n,\ \ldots$

2. *Сходящимся* называется ряд, у которого последовательность частичных сумм имеет конечный предел S, т.е. $\lim_{n\to\infty} S_n = S$, иначе ряд называется *расходящимся*. S называется *суммой сходящегося ряда*. Нахождение частичной суммы S_n – цель лабораторной работы.

3. Величина n! называется “*n*-факториал” и вычисляется по формуле

$$n! = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n = (n-1)! \cdot n \qquad \text{или} \qquad n! = \prod_{k=1}^{n} k \quad \text{при} \quad 0! = 1.$$

Комментарии к заданию

1. Обратите внимание на то, что в Вашем индивидуальном задании:
   a) список членов ряда может начинаться не с «a_0», а с «a_1» (начальное значение для n=1);
   b) a_0 (или a_1) может быть функцией от x или константой;
   c) сумме может предшествовать слагаемое или сомножитель.
2. При суммировании ряда необходимо решить следующие задачи:
   a) свести вычисления к простейшим арифметическим операциям;
   b) уменьшить число этих операций и время расчета;
   c) уменьшить погрешность вычислений.

   Эти задачи решает *рекуррентная* формула, позволяющая вычислить значение очередного члена ряда, используя уже найденное значение предыдущего. Рекуррентная формула имеет вид:

   $$a_{n+1} = a_n \cdot T_n, \qquad \text{где } T_n - \text{коэффициент рекурсии.}$$
3. Возможно суммирование только конечного количества членов ряда. Для сходящегося ряда $\lim\limits_{n\to\infty} a_n = 0$. Следовательно, начиная с некоторого n, отношение $|a_n| > \varepsilon$ перестанет выполняться для любого положительного ε. Этим значением n и следует ограничиться при суммировании бесконечного ряда.
4. Во избежание зацикливания программы вследствие ошибок, значение n следует ограничить, для чего в программе предусмотрена переменная *Nmax* (равная, например, 100). При $n > Nmax$ суммирование прекращается и выдается сообщение с дополнительной информацией (в том числе значение a_n).

   Аварийное сообщение может появиться, если:
   a) значение $|a_n|$ приближается к ε, но, возможно, ряд просто «не успел» сойтись и достаточно просто увеличить значение *Nmax* ;
   b) значение $|a_n|$ на порядки превышает ε, то есть ряд *не сходится*. В этом случае следует проверить:
      – находится ли x в области определения аргумента;
      – достижима ли задаваемая погрешность вычислений ε при использовании *Real* – одинарной точности вещественных переменных;
      – на ошибки в рекуррентной формуле, алгоритме, программе.

Пример

Рассмотрим вычисление функции *sin x*. Разложение этой функции в ряд Тейлора имеет следующий вид

$$f(x) = \sin x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \text{K}, \text{ где } |x| < \infty$$

Расчет коэффициента рекурсии

Формула для вычисления общего члена ряда

$$a_n = \frac{(-1)^n x^{2n+1}}{(2n+1)!}, \qquad \text{где } n = 0, 1, 2, ...$$

При $n = 0$, $a_0 = x$. Каждый следующий член ряда a_{n+1} можно вычислить с помощью рекуррентной формулы

$$a_{n+1} = a_n \cdot T_n, \qquad \text{где } n = 0, 1, 2, \ldots .$$

Коэффициент рекурсии T_n определяется из соотношения

$$T_n = \frac{a_{n+1}}{a_n}, \qquad \text{где } n = 0, 1, 2, \ldots .$$

Выполним подстановки

$$a_{n+1} = \frac{(-1)^{n+1} \cdot x^{2(n+1)+1}}{(2(n+1)+1)!} = (-1)^n \cdot (-1) \frac{x^{2n} \cdot x^3}{(2n+3)!}$$ – *подставлено n+1 вместо n*

$$a_n = \frac{(-1)^n \cdot x^{2n+1}}{(2n+1)!} = (-1)^n \frac{x^{2n} \cdot x}{(2n+1)!}$$

$$T_n = \left[(-1)^n \cdot (-1) \frac{x^{2n} \cdot x^3}{(2n+3)!} \right] : \left[(-1)^n \frac{x^{2n} \cdot x}{(2n+1)!} \right] =$$

– *подставлено* a_{n+1}, a_n

$$= \frac{(-1)^n \cdot (-1)}{(-1)^n} \cdot \frac{x^{2n} \cdot x^3}{x^{2n} \cdot x} \cdot \frac{(2n+1)!}{(2n+3)!} = (-1) \cdot x^2 \cdot \frac{(2n+1)!}{(2n+3)!}$$

Чтобы сократить факториалы, приведем числитель и знаменатель отдельно

$(2n + 1)! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (2n + 1)$

$(2n + 3)! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (2n + 1) \cdot (2n + 2) \cdot (2n + 3)$

Коэффициент рекурсии $$T_n = \frac{-x^2}{(2n+2)(2n+3)}$$

Рекуррентная формула для расчетов $$a_{n+1} = a_n \cdot \frac{-x^2}{(2n+2)(2n+3)}$$

Проверка рекуррентной формулы

Подставив $n = 0$ в формулу общего члена ряда a_n, получаем $a_0 = x$.

Далее определим по рекуррентной формуле a_1 и a_2, сверяя результаты с соответствующими членами ряда:

при $n = 0$ $$a_1 = a_0 \cdot T_0 = x \cdot (-x^2) \cdot \frac{1}{2 \cdot 3} = -\frac{x^3}{2 \cdot 3} = -\frac{x^3}{3!}$$

при $n = 1$ $$a_2 = a_1 \cdot T_1 = -\frac{x^3}{2 \cdot 3} \cdot (-x^2) \cdot \frac{1}{4 \cdot 5} = \frac{x^5}{2 \cdot 3 \cdot 4 \cdot 5} = \frac{x^5}{5!}$$

Совпадение полученных значений с членами ряда показывает, что коэффициент рекурсии выведен правильно.

Блок-схема алгоритма суммирования ряда

В алгоритме реализуются те же действия, что и при проверке рекуррентной формулы.

Одна переменная *an* используется в программе при вычислении всех членов ряда $a_0, a_1, \ldots a_n$, принимая перечисленные значения по очереди.

Суммирование членов ряда выполняется в цикле

do while «пока |*An*| > *eps*», аварийный выход из цикла при *N* > *Nmax.*

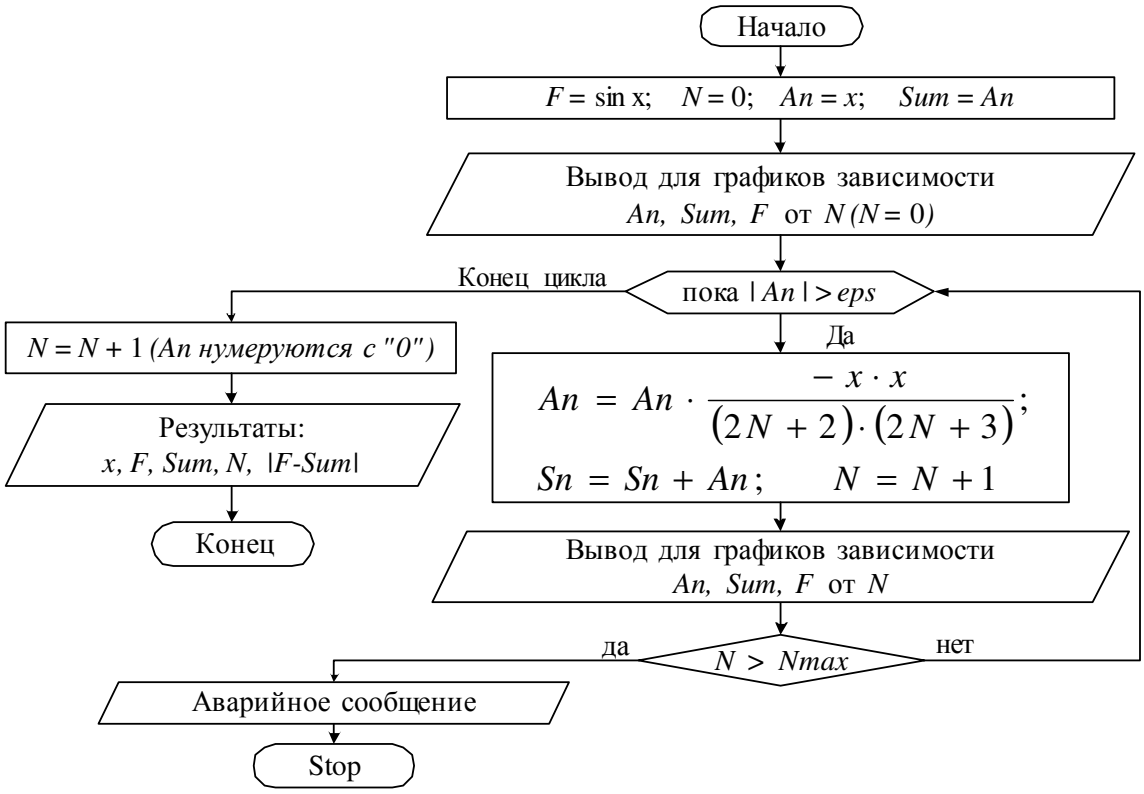


Рис.3. Блок-схема алгоритма суммирования ряда

Программа суммирования ряда для одного значения аргумента

```
Program Pro1     ! суммирование ряда – первая версия – для одного x
! студент (фамилия, имя)  группа №  работа №  вариант №
Implicit None
Real:: x = 1.1       ! рекомендованный x0 в радианах
Real:: eps = 1E-04   ! методическая погрешность
Integer:: N          ! номер члена ряда
Integer:: Nmax=100   ! максимально допустимое значение члена ряда
Real An, Sn, F       ! член ряда A0,A1,A2..An,  сумма,  функция
Real Delta           ! модуль разности между Sn и F
! список переменных, выводимых при достижении заданной точности:
    Namelist /result/ x, F, Sn, n, Delta
! список переменных, выводимых при аварийном предупреждении:
    Namelist /avaria/ x, An, Sn, F, N, Nmax
!-------------начало выполнения программы
```

```
Open(1,file='Out.txt')! файл с результатами
Open(2,file='An.txt')! файл для AGrapher - зависимость An от N
Open(3,file='Sn.txt')! файл для AGrapher- зависимость Sn от N
Open(4,file='F.txt')! файл для AGrapher - F (одинаково при всех N)
Write(1, *) 'Точность вычисления ', eps
```

Суммирование ряда, используя do while, выполнить по блок-схеме, приведенной выше; внести изменения с учетом индивидуального задания. Для вывода аварийного сообщения используйте оператор **Write**(1,avaria)

```
  ! РАСЧЕТ ЗАКОНЧЕН
   Delta=ABS(F–Sn)! модуль разности между суммой Sn и функцией F
   Write(1,result) ! вывод результатов
End Program Pro1
```

Результаты работы Pro1

1. Файл *Out.txt*, в который выведены исходные данные (eps и *x)* и вычисленные значения *F, Sn, n, Delta* = ABS(F – Sn).

   Если *Delta* **<** *eps*, то результаты удовлетворительны.

   Соотношение *Delta* **>** *eps* означает, что ряд сходится, но не к ожидаемому значению. В этом случае следует проверить:

   – достижима ли задаваемая погрешность вычислений ε при использовании вещественных переменных типа Real. Возможно, следует использовать переменные типа Real*8;
   – рекуррентную формулу, алгоритм и программу.
2. Файлы *An.txt*, *Sn.txt*, *F.txt* с данными для графиков. Построив графики с помощью пакета *AGrapher*, убедитесь, что

$$\lim_{n\to\infty} A_n = 0, \qquad \lim_{n\to\infty} S_n = \sin(x).$$

Программа Pro2, дополненная циклом по значениям аргумента x

Для программы *Pro2* создать проект, скопировать в него текст *Pro1* для дальнейших изменений, которые следует проводить в два этапа.

1. Повторить суммирование для 10 – 15 значений *x* из [*xm , xk*] с шагом *xh* и вывести *неформатированные* результаты для каждого *x*

```
Program Pro2   ! суммирование ряда для различных значений x
     . . .
 Real:: xm=-1.6, xk=1.6, xh=0.2 ! границы и шаг изменения x
     !!! Исключить открытие файлов для AGrapher и вывод в них
   Do x = xm, xk+xh/2, xh    ! добавить цикл
     !*** по окончании суммирования ряда для x, выполненного как в Pro1
    Write(1,*) 'x=', x, ' F=', F, ' Sn=', Sn, &
      'n=', n, ' |F-Sn|=', Delta !вывод строки таблицы
   EndDo
End Program  Pro2
```

Рассмотреть результаты, сравнив *Delta* и *eps*; если удовлетворяется *Delta* **<** *eps* для всех значений *x*, перейти к следующему этапу.

2. Окончательная редакция программы с форматным выводом

a) добавить операторы `Format` для вывода строки таблицы

```
  Write(1,12) x, F, Sum, N, Delta !
   12 Format &     ! внутри цикла по x
('|',F6.1,2(' |', F7.4),' |',I7,4x,' |',e9.2,' |')
```

b) «шапку» и «донышко» таблицы можно "пририсовать" прямо в файле *out.txt*, выведенном по формату пункта (а), а затем скопировать получившиеся строки в форматы, подобные показанным ниже

```
  Write(1,10) eps  ! методическая погрешность и шапка таблицы
10 Format('Точность вычисления ряда ',E7.1 /& ! до цикла по x
  '+-------+---------+---------+-----------+----------+'/&
  '|   x   |  станд  | Тейлор  |Членов ряда| Разница  |'/&
  '+-------+---------+---------+-----------+----------+')
 11 Format &   ! «донышко» таблицы
 ('+-------+---------+---------+-----------+----------+')
    Write(1,11) ! после цикла по x  вывод «донышка» таблицы
```

Примечание. Выбор формата для вывода значений функции *f(x)* и суммы ряда определяется значением точности ε; например, при $\varepsilon = 10^{-3}$ достаточно выводить эти значения с точностью до третьего десятичного знака, при $\varepsilon = 10^{-5}$ – с точностью до пятого знака и т.д. (всего в числе 7 знаков).

Контрольные вопросы к защите работы

1. Что такое сходящийся ряд? Условие сходимости ряда (предел). Покажите на графике для своего задания.
2. При каком условии прекращается суммирование ряда в выбранном алгоритме? Каков геометрический смысл точности ε?
3. Зачем в алгоритм введено ограничение количества членов ряда? Каковы возможные причины нарушения этого ограничения? Значения каких переменных помогут понять его причину?
4. Зачем нужна рекуррентная формула?
5. Зачем нужна проверка вычислений? Проверьте коэффициент рекурсии для своего варианта работы на примере элемента a_{11}.
6. Что можно сказать о значении функции, вычисленном по стандартной программе, будет ли оно точным? Значения каких переменных вычисляются точно?
7. С какой целью в программах используют цикл *do while*?
8. С какой целью в программах используют оператор *exit*?
9. Напишите операторы, соответствующие выводу строки (символ «˜» означает пробел): «при˜x˜=˜0.05˜sin(x)˜=˜˜.85E-05»

10. Как компактнее записать формат

```
1 Format('|',F12.7, '|', F12.7, '|', F12.7, '|')
```

11. Исправьте ошибку

```
Real:: f;  integer:: n
write (1,7) f, n
7 Format (I6, 5x, '|', e11.4)
```

12. Как в программе *Pro1* заменить цикл `do while` бесконечным циклом `do`? Приведите блок-схему и фрагмент программы.
13. Что меняется в результатах программы *Pro2* при изменении значения точности ε, покажите на своих результатах. Объясните, почему.

Пример контрольного задания

1. Вывод формулы коэффициента рекурсии с проверкой для двух членов ряда (задача, аналогичная вариантам индивидуальных заданий).
2. Составьте блок-схему и напишите оператор `do while` для вывода в файл *Out.txt* значений a и $b = \sqrt{a^2 - 2}$, при $a = 8$ и каждом следующем a, равном половине предыдущего с противоположным знаком. Выполнение цикла прекратить при недопустимом значении a.
   a) a и b выводить по формату "E9.3" с названиями переменных.
   b) Показать *все* выводимые строки с пробелами.
   c) Чему равно значение a после выхода из цикла?
3. Как будут выглядеть выведенные строки (покажите с пробелами)? На каком устройстве?

```
Real:: f=2700000, z=-0.00017
write (*,4) f, z;  write (*, 5) f, z
          4 Format (F7.1);     5 Format (2E9.2)
```

4. Напишите *Format*, соответствующий выводу строки (символ «˜» означает пробел):

```
˜:˜˜.68E-04˜:˜˜.75E-02˜:˜˜.13E-03˜:
```

Таблица 8.

Варианты индивидуальных заданий «Сумма степенного ряда»

Значения a_0 и x_0 рекомендуются для программы *Pro1*.

№	Разложение функции в степенной ряд	Аргумент
1	$\cos(x+a) = \sum_{n=0}^{\infty} \frac{x^n \cos\left(a + \frac{n\pi}{2}\right)}{n!} = \cos a - x \cdot \sin a - \frac{x^2 \cos a}{2!} + \text{K}$ *Примечание* $\cos\left(\alpha + \frac{\pi}{2}\right) = -\sin\alpha$	$\lvert x \rvert < \infty$ $0 < a < \frac{\pi}{2}$ $a_0 = 1.5$ $x_0 = 1.5$

№	Разложение функции в степенной ряд	Аргумент
2	Гиперболический косинус $\mathrm{Ch}\ x = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \mathrm{K}$	$\lvert x\rvert < \infty$ *x_0 = 3.5*
3	$\ln\frac{x+1}{x-1} = 2\sum_{n=0}^{\infty} \frac{1}{(2n+1)x^{2n+1}} = 2\left(\frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \mathrm{K}\right)$	$\lvert x\rvert > 1$ *x_0 = 1.4*
4	$e^{-x} = \sum_{n=0}^{\infty} \frac{(-1)^n x^n}{n!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \mathrm{K}$	$\lvert x\rvert < \infty$ *x_0 = ±1.4*
5	$\mathrm{Arch}\,x = \ln 2x - \sum_{n=1}^{\infty} \frac{1\cdot 3\cdot 5\cdot \mathrm{K}\cdot(2n-1)}{2\cdot 4\cdot 6\cdot \mathrm{K}\cdot 2n\cdot 2n\cdot x^{2n}} =$ $= \ln 2x - \left[\frac{1}{2\cdot 2\cdot x^2} + \frac{1\cdot 3}{2\cdot 4\cdot 4\cdot x^4} + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6\cdot 6\cdot x^6} + \mathrm{K}\right]$ *Ареа-косинус* $\mathrm{Arch}x = \ln\left(x + \sqrt{x^2-1}\right)$ при $\lvert x\rvert > 1$	$x > 1$ *x_0 = 1.1*
6	$\arcsin x = x + \sum_{n=1}^{\infty} \frac{1\cdot 3\cdot \mathrm{K}\cdot(2n-1)\cdot x^{2n+1}}{2\cdot 4\cdot \mathrm{K}\cdot 2n\cdot(2n+1)} =$ $= x + \frac{x^3}{2\cdot 3} + \frac{1\cdot 3\cdot x^5}{2\cdot 4\cdot 5} + \frac{1\cdot 3\cdot 5\cdot x^7}{2\cdot 4\cdot 6\cdot 7} + \frac{1\cdot 3\cdot 5\cdot 7\cdot x^9}{2\cdot 4\cdot 6\cdot 8\cdot 9} + \mathrm{K}$	$\lvert x\rvert < 1$ *x_0 = 0.8*
7	$\ln(x+1) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{n+1}}{n+1} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \mathrm{K}$	$-1 < x \le 1$ *x_0 = ±0.6*
8	$\ln x = \sum_{n=0}^{\infty} \frac{(x-1)^{n+1}}{(n+1)x^{n+1}} = \frac{x-1}{x} + \frac{(x-1)^2}{2x^2} + \frac{(x-1)^3}{3x^3} + \mathrm{K}$	$x > \frac{1}{2}$ *x_0 = 0.6*
9	$\mathrm{arcctg}\ x = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1} x^{2n+1}}{2n+1} = \frac{\pi}{2} - x + \frac{x^3}{3} - \frac{x^5}{5} + \mathrm{K}$ *Примечание* $\mathrm{arcctg}\ x = \frac{\pi}{2} - \mathrm{arctg}\ x$	$\lvert x\rvert \le 1$ *x_0 = ±0.7*
10	$\mathrm{Arsh}x = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!\cdot x^{2n+1}}{4^n (n!)^2\cdot(2n+1)} = x - \frac{2!\cdot x^3}{4\cdot 3} + \frac{4!\cdot x^5}{4^2 (2!)^2\cdot 5} - \frac{6!\cdot x^7}{4^3 (3!)^2\cdot 7} + \mathrm{K}$ *Ареа-синус* $\mathrm{Arsh}\ x = \ln\left(x + \sqrt{x^2+1}\right)$ при $\lvert x\rvert < 1$	$\lvert x\rvert < 1$ *x_0 = 0.95*
11	$\ln\frac{1+x}{1-x} = 2\sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1} = 2\left[x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \mathrm{K}\right]$	$\lvert x\rvert < 1$ *x_0 = 0.8*

№	Разложение функции в степенной ряд	Аргумент
12	$\sin(x+a)=\sum_{n=0}^{\infty}\frac{x^n \sin\left(a+\frac{n\pi}{2}\right)}{n!}=\sin a + x\cdot\cos a-\frac{x^2\sin a}{2!}+\text{K}$ *Примечание* $\sin\left(\alpha+\frac{\pi}{2}\right)=\cos\alpha$	$\lvert x\rvert<\infty$ $0<a<\frac{\pi}{2}$ *a_0 = 1.5* *x_0 = 1.5*
13	$\text{Arth}\quad x=\sum_{n=0}^{\infty}\frac{x^{2n+1}}{2n+1}=x+\frac{x^3}{3}+\frac{x^5}{5}+\frac{x^7}{7}+\text{K}$ *Ареа-тангенс* $\text{Arth}\,x=\frac{1}{2}\ln\frac{1+x}{1-x}$ при $\lvert x\rvert<1$	$\lvert x\rvert<1$ *x_0 = 0.8*
14	$e^x=\sum_{n=0}^{\infty}\frac{x^n}{n!}=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\text{K}$	$\lvert x\rvert<\infty$ *x_0 = ±1.4*
15	Гиперболический синус $\text{Sh}\,x=\sum_{n=0}^{\infty}\frac{x^{2n+1}}{(2n+1)!}=x+\frac{x^3}{3!}+\frac{x^5}{5!}+\frac{x^7}{7!}+\text{K}$	$\lvert x\rvert<\infty$ *x_0 = 3.5*
16	$\ln x=2\sum_{n=0}^{\infty}\frac{(x-1)^{2n+1}}{(2n+1)(x+1)^{2n+1}}=2\left[\frac{x-1}{x+1}+\frac{(x-1)^3}{3(x+1)^3}+\frac{(x-1)^5}{5(x+1)^5}+\text{K}\right]$	$x>0$ *x_0 = 0.2*
17	$\text{arctg}\,x=\frac{\pi}{2}+\sum_{n=0}^{\infty}\frac{(-1)^{n+1}}{(2n+1)x^{2n+1}}=\frac{\pi}{2}-\frac{1}{x}+\frac{1}{3x^2}-\frac{1}{5x^5}+\text{K}$	$x>1$ *x_0 = 1.2*
18	$\arccos x=\frac{\pi}{2}-\sum_{n=0}^{\infty}\frac{(2n)!x^{2n+1}}{4^n(n!)^2(2n+1)}=\frac{\pi}{2}-\left[x+\frac{2!\cdot x^3}{4\cdot 3}+\frac{4!\cdot x^5}{4^2(2!)^2 5}+\frac{6!\cdot x^7}{4^3(3!)^2 7}+\text{K}\right]$	$\lvert x\rvert<1$ *x_0 = 0.8*
19	$\ln(1-x)=-\sum_{n=1}^{\infty}\frac{x^n}{n}=-\left[x+\frac{x^2}{2}+\frac{x^3}{3}+\frac{x^4}{4}+\text{K}\right]$	$-1\le x<1$ *x_0 = ±0.6*
20	$a^x=\sum_{n=0}^{\infty}\frac{(x\cdot\ln a)^n}{n!}=1+x\cdot\ln a+\frac{(x\cdot\ln a)^2}{2!}+\frac{(x\cdot\ln a)^3}{3!}+\text{K}$	$\lvert x\rvert<\infty$ *a_0 = 1.5* *x_0 = 3.5*
21	$\cos x=\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{(2n)!}=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\text{K}$	$\lvert x\rvert<\infty$ *x_0 = 2.5*
22	$\text{arctg}\,x=\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n+1}}{(2n+1)}=x-\frac{x^3}{3}+\frac{x^5}{5}-\frac{x^7}{7}+\text{K}$	$\lvert x\rvert\le 1$ *x_0 = 0.8*
23	$e^{-x^2}=\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{n!}=1-x^2+\frac{x^4}{2!}-\frac{x^6}{3!}+\frac{x^8}{4!}-\text{K}$	$\lvert x\rvert<\infty$ *x_0 = 1.2*

№	Разложение функции в степенной ряд	Аргумент
24	$\text{Arch } x = \ln 2x - \sum_{n=1}^{\infty} \frac{(2n)!}{4^n (n!)^2 \cdot 2n \cdot x^{2n}} =$ $= \ln 2x - \left[\frac{2!}{4 \cdot 2 \cdot x^2} + \frac{4!}{4^2 (2!)^2 \cdot 4 \cdot x^4} + \frac{6!}{4^3 (3!)^2 \cdot 6 \cdot x^6} + \text{K} \right]$ *ареа-косинус* $\text{Arch} x = \ln\left(x + \sqrt{x^2 - 1}\right)$ при $\lvert x \rvert > 1$	$x > 1$ $x_0 = 1.1$
25	$\ln x = \sum_{n=0}^{\infty} \frac{(-1)^n (x-1)^{n+1}}{(n+1)} = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} + \text{K}$	$0 < x \le 2$ $x_0 = 0.3$ $x_0 = 1.7$
26	$\frac{\sin x}{x} = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n+1)!} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} + \text{K}$	$\lvert x \rvert < \infty$ $X_0 = 4.5$
27	$\text{arctg} x = -\frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2n+1) x^{2n+1}} = -\frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \text{K}$	$x < -1$ $X_0 = -1.2$
28	$\text{Arcth } x = \sum_{n=0}^{\infty} \frac{1}{(2n+1) x^{2n+1}} = \frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \text{K}$ *ареа-котангес* $\text{Arcth} x = \frac{1}{2} \ln \frac{1+x}{x-1}$ при $\lvert x \rvert > 1$	$\lvert x \rvert > 1$ $X_0 = 1.3$
29	$\text{Arsh} x = x + \sum_{n=1}^{\infty} \frac{(-1)^n 1 \cdot 3 \cdot \text{K} \cdot (2n-1) \cdot x^{2n+1}}{2 \cdot 4 \cdot \text{K} \cdot 2n \cdot (2n+1)}$ $= x - \frac{x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} - \frac{1 \cdot 3 \cdot 5 \cdot x^7}{2 \cdot 4 \cdot 6 \cdot 7} + \text{K}$ *Ареа-синус* $\text{Arsh } x = \ln\left(x + \sqrt{x^2 + 1}\right)$ при $\lvert x \rvert < 1$	$\lvert x \rvert < 1$ $X_0 = 0.95$
30	$\text{arcsin } x = \sum_{n=0}^{\infty} \frac{(2n)! \, x^{2n+1}}{4^n (n!)^2 (2n+1)} =$ $= x + \frac{2! \cdot x^3}{4 \cdot 3} + \frac{4! \cdot x^5}{4^2 (2!)^2 5} + \frac{6! \cdot x^7}{4^3 (3!)^2 7} + \text{K}$	$\lvert x \rvert < 1$

2.4. Решение задач с одномерными массивами

Задание

Написать программу, в которой:

1. Ввести 12 вещественных чисел из файла *in.txt* в одномерный массив и вывести этот массив в результирующий файл *out.txt.*
2. Дополнительные параметры, если они есть в варианте задания, ввести в диалоге с клавиатуры и вывести в файл *out.txt.*

3. Вычислить значения трех переменных, которые входят в состав выражения, и значение выражения.
4. По мере вычисления переменных и выражения выводить их значения в файл *out.txt*, поясняя фразами из индивидуального задания.

Комментарии к заданию

1. При подготовке исходных данных в файле *In.txt* воспользуйтесь любым текстовым редактором. Для отладки в файле подготовьте числа, значения которых соответствуют индивидуальному заданию - среди них должны быть числа отрицательные, положительные, большие D и т. д.
2. Исходные данные и результаты выводите в файл *out.txt* только по формату.
3. Если вычисление какой-либо переменной невозможно из-за отсутствия в массиве подходящих элементов, вывести соответствующее сообщение (смотри пример программы).
4. Все переменные, используемые как индексы элементов массива и счетчики, должны быть *целого* типа.
5. Имена переменных в программе не должны совпадать с именами встроенных функций Фортрана.

Содержание отчета

1. Название работы и номер варианта задания.
2. Фамилия, имя, отчество и номер группы студента.
3. Текст варианта индивидуального задания.
4. *Единая* блок-схема программы, составленная из фрагментов.
5. Распечатка текста программы.
6. Распечатка файла *out.txt*.

Справочная информация

Таблица 9.

Средние значения множества вещественных чисел $a_1, a_2, a_3, \ldots, a_n$

Среднее арифметическое	$A_n = \frac{a_1+a_2+a_3+\ldots+a_n}{n}$
Среднее геометрическое	$G_n = \sqrt[n]{a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_n}$
Среднее квадратичное	$Q_n = \sqrt{\frac{1}{n}\left(a_1^2 + a_2^2 \ldots + a_n^2\right)}$
Среднее гармоническое	$M_n = \frac{n}{\frac{1}{a_1}+\frac{1}{a_2}+\frac{1}{a_3}+\ldots+\frac{1}{a_n}}$, где $a_i \neq 0$, n – количество ненулевых элементов

Пример

В массиве *M* из 12 вещественных чисел (должны быть отрицательные, положительные и равные нулю) найти:

1. *A* – среднее арифметическое отрицательных элементов, больших *D* (*D* <0, ввести с клавиатуры),
2. *B* – номер минимального положительного элемента,
3. C – среднее гармоническое ненулевых элементов с четными номерами.

Вычислить $Z = A + B – C$.

Блок-схемы

Далее приведены 3 *фрагмента* блок-схемы программы. Для отчета следует из фрагментов составить *единую* блок-схему программы ***Vector***.

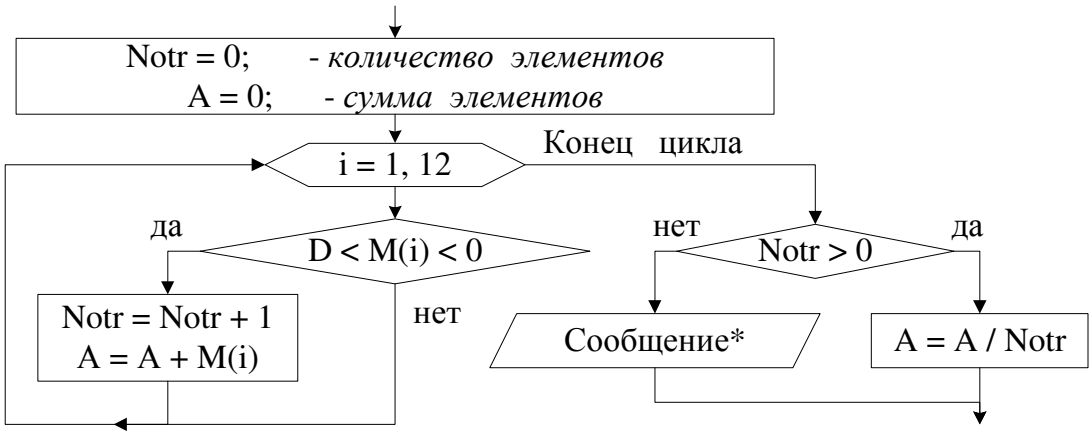


Рис.4. A – среднеарифметическое отрицательных, больших D

* - сообщение: «В массиве нет отрицательных элементов, больших D»

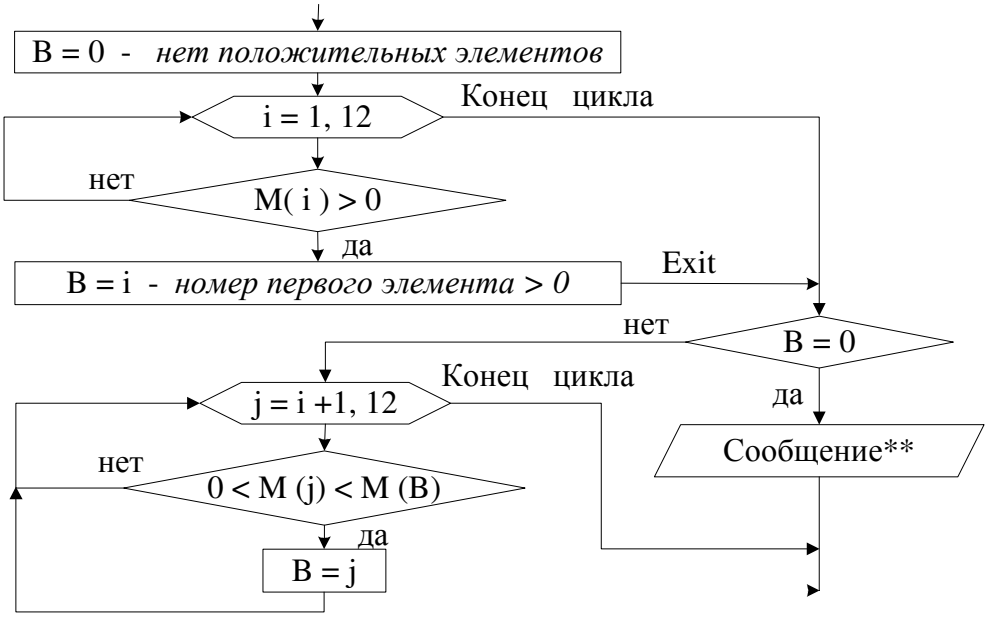


Рис.5. B – номер минимального положительного элемента

** - сообщение: «*В массиве нет положительных элементов*»

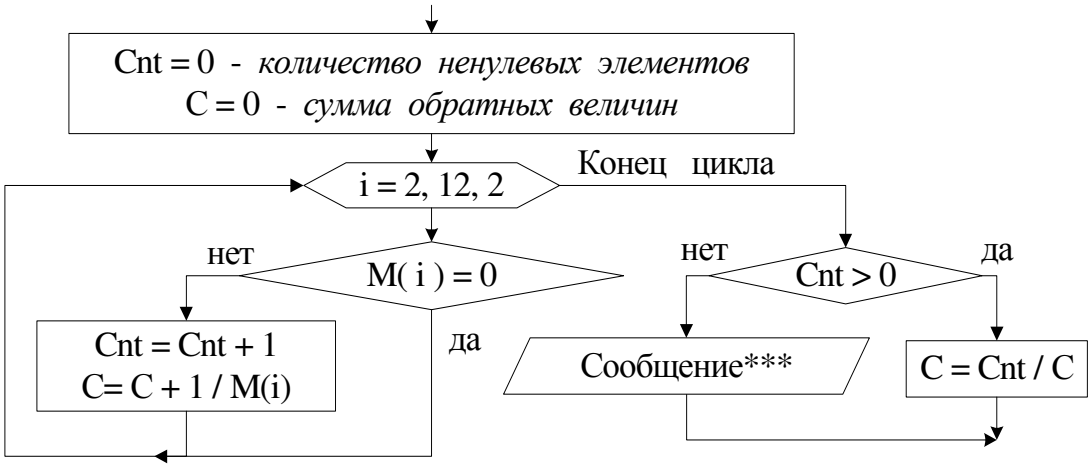


Рис.6. C – средне гармоническое ненулевых элементов с четными номерами

*** - сообщение: «*Все элементы массива с четными номерами = 0*»

Программа (пример)

```
Program Vector
! студент (фамилия, имя)  группа №  работа №  вариант №
 Implicit None
  Real,dimension(1:12):: M
    Real:: A, C, D, Z
     Integer B, i, Notr, Cnt
Open(1,FILE='In.txt') ! файл с исходным массивом
Open(2,FILE ='Out.txt') ! результирующий файл
Write(2,*) 'Программа Vector'
Read(1,*)M ! ввод массива
Write(2,10) M ! вывод массива в 2 колонки с заголовком
10 Format (?????)! допишите оператор Format
Write(*,*) 'Input D < 0' ! диалог для ввода значения D <0
Read(*,*) D
Write(2,14) 'D =', D, '- должно быть отрицательным'
14 Format (?????)! допишите оператор Format
! *****
```

По блок-схемам, составленным с учетом индивидуального задания, написать операторы программы Vector для вычисления и форматного вывода трех переменных и результирующего выражения. Выводить значения переменных по мере их вычисления, поясняя формулировками из индивидуального задания.

```
! *****
End Program Vector
```

Результаты программы Vector

Содержимое файла *Out.txt* в результате работы программы *Vector* для приведенного примера задания, пояснения пишите по-русски:

Программа Vector
Исходный массив
· · · 6 строк в 2 колонки
`D = -12.00` – должно быть отрицательным
`A = -17.05` – среднее арифметическое элементов D<M<0
`B =  10` – номер минимального положительного элемента
`C =  25.13` – средне гармоническое элементов с нечетными номерами
`Z =  34.77 = A + B – C`

Контрольные вопросы

1. Что в программировании называют массивом?
2. Основные характеристики одномерного массива.
3. Ранг массива; приведите примеры массивов 0, 1, 2 ранга и их математическую интерпретацию.
4. Примеры описания одномерного массива, диапазон индекса. Что в программе объявляют операторы?
   ```
   Logical C(7), D(7), B(-7:0), U(0:7)
   Integer A(-3:3)
   Real E(0:6)
   ```
   Какие массивы конформны?
   Как лучше описать эти массивы в Фортран-90?
5. Порядок размещения элементов массива в памяти ПК.
6. В программе объявлен одномерный массив A с нумерацией элементов от 1 до 12. Какой триплет задаёт секцию:
   - из всех элементов массива в обратном порядке?
   - из последних 8 элементов массива?
   - из первой трети массива?
   - из элементов массива с нечетными номерами?
7. Как работает оператор `where`? Приведите примеры оператора `where` а) без блока; в) с одним блоком; с) с двумя блоками.
8. В программе объявлен массив: `Real,dimension(1:10)::A`

   Что произойдет при выполнении каждого из операторов?
   ```
   Read(1,*) A(10)
   Write(*,*) A(10)
   Read(*,*) A(11)
   Write(*,*) A(11)
   Write(*,11) A(8:10)
   ```
   Напишите оператор *format* для последнего оператора *Write*.
   Как вывести 6 последних элементов массива A?

Пример контрольного задания

1. Дан массив из 20 элементов. Написать программу нахождения номера максимального по модулю среди первых 8 элементов массива (аналогичные задачи смотри в вариантах заданий). Блок-схема обязательна. Вывести значение по формату.
2. В программе используется вещественный массив из 100 элементов. Напишите операторы для вывода элементов массива в виде:
   a) Заголовок «Исходный массив»;
   b) Элементы массива по четыре в строке.
3. В файле *data.txt* – 12 чисел натурального ряда. Как будут выглядеть выведенные на экран строки (с пробелами)?

```
Integer, dimension(1:12):: Mas = 0
Open(1,file = 'data.txt')
Read(1,*) Mas(3), Mas(5)
Write(*,11) Mas(3:5)
        11 Format (I4)
```

Таблица 10.

Варианты индивидуальных заданий «Одномерные массивы»

№	Выражение	Определение переменных
1	$B + \frac{A}{C+1}$	A – сумма отрицательных элементов с четными номерами; B – максимальный элемент среди *N* первых элементов; C – среднее геометрическое положительных элементов.
2	$\frac{R+Q+S}{R \cdot Q \cdot S + 2}$	R – произведение положительных элементов с нечетными номерами; Q – последний положительный элемент с четным номером; S – среднее гармоническое положительных элементов.
3	$\left(H + \frac{E}{H+1}\right)G$	H – количество нулей среди *N* последних элементов; E – номер минимального элемента; G – среднее квадратичное элементов с четными номерами.
4	$U + \frac{1}{\lvert S+T \rvert + 1}$	S – сумма положительных элементов, меньших *D;* T – минимальный по модулю ненулевой элемент (со знаком); U – среднее арифметическое *N* первых элементов.
5	$Z + \frac{X}{10+Y}$	X – количество элементов со значениями из интервала [*A*, *B*]; Y – модуль минимального элемента; Z – среднее арифметическое элементов с нечетными номерами.
6	$\frac{U}{R+1} + S$	U – произведение элементов с четными номерами; R – номер максимального по модулю элемента; S – среднее арифметическое *N* последних элементов.

№	Выражение	Определение переменных
7	$\frac{A \cdot B}{C+2}$	A – произведение ненулевых элементов; B – последний отрицательный элемент с нечетным номером; C – среднее арифметическое положительных элементов.
8	$\frac{R}{Q+1}+S$	R – сумма всех элементов; Q – номер первого нулевого элемента; S – среднее геометрическое положительных среди *N* последних элементов.
9	$\frac{H}{(E+1)(H+1)}+G$	H – сумма положительных среди *N* первых элементов; E – номер первого отрицательного элемента; G – среднее арифметическое отрицательных элементов.
10	$C+\frac{A}{10}+\frac{B}{10+A}$	A – первый положительный элемент с четным номером; B – сумма элементов с нечетными номерами; C – среднее гармоническое ненулевых среди *N* первых элементов.
11	$Y+\frac{X}{10+Z}$	X – количество элементов, меньших *D,* с нечетными номерами; Y – максимальный по модулю элемент (со знаком); Z – среднее квадратичное положительных элементов.
12	$C+\frac{A+B}{A \cdot B+1}$	A – произведение положительных элементов; B – последний положительный элемент с четным номером; C – среднее квадратичное *N* первых элементов.
13	$\frac{T \cdot V+U}{U+1}$	V – сумма отрицательных элементов, больших *D, (D<0)* ; T – номер максимального элемента; U – среднее геометрическое положительных среди *N* первых элементов.
14	$(U+T)\ (S+2)$	S – сумма модулей отрицательных элементов; T – номер минимального по модулю ненулевого элемента; U – среднее квадратичное *N* последних элементов.
15	$\frac{H}{(E+1)(H+1)}+G$	H – произведение модулей отрицательных элементов; E – номер последнего положительного элемента: G – среднее арифметическое элементов с четными номерами.
16	$\frac{X+Z}{Y+2}$	X – сумма элементов, больших *D,* с четными номерами; Y – номер максимального отрицательного элемента; Z – среднее квадратичное всех элементов.
17	$B+\frac{A}{C+1}$	A – среднее гармоническое отрицательных элементов; B – номер второго нулевого элемента; C – количество отрицательных элементов с четными номерами.
18	$C+\frac{A}{10}+\frac{B}{10+A}$	A – сумма положительных элементов; B – номер минимального положительного элемента; C – среднее гармоническое ненулевых элементов.

№	Выражение	Определение переменных
19	$\frac{X+Z}{Y+2}$	X – количество положительных элементов с четными номерами; Y – номер последнего нулевого элемента: Z – среднее гармоническое ненулевых элементов, больших *D*.
20	$\frac{A \cdot B}{C+2}$	A – произведение отрицательных элементов; B – максимальный по модулю элемент (со знаком); C – среднее арифметическое модулей отрицательных элементов.
21	$\left(H+\frac{E}{H+1}\right)G$	H – количество отрицательных элементов; E – минимальный элемент среди *N* последних элементов; G – среднее квадратичное элементов с четными номерами.
22	$U+\frac{1}{S+T+1}$	S – количество положительных элементов; T – минимальный положительный элемент; U – среднее арифметическое элементов.
23	$\frac{U}{R+1}+S$	U – произведение ненулевых среди *N* первых элементов; R – номер максимального по модулю элемента; S – среднее геометрическое положительных элементов.
24	$\frac{R}{Q+1}+S$	R – максимальный элемент с четным номером; Q – количество ненулевых элементов; S – среднее гармоническое положительных элементов.
25	$(U+T)\ (S+2)$	S – сумма *N* последних элементов; T – максимальный отрицательный элемент; U – среднее квадратичное элементов с четными номерами.
26	$C+\frac{A+B}{A \cdot B+1}$	A – количество элементов, больших *D;* B – первый положительный элемент с четным номером; C – среднее арифметическое *N* первых элементов.
27	$\frac{H+1}{(C+2)E}$	H – сумма элементов со значениями из интервала [*A*, B]; E – последний положительный элемент с четным номером; C – среднее арифметическое положительных элементов.
28	$\frac{(H+E) \cdot C}{(E+C+4)}$	H – произведение элементов со значениями из интервала [A, B]; E – номер минимального по модулю ненулевого элемента; C – среднее геометрическое положительных среди *N* последних элементов.
29	$\frac{R+Q+S}{R \cdot Q \cdot S+2}$	R – количество элементов, меньших *D;* Q – модуль минимального элемента; S – среднее квадратичное положительных элементов.
30	$\frac{R}{R+1}+U+T$	U – минимальный элемент с номером из интервала [*K, L*]; T – среднее арифметическое элементов с нечетными номерами. R – сколько нулей среди *N* последних элементов.

2.5. Решение задач с двумерными массивами

Задание

1. Написать программу, которая:
   a) вводит из файла *In.txt* на диске количество строк и столбцов, динамически размещает матрицу; вводит её;
   b) выводит по формату матрицу *A* с заголовком в файл *Out.txt*;
   c) вводит с клавиатуры дополнительные параметры, если они есть в варианте индивидуального задания;
   d) выводит по формату результаты работы программы, поясняя смысл каждого значения формулировками из индивидуального задания.
2. Дополнить *программу* внешним циклом, в котором последовательно ввести матрицы разных размеров и выполнить для каждой из них индивидуальное задание.

Содержание отчета

1. Название работы и номер варианта индивидуального задания.
2. Фамилия, имя, отчество и номер группы студента.
3. Текст варианта индивидуального задания.
4. Блок-схема программы.
5. Распечатка текста программы.
6. Распечатка файла результатов *Out.txt.*

Комментарии к заданию

1. Каждую матрицу готовить в файле *In.txt* следующим образом:
   a) *M*, *N* (*M* и *N* – количество строк и столбцов матрицы)
   b) Матрица, содержащая *M* строк и *N* столбцов.
   Далее пункты a) и b) повторяются для каждой следующей матрицы.
2. Рекомендуется выбрать $M \le 10$ и $N \le 10$ и для контроля вывести *M* и *N* в файл результатов *Out.txt*.

Справочная информация

Размещение двумерного массива в памяти компьютера

В Фортране принято, что элементы двумерного массива размещаются в линейной памяти компьютера по столбцам, то есть первый индекс массива изменяется быстрее второго. Например, матрица $\begin{Bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{Bmatrix}$ размещается в массиве *A(1:2, 1:3)* в следующем порядке: $\{a_{11}a_{21}\ a_{12}a_{22}\ a_{13}a_{23}\}$.

Для матрицы, подготовленной в текстовом файле, как в математике – по строкам $\{a_{11}a_{12}a_{13}\ a_{21}a_{22}a_{23}\}$, чтение *Read(1,*) A* приведет к *ошибочному* размещению $\{a_{11}a_{12}\ \ a_{13}a_{21}\ \ a_{22}a_{23}\}$. Оператор *Write(2,*) A*

выводит массив в порядке размещения его в памяти, т.е. по столбцам $\{a_{11}a_{21}a_{12}\ a_{22}a_{13}a_{23}\}$.

Внимание! Последовательность операторов

```
Read(1,*) A;    Write(2,*) A
```

приведет к получению при выводе как будто *правильной* матрицы, хотя из-за неправильного размещения массива в памяти результаты работы программы будут неожиданными.

Ввод и вывод двумерного массива

Ввод матрицы делают построчным и бесформатным:

```
Integer,parameter:: kolStr=2, kolStlb=3
                  ! количество строк  и столбцов
Integer::str,stlb ! номера строки и столбца
Real,dimension(1:kolStr,1:kolStlb):: A !массив
! либо диапазон индекса столбца = «:» – по умолчанию [1:kolStlb]
Read(1,*)(A(str,:),str=1,kolStr)
! либо циклы по секциям-строкам:
Read(1,*)(A(str,1:kolStlb), str=1,kolStr)
! либо вложенные неявные циклы – по столбцам внутри строк
Read(1,*)((A(str,stlb),stlb=1,:),str=1,kolStr)
```

Вывод массива по строкам аналогичен, но обязательно по формату.

```
Write(2,1)(A(str,:),str=1, kolStr)
1 format('Матрица'/3(f5.1))
```

В операторе `Format` целый повторитель *kolStlb* – длина строки матрицы:

```
1 format('Матрица'/<kolStlb>(f5.1))
```

Динамические массивы.

Размер *статического* массива не может быть изменен в процессе вычислений, что не всегда удобно при решении задач. Этого неудобства можно избежать, применяя *динамические* массивы. Работа с динамическим массивом требует (смотри пример программы):

a) описать динамический массив с атрибутом `Allocatable`, указывая количество измерений и не указывая размеров;
b) определить размер массива оператором ввода (для двумерного массива – количество строк и столбцов, для одномерного – количество элементов);
c) разместить массив в памяти оператором `Allocate`;
d) выполнить действия с массивом;
e) освободить память оператором `Deallocate`.

Пример

В каждой строке *целочисленной* матрицы найти первый элемент, не равный нулю, и сменить его знак. Вывести исходную матрицу, коорди-

наты найденных в каждой строке элементов и результирующую матрицу. О строках со всеми нулями вывести сообщение с номером строки.

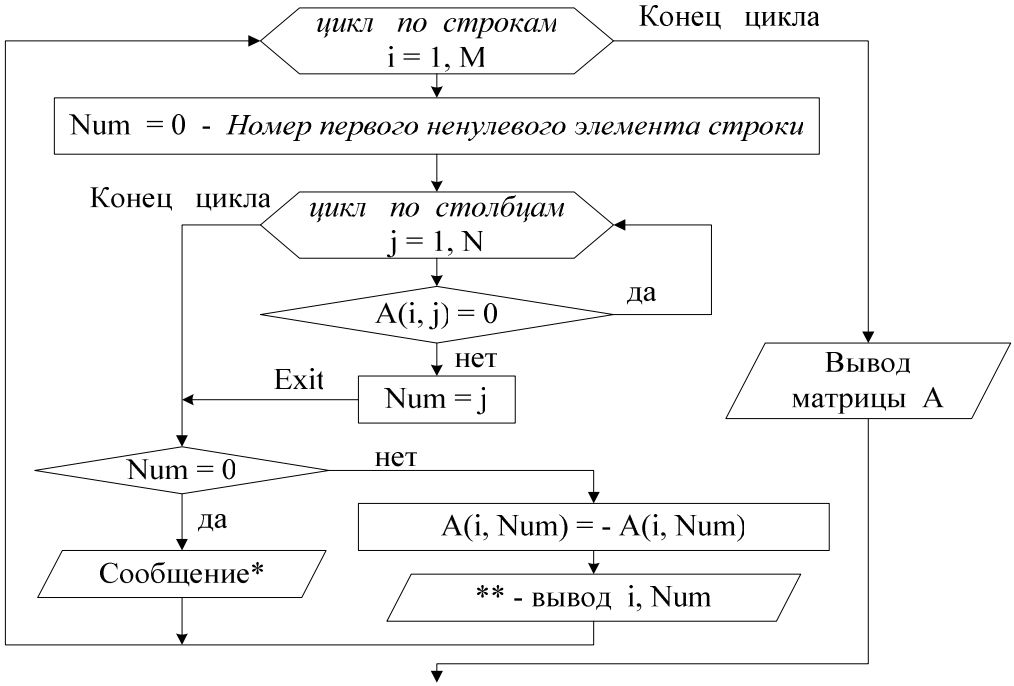


Рис.7. Блок-схема алгоритма для матрицы

* — *«В строке i все элементы равны 0»*

** — *«Координаты первого ненулевого элемента строки = [i, Num]»*

Программа

```
Program Matrix;    Implicit None
! студент (фамилия, имя)  группа №  работа №  вариант №
Integer,Allocatable,dimension(:,:)::A ! динамическая матрица
Integer,Allocatable,dimension(:):: B ! динамический массив
Integer varStr,varStlb ! переменные размеры матрицы по измерениям
 Integer i, j  ! индексы строк и столбцов
Open(1,file='in.txt')     ! файл с исходной матрицей
Open(2,file='out.txt')    ! результирующий файл
Do  ! цикл по нескольким матрицам разной формы
 Read(1,*,end=10) varStr,varStlb  ! ввод размеров (строк, столбцов)
  ! размещение динамических массивов varStr строк, varStlb столбцов
    Allocate(A(1:varStr,1:varStlb),B(1:varStlb))
  ! ввод массива - неявный цикл по секциям-строкам
    Read(1,*) ( A(i,:), i = 1,varStr )
  ! По блок-схеме, составленной по заданию, написать программу
  ! Не забудьте добавить необходимые описания переменных и массивов
   Deallocate(A)    ! освобождение динамической памяти
Enddo
End Program Matrix
```

Результаты работы программы

Содержимое файла Out.txt с одной матрицей:

```
В исходной матрице 3 строки и 5 столбцов
Исходная матрица
     0  -123   -33    12    -5
     0     0     0     0     0
     0     0    12     0     1
Координаты первого ненулевого элемента строки = [ 1, 2]
В строке 2 все элементы равны 0
Координаты первого ненулевого элемента строки = [ 3, 3]
Преобразованная матрица
     0   123   -33    12    -5
     0     0     0     0     0
     0     0   -12     0     1
```

Контрольные вопросы

1. Что такое двумерный массив? Его аналоги в математике на примере двух массивов. Описание двумерных массивов в *Fortran90*, диапазоны индексов.
2. Форма – основная характеристика массива, напишите примеры *конформных* (одинаковых по форме) массивов.
3. Назовите атрибуты массива, определяющие характеристики, не связанные с его размерностью.
4. Укажите порядок размещения элементов двумерных массивов в памяти компьютера. Приведите пример.
5. В программе объявлен двумерный массив *A(12,12)*. Как его описать в *Fortran90*? Как записать в программе ссылку: а) на полный массив, б) на секцию массива, в) на элемент массива?
6. Допустим ли оператор присваивания к массивам, объявленным ниже? Почему?

```
Real A(2, -3:3), B(0:1, 0:6), C(-2:-1, 1:7)
A = cos(B**2)- 3*C
```

7. Поясните правила записи выражений в операторах присваивания с массивами на примерах.
8. Как работает оператор *where?* Приведите примеры оператора *where* а) без блока; в) с одним блоком; с) с двумя блоками.
9. Что произойдет при выполнении оператора *Read*?

```
Real,dimension(1:12,1:12) A
Read (1,*) A(1, 12)
```

10. Как ввести первую строку массива, объявленного оператором

```
Real, dimension(1:20,1:12):: A?
```

11. Как вывести вторую строку массива, объявленного оператором

```
Real, dimension(1:20,1:12):: A?
```

12. Форматный вывод двумерного массива на примерах.

13. Что будет выведено при выполнении оператора *Write*?

```
Real, dimension(1:12, 1:12):: A
Write(*,*) A(12,12)
```

14. Что и как будет выведено при выполнении оператора *Write*?

```
Real, dimension(1:12, 1:12):: A
Write (*,11) A(10:12, 10:12)
 11 Format(F10.2)
```

15. Объявите вещественный массив из 100 строк и 5 столбцов и напишите оператор *Format* и оператор вывода элементов массива:
    a) заголовок «Результаты работы»;
    b) далее элементы массива в виде матрицы.

Пример контрольного задания

1. Составить и вывести одномерный массив из сумм положительных элементов с четными номерами строк в каждом столбце матрицы *A*. Составить блок-схему и написать программу. Вывести по формату исходную матрицу и полученный массив.

2. Записать операторы для вывода элементов правой нижней четверти матрицы. Массив объявлен оператором:
   `Real,dimension(1:8, 1:10):: A`

3. В файле *data.txt* подготовлены данные в виде целочисленной матрицы

11	12	13	14	15
21	22	23	24	25
31	32	33	34	35

   Что будет выведено на экран монитора (с пробелами) в результате работы программы, если добавить вывод массива?

```
Integer, dimension(3,5):: Mas = 0
Open(1,file = 'data.txt')
Read(1,*) Mas(2, 1:5:2)
   ! добавьте вывод всего массива по формату
End
```

Таблица 11.

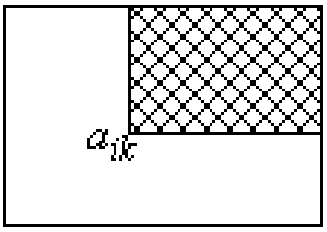
Варианты индивидуальных заданий «Двумерные массивы»

Вариант 1

Сформировать одномерный массив из элементов, расположенных по «периметру» матрицы (элементы первой и последней строк и первого и последнего столбца). Элементы «периметра» выбирать по часовой стрелке, начиная с верхнего левого элемента (элемента a_{11}). Вывести сформированный одномерный массив.

Вариант 2

Сформировать матрицу *B* из элементов матрицы *A*. Каждый элемент b_{ik} должен быть равен сумме элементов матрицы *A* из области, определяемой индексами *i,k* в соответствии с рисунком. Вывести результирующую матрицу.



Вариант 3

Отобрать в матрицу *B* только строки матрицы *A*, элементы которых расположены в порядке возрастания. Если в матрице *A* таких строк нет, вывести сообщение, иначе вывести:

- номера строк матрицы *A*, помещенных в матрицу *B*;
- полученную матрицу *B*.

Вариант 4

В каждой строке матрицы каждый отрицательный элемент, стоящий между двумя положительными, заменить их полусуммой. Вывести преобразованную матрицу.

Вариант 5

Записать в одномерный массив максимальные отрицательные элементы каждой строки матрицы. Для строк матрицы, не содержащих отрицательных элементов, в результирующий массив записать 0. Вывести полученный одномерный массив.

Вариант 6

Найти среднее арифметическое значение максимальных по модулю элементов строк матрицы. Вывести:

- значения максимальных по модулю элементов строк (со знаком);
- полученное значение.

Вариант 7

В каждом столбце *целочисленной* матрицы подсчитать сумму положительных элементов с четными значениями и записать ее в одномерный массив. Вывести полученный массив.

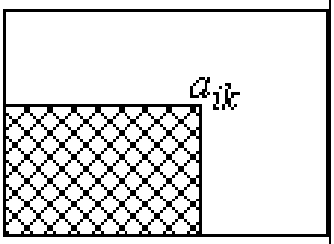
Вариант 8

Вычислить среднее арифметическое значение элементов каждого столбца матрицы без учета минимального и максимального элементов этого столбца; записать это значение в одномерный массив. Вывести:

- значения минимума и максимума столбцов;
- полученный одномерный массив.

Вариант 9

Сформировать матрицу *B* из элементов матрицы *A*. Каждый элемент b_{ik} должен быть равен сумме элементов матрицы *A* из области, определяемой индексами *i,k* в соответствии с рисунком.



Вывести результирующую матрицу.

Вариант 10

Найти максимальный и минимальный по модулю ненулевые элементы матрицы и поменять их местами. Вывести:

- координаты найденных элементов;
- преобразованную матрицу.

Вариант 11

Если сумма положительных элементов матрицы больше произведения модулей ее отрицательных элементов, сформировать одномерный массив из положительных элементов матрицы, иначе массив формировать из ее отрицательных элементов. При формировании одномерного массива матрицу просматривать построчно. Вывести:

- вычисленные значения суммы и произведения;
- полученный одномерный массив.

Вариант 12

В каждом столбце *целочисленной* матрицы найти количество пар рядом стоящих одинаковых элементов. Результаты записать в одномерный массив. Вывести полученный массив.

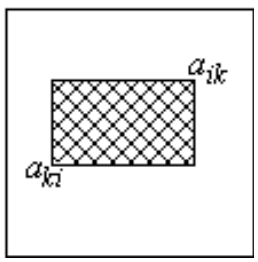
Вариант 13

Преобразовать *целочисленную* матрицу, удалив из нее строки, содержащие нулевые элементы, и сдвинув оставшиеся строки. Если нулевых элементов в матрице нет, вывести сообщение. Вывести:

- номера удаленных строк или сообщение;
- преобразованную матрицу.

Вариант 14

Сформировать квадратную матрицу *B* из элементов квадратной матрицы *A*. Каждый элемент b_{ik} должен быть равен сумме элементов матрицы *A* из области, определяемой индексами *i,k* в соответствии с рисунком.



Вывести результирующую матрицу.

Вариант 15

Преобразовать *целочисленную* матрицу, переставив на первое место столбец с наибольшим количеством нулей (сдвинуть остальные столбцы). При отсутствии в матрице нулевых элементов вывести сообщение, иначе вывести преобразованную матрицу.

Вариант 16

Если в каждой строке *целочисленной* матрицы *A* есть равный нулю элемент, то сформировать матрицу *B* из строк матрицы *A* с четными номерами, иначе - из ее строк с нечетными номерами. Вывести:

- номера строк, в которых нет нулей;
- полученную матрицу.

Вариант 17

Записать в одномерный массив столбец матрицы с максимальным количеством отрицательных элементов. При отсутствии отрицательных элементов в матрице вывести сообщение. Вывести:

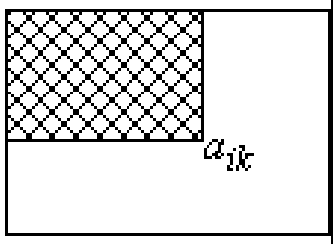
- количество отрицательных элементов в столбцах или сообщение;
- результирующий одномерный массив.

Вариант 18

Назовем допустимым преобразованием матрицы перестановку двух строк или двух столбцов. С помощью допустимых преобразований добиться того, чтобы наибольший по модулю элемент матрицы располагался в ее левом верхнем углу. Вывести преобразованную матрицу после каждого перемещения строки или столбца.

Вариант 19

Сформировать матрицу *B* из элементов матрицы *A*. Каждый элемент b_{ik} должен быть равен максимальному элементу матрицы *A* из области, определяемой индексами *i,k* в соответствии с рисунком.



Вывести результирующую матрицу.

Вариант 20

Найти координаты и значение максимума из минимальных элементов каждого столбца матрицы. Вывести:

- координаты минимальных элементов столбцов;
- координаты и значение максимума.

Вариант 21

Найти минимальный по модулю ненулевой элемент в каждой строке *квадратной* матрицы и поменять его местами с элементом этой же строки, находящимся на побочной диагонали матрицы. Вывести

- координаты найденных в строках элементов;
- преобразованную матрицу.

Вариант 22

Рассматривая построчно *целочисленную* матрицу *A,* сформировать три одномерных массива, поместив в массив *B* - четные элементы, в массив *C* - элементы, кратные 3, в массив *D* - элементы, кратные 5. Возможна запись элемента в два или три массива. Вывести полученные одномерные массивы.

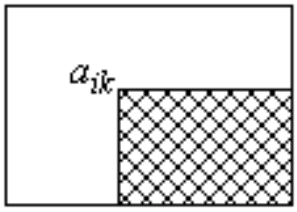
Вариант 23

Если модуль суммы отрицательных элементов *целочисленной* матрицы больше суммы ее положительных элементов, сформировать *одномерный* массив из первых элементов строк матрицы, иначе массив формируется из последних элементов строк матрицы. Если в матрице нет положительных или отрицательных элементов, вывести сообщение, иначе вывести:

- суммы отрицательных и положительных элементов матрицы;
- результирующий массив.

Вариант 24

Сформировать матрицу *B* из элементов матрицы *A*. Каждый элемент b_{ik} должен быть равен максимальному элементу матрицы *A* из области, определяемой индексами *i,k* в соответствии с рисунком.



Вывести результирующую матрицу.

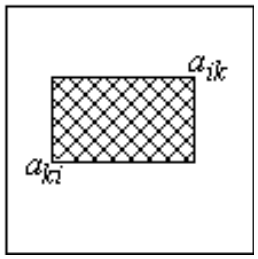
Вариант 25

Поменять местами первый и второй положительные элементы в каждом столбце матрицы. Для столбцов матрицы, содержащих менее двух положительных элементов, вывести сообщение. Вывести:

- координаты элементов, найденных в каждом столбце;
- преобразованную матрицу.

Вариант 26

Сформировать квадратную матрицу *B* из элементов квадратной матрицы *A*. Каждый элемент b_{ik} должен быть равен максимальному элементу матрицы *A* из области, определяемой индексами *i,k* в соответствии с рисунком. Вывести результирующую матрицу.



Вариант 27

Заменить максимальный элемент матрицы *A* средним арифметическим его соседей. Соседями элемента a_{ij} в матрице считать элементы $a_{i-1,j}$, $a_{i+1,j}$, $a_{i,j-1}$, $a_{i,j+1}$.Следует учесть, что в зависимости от положения элемента a_{ij} в матрице у него может быть 2, 3 или 4 соседа. Вывести

- координаты и значение максимального элемента матрицы;
- преобразованную матрицу.

Вариант 28

Умножить матрицу *A* на вектор *B*. Произведением матрицы A(M, N) и вектора B(N) является вектор C(M), каждый элемент которого вычисляется по формуле $c_j = \sum_{k=1}^{n} a_{jk} \cdot b_k$. Вывести полученный массив.

Вариант 29

Удалить из *целочисленной* матрицы строки и столбцы, заполненные нулями (оставшиеся строки и столбцы сдвинуть). Вывести результирующую матрицу после каждого удаления (строк или столбцов).

Вариант 30

Записать в одномерный массив модули разности первого и последнего отрицательных элементов каждого столбца матрицы. Для столбцов, содержащих менее двух отрицательных элементов, в результирующий массив записать (-1) и вывести сообщение. Вывести:

- для каждого столбца – значения первого и последнего отрицательных элементов или сообщение;
- полученный одномерный массив.

2.6. Решение задач с использованием процедур

Задание

1. В соответствии с индивидуальным заданием составить блок-схему и текст процедуры (функции или подпрограммы).
2. Написать главную программу, которая должна содержать:
   а) интерфейс с правилами применения каждой процедуры;

b) ввод данных из файла *In.txt* на диске и форматный вывод их в результирующий файл *Out.txt*;

c) вызовы процедур с различными данными;

d) форматный вывод в файл *Out.txt* результатов работы каждой процедуры, пояснив их формулировками из индивидуального задания;

e) если возможно, выполнить проверку результатов работы процедур обращением к стандартным подпрограммам или функциям;

f) обработку результатов процедур в соответствии с индивидуальным заданием на главную программу и форматный вывод в файл *Out.txt* результатов с пояснениями.

Содержание отчета

1. Название работы и номер варианта индивидуального задания.
2. Фамилия, имя, отчество и номер группы студента.
3. Текст варианта индивидуального задания.
4. Постановка задачи, методы вычислений.
5. Графическая интерпретация (если требуется), выполненная вручную или в пакете *Agrapher*.
6. Блок-схемы главной программы и процедур.
7. Распечатка текста программы.
8. Распечатка файла результатов *Out.txt.*

При подготовке к защите ответить на контрольные вопросы.

Справочная информация

Таблица 12.

Функции для определения размеров массивов (*Array* – массив).

Функция	**Результат функции**	**Примечания**
Size(Array)	Целое число, количество элементов массива *Array*.	Рекомендуется для определения длины одномерного массива.
Size(Array,dim)	Целое число - протяженность вдоль заданного измерения.	При dim = 1 - количество столбцов двумерного массива, при dim = 2 – количество строк.
Shape(Array)	Целочисленный одномерный массив из протяженностей по каждому измерению массива.	Размер результирующего массива равен количеству измерений массива *Array* (для двумерного массива – 2)

Таблица 13.

Некоторые полезные встроенные функции:

Функция	**Аргументы**	**Результат**
Произведение матриц `MC = Matmul(MA,MB)`	*MA* – матрица $M \times N$ *MB* – матрица $N \times P$	*MC* – матрица $M \times P$
Умножение матрицы на вектор `V2 = Matmul(MA,V1)`	*MA* – матрица $M \times N$ *V1* – вектор N элементов	*V2* – вектор из M элементов
Транспонирование матрицы `MB=Transpose(MA)`	*MA* – матрица $M \times N$	*MB* – матрица $N \times M$
Скалярное произведение векторов `S=Dot_product(V1,V2)`	*V1*, *V2* – векторы одинаковой длины	*S* – число (скаляр)
Генератор случайных чисел `Call Random_number(M1)`	Выходной аргумент M1 – вещественная переменная или массив – псевдослучайные числа из [0; 1]	

Комментарии к заданию

1. В составе данного проекта независимые программные единицы – главная программа и процедуры, каждая с собственной системой обозначений объектов.
2. Правила применения каждой процедуры должны быть описаны в вызывающей программе в операторе *Interface*.
3. Данные передаются сопоставлением *аргументов – фактических* в главной программе и *формальных* в процедуре.
4. Функция – это процедура, вызываемая из выражения по имени.
5. Аргументы функции *по назначению* – только *входные*, единственный результат связан с именем функции.
6. Результаты подпрограммы передаются в вызывающую программу через *выходные* аргументы (атрибут `Intent(Out)`) и не связан с именем подпрограммы. Количество выходных аргументов может быть любым, в том числе один или ни одного.
7. Количество фактических и формальных аргументов должно совпадать. Кроме того, попарно в порядке следования фактические и формальные аргументы должны соответствовать по типу, назначению и форме. Совпадение имен фактических и формальных аргументов не требуется, но и не запрещается.
8. Атрибут назначения каждого формального аргумента должен быть описан оператором `Intent`.

Пример 1 (функция)

В функции: вычислить среднее арифметическое значение N последних элементов одномерного вещественного массива.

В главной программе: вызвав функцию, вычислить среднее значение 5 последних элементов вещественного массива из 10 элементов.

***Программа* 1**

```
Real Function Primer(Array,N)
 Integer, Intent(In) :: N
  Real, Intent(In), Dimension(1:N):: Array
   Integer Len  ! локальная переменная
    Len = Size(Array)
        ...     ! операторы, реализующие алгоритм функции
    If(N>0)Primer=sum(Array)/N ! результирующее значение
End Function Primer

Program Main ! Главная программа – Interface и вызов функции
 Implicit None
  Interface
   Real Function Primer(Array, N)
    Integer,Intent(In) :: N
    Real,Intent(In),Dimension(:) :: Array
   End Function Primer
  End Interface
   Real Result
    Real Dimension(1:10) :: Mas
         ...     ! ввод и вывод исходных данных
Result=Primer(Mas(6:10),5) ! вызов: Mas,5-фактические аргументы
         ...     ! вывод результатов
End Program Main
```

Результаты работы программы 1 в файле Out.txt

```
 Исходный массив
    2.75 -12.88   0.12  -0.09  99.55
   33.77   1.11  -4.66   3.00   2.99
 Среднее арифметическое   5 последних элементов
      5.923
```

Пример 2 (подпрограмма)

В подпрограмме: найти номер и значение максимального по модулю элемента одномерного массива.

В главной программе: Используя подпрограмму, найти номера и значения максимальных по модулю элементов для нескольких массивов разной длины.

Программа 2

Далее приведена подпрограмма и главная программа, в составе которой показаны интерфейс и вызовы подпрограммы.

```
  Subroutine MaxAbs(Array, eLem, Num)
    Implicit none
    Real,intent(in),dimension(:)::Array ! входной массив
    Real,intent(out)::eLem  ! выходной: max по модулю элемент
    Integer,intent(out)::Num ! выходной: номер max по модулю
     Integer Len   ! локальная переменная
       Len = Size(Array)
          ...     ! операторы, реализующие алгоритм подпрограммы
       eLem = ... ! результирующее значение
       Num = ... !  результирующее значение
  End Subroutine MaxAbs

  Program Sub1 ! ФИО Лаб 6  Группа. Вар.
   Implicit none
   Interface    ! шаблон вызываемой подпрограммы
    Subroutine MaxAbs(Array, eLem, Num)
     Implicit none
     Real,intent(in),dimension(:)::Array ! входной массив
     Real,intent(out)::eLem  ! выходной: max по модулю элемент
     Integer,intent(out)::Num ! выходной: номер max по модулю
    End Subroutine MaxAbs
   End  Interface
    Real,dimension(1:12)::B
     Real,dimension(1:10)::C
      Real MaxB, MaxC   ! max по модулю элементы для B,C
       Integer NumB, NumC ! и их порядковые номера в массивах
        ...     ! ввод и вывод в результирующий файл массива B
     Call MaxAbs(B, MaxB, NumB)
        ...     ! вывод в результирующий файл NumB, MaxB
        ...     ! ввод и вывод в результирующий файл массива C
     Call MaxAbs(C, MaxC, NumC)
        ...     ! вывод в результирующий файл NumC, MaxC
  End Program Sub1
```

Результаты работы программы Sub1

```
 Массив B
     1.0    -4.0     6.0    20.0    45.0   -71.0
     4.0   -17.0      .0      .0      .0    11.0
 max по модулю элемент     -71.0 его номер  6 в массиве B

Массив C
  -122.0      .0    45.0    71.0     4.0   -17.0
      .0      .0      .0    11.0
 max по модулю элемент   -122.0 его номер  1 в массиве C
```

Контрольные вопросы

1. Назовите четыре вида *программных единиц* в Фортране.
2. Что такое процедура? Когда целесообразно использование процедур? Сколько разновидностей процедур в Фортане?
3. Каковы основные атрибуты функции?
4. Как результат функции передается в вызывающую программу?
5. Что в программировании называют *подпрограммой*? Сколько разновидностей подпрограмм в Фортане?
6. Зачем подпрограмме имя? Есть ли значение, связанное с именем подпрограммы? Как передаются результаты подпрограммы в вызывающую программу?
7. Что такое *формальные и фактические* аргументы? Правила согласования формальных и фактических аргументов, поясните примерами.
8. Какими по назначению могут быть формальные аргументы процедуры? Приведите примеры описания назначения формальных аргументов.
9. Где и для чего пишут интерфейсы?
10. Чем процедура отличается от функции? Сравните на примере, оформив как функцию и как подпрограмму вычисление количества повторений цикла.

Примеры задач контрольного задания

1. *В функции:* вычислить среднее геометрическое положительных элементов одномерного массива. Если в массиве нет положительных элементов, результат приравнять нулю. *В главной программе* применить функцию к трем массивам разной длины. Вывести результаты с именами массивов. Найти и вывести минимум среди полученных значений, не равных нулю.

   Блок-схема функции обязательна.

2. *В подпрограмме*: вычислить сумму элементов, расположенных ниже, выше и на главной диагонали квадратной вещественной матрицы. *В главной программе*: применить подпрограмму к матрице размером 8×8. Вывести три значащих цифры результатов.

   Блок-схема подпрограммы обязательна.

3. В вещественном массиве обнулить элементы, синус которых превышает 0.5 (оператор *where*). Как получить тот же результат без использования оператора *where*?

4. Исправьте ошибку

```
Integer i
Real, dimension(7,4):: A
```

```
write(*,10) 'Массив',(A(i,:), i=1,4)
10 Format(a/(4F10.2))
```

Таблица 14.

Варианты индивидуального задания «Процедуры»

Вариант 1

В подпрограмме 1: транспонировать матрицу. Логический выходной параметр равен *.true*., если транспонирование выполнимо (иначе *.false*.)
В подпрограмме 2: перемножить две матрицы. Логический выходной параметр равен *.true*., если умножение матриц выполнимо (иначе *.false*.)

В главной программе: используя подпрограммы, на примере двух матриц A и B показать, что $(AB)^{\mathrm{T}} = A^{\mathrm{T}}B^{\mathrm{T}}$. Размеры матриц (не квадратных) выберите сами.
Работу подпрограммы 1 проверить, вызвав функцию *Transpose,* подпрограммы 2 – функцию *Matmul.*
После обращения к каждой подпрограмме выводить сформированную матрицу или аварийное сообщение о невозможности получения результата. После аварийного сообщения завершить программу.

Вариант 2

В функции: вычислить произведение ненулевых элементов одномерного массива, расположенных между максимальным и минимальным элементами. Если между максимумом и минимумом нет ненулевых элементов, результат приравнять нулю. Для проверки вывести координаты максимума и минимума в массиве.

В главной программе: применить функцию к трем массивам разной длины, вывести результаты с именами массивов. Найти и вывести максимум среди полученных значений.

Вариант 3

В подпрограмме: координаты седловых элементов матрицы записать в двумерный массив размером $2\times N$, в котором каждому элементу соответствует столбец, а N – количество седловых элементов.
Седловым считать элемент матрицы, значение которого строго меньше его «соседей» по горизонтали и строго больше «соседей» по вертикали. Учесть, что количество «соседей» элемента колеблется от 2 до 4 в зависимости от его положения в матрице.

В главной программе: воспользоваться подпрограммой для двух матриц, для каждой вывести таблицу с координатами седловых элементов и их значениями.

Вариант 4

В функции: методом трапеций найти $\int_a^b f(x)dx$ по таблице значений $f(x)$

В главной программе: составить две таблицы значений $f(x)=2x^4-x^2+1$ в интервале $[a, b] = [-1, 1.4]$ с шагом $n_1 = 0{,}2$ и $n_2 = 0{,}04$. Применив функцию, вычислить величину интеграла. Сравнить полученные значения с интегралом, вычисленным по первообразной функции. В правильности вычислений убедиться средствами пакета *Agrapher*.

Вариант 5

В подпрограмме: создать матрицу, в которой каждый отрицательный элемент исходной матрицы заменен средним арифметическим его «соседей» по горизонтали и по вертикали при условии, что все «соседи» неотрицательны. Учесть, что количество «соседей» элемента колеблется от 2 до 4 в зависимости от его положения в матрице.
Как выходной аргумент создать двумерный массив, в который записать координаты замененных элементов матрицы.

В главной программе: воспользоваться подпрограммой для двух матриц разных размеров, для каждой вывести измененную матрицу и координаты замененных элементов.

Вариант 6

В функции: реализовать циклический сдвиг матрицы на *n* элементов вправо, влево, вверх или вниз в зависимости от заданного режима. Значение *n* может превышать протяженность матрицы по любому измерению.

В главной программе: ввести матрицу и многократно сдвигать ее, меняя направление и величину сдвига в диалоге и используя функцию.
После каждого обращения к подпрограмме выводить направление, величину сдвига и полученную в результате матрицу.
Работу программы прекратить после ввода слова, отличного от слов *Up, Down, Left, Right (*например*, Stop)*.

Вариант 7

В подпрограмме: по значениям трех коэффициентов квадратного уравнения найти его корни (действительные или комплексные).
Логический выходной параметр равен *.true.*, если корни действительные (иначе *.false.*).

В главной программе: используя подпрограмму, найти корни 10 уравнений, коэффициенты которых расположены в двумерном массиве размером 3×10 (каждому уравнению соответствует столбец массива). Результаты оформить в виде таблицы.

Вариант 8

В функции: вычислить определенный интеграл как площадь под кривой по формуле трапеций $S = \sum_{i=1}^{n-1} \Delta x \cdot \frac{(y_{i+1} + y_i)}{2}$, где n – количество значений функции Y в одномерном массиве, а Δx – постоянный шаг изменения аргумента.

В главной программе: сформировать массивы значений функции
$y = exp(-x/5)$ в интервале [-2, 2]:
a) с шагом $\Delta x_1 = 0.1$ b) с шагом $\Delta x_2 = 0.01$.
Вычислить определенный интеграл для каждого из этих массивов, сравнить вычисленные значения с определенным интегралом, вычисленным по первообразной функции.

Вариант 9

В подпрограмме: умножить матрицу на вектор. Логический выходной параметр равен *.true.*, если умножение матрицы на вектор выполнимо (иначе *.false.*)

В главной программе: на примере двух матриц A и B и вектора V показать, что $(A+B)V = AV + BV$. Размеры матриц и вектора выберите сами. Работу подпрограммы проверить, вызвав функцию *Matmul.*
После каждого обращения к подпрограмме выводить сформированный вектор или аварийное сообщение о невозможности получения результата. После аварийного сообщения завершить программу.

Вариант 10

В функции: с точностью ε вычислить приближенное значение функции e^x при заданных значениях x и ε. Вычисления проводить суммированием членов ряда Тейлора.

В главной программе: значение ε ввести в диалоге и вывести в результирующий файл. Используя функцию, составить и вывести таблицу значений e^x, изменяя значение аргумента от x_n до x_k с шагом Δx (10 значений). Для проверки дополнить таблицу значениями e^x, вычисленными по стандартной программе.

Вариант 11

В подпрограмме: вычислить математическое ожидание и дисперсию случайной величины, значения которой находятся в одномерном массиве. *Примечание*: для равномерно распределенной случайной величины математическое ожидание и дисперсия вычисляются по формулам

$$M(x) = \frac{\sum_{i=1}^{n} x_i}{n} \qquad D(x) = \frac{\sum_{i=1}^{n} [x_i - M(x)]^2}{n}$$

В главной программе: с помощью генератора псевдослучайных чисел *RANDOM_NUMBER* создать два одинаковых по длине одномерных массива *X* и *Y*, значения которых находятся в интервале [0; 10].
Используя подпрограмму, на примере этих массивов показать, что *D(X+Y) = D(X) +D(Y)*. После каждого обращения к подпрограмме выводить результаты с пояснениями.

Вариант 12

В функции: с точностью ε вычислить приближенное значение функции $\cos x$ при заданных значениях x и ε. Вычисления проводить суммированием членов ряда Тейлора (использовать рекуррентную формулу).

В главной программе: Значение ε ввести в диалоге и вывести в результирующий файл. С помощью функции составить вывести таблицу значений $\cos x$, изменяя значение аргумента от x_n до x_k с шагом Δx (10 значений). Для проверки дополнить таблицу значениями $\cos x$, вычисленными по стандартной программе.

Вариант 13

В подпрограмме 1: вычислить *скалярное (внутреннее)* произведение двух векторов. Логический выходной параметр равен *.true*., если умножение выполнимо (иначе . *false*.)
В подпрограмме 2: вычислить матрицу – *внешнее* произведение двух векторов. Логический выходной параметр равен.*true*., если умножение выполнимо (иначе . *false*.)

В главной программе: применить подпрограммы к двум векторам.
Для проверки подпрограммы 1 вызвать функцию *Dot_product,* подпрограммы 2 – функцию *Matmul.*
После обращения к каждой подпрограмме выводить результат умножения или аварийное сообщение о невозможности его получения. После аварийного сообщения завершить программу.

Вариант 14

В функции: методом прямоугольников вычислить $\int_a^b f(x)dx$ по таблице значений *f(x)*.

В главной программе: составить две таблицы значений $f(x)=2x^3-3x+4$ в интервале [*a, b*] = [–1.5, 1.5] с шагом n_1 = 0,2 и n_2 = 0,04. Применив функцию, вычислить величину интеграла. Сравнить полученные значения с интегралом, вычисленным по первообразной функции. В правильности вычислений убедиться средствами пакета *Agrapher*.

Вариант 15

В подпрограмме 1: сортировать одномерный массив по убыванию методом последовательного нахождения минимального элемента массива и перестановки его в начало массива.
В подпрограмме 2: сортировать одномерный массив по убыванию методом «пузырька».
В каждой подпрограмме: подсчитать количество перестановок элементов при сортировке.

В главной программе: с помощью генератора псевдослучайных чисел *RANDOM_NUMBER* создать одномерный массив, значения которого находятся в интервале [0,10]. Обратившись к подпрограммам, сортировать этот массив по убыванию. Вывести массив до и после сортировки. Сравнить полученные массивы и количества перестановок.

Вариант 16

В функции 1: любым способом сортировать одномерный массив по убыванию.
В функции 2: объединить два отсортированных по убыванию массива, не нарушив сортировки.

В главной программе: с помощью генератора псевдослучайных чисел *RANDOM_NUMBER* создать два одномерных массива разной длины, значения которых находятся в интервале [0; 100]. Обратившись к функции 1, отсортировать эти массивы по убыванию. Вывести массивы до и после сортировки. Используя функцию 2, объединить эти массивы. Вывести объединенный массив.

Вариант 17

В подпрограмме: перемножить две матрицы. Логический выходной параметр равен *.true*., если умножение матриц выполнимо (иначе . *false*.).

В главной программе: на примере трех матриц *A, B* и *C* показать, что $(A+B)C = AC + BC$. Размеры матриц (не квадратных) выберите сами.
Работу подпрограммы проверить, вызвав функцию *Matmul.*
После обращения к каждой подпрограмме выводить сформированную матрицу или аварийное сообщение о невозможности получения результата. После аварийного сообщения завершить программу.

Вариант 18

В функции: методом парабол (Симпсона) вычислить $\int_a^b f(x)dx$ по таблице значений $f(x)$.

В главной программе: составить две таблицы значений $f(x) = \frac{1}{\sin^2 x}$ в интервале $[a, b] = [1.5, 2.5]$ с шагом $n_1 = 0{,}1$ и $n_2 = 0{,}02$. Применив функцию, вычислить величину интеграла. Сравнить полученные значения с интегралом, вычисленным по первообразной функции. В правильности вычислений убедиться средствами пакета *Agrapher*.

Вариант 19

В подпрограмме: вычислить длины сторон, периметр и площадь многоугольника по координатам его вершин. Координаты вершин многоугольника расположены в двумерном массиве, в котором каждой вершине соответствует столбец (первая строка содержит координату *x*, вторая – координату *y*).

В главной программе: последовательно читая из файла координаты вершин треугольника, четырехугольника, пятиугольника и применяя к ним подпрограмму, составить таблицу (длины сторон, периметр и площадь).
В отчете привести графическую интерпретацию.

Вариант 20

В функции: получить логическое значение (*.true.* или *.false.*), определяющее, расположен ли максимальный отрицательный элемент одномерного массива ранее минимального положительного элемента. Для проверки вывести координаты найденных элементов массива.

В главной программе: применить функцию к трем массивам разной длины. В соответствии с результатами вывести сообщения, указав имена массивов.

Вариант 21

В подпрограмме: методом *касательных (Ньютона)* вычислить действительный корень уравнения $y = (x - 0.5)^2 + 1.3e^{-0.15x} + 1.1e^{-1.5x}$ в интервале [*a, b*] с абсолютной погрешностью ε. Логический выходной параметр равен *.true*., если корень существует (иначе . *false*.).

В пакете *Agrapher* построить график функции и определить интервалы нахождения корней.

В главной программе: дважды применить подпрограмму к каждому из найденных интервалов, приняв абсолютную погрешность равной 10^{-3} и 10^{-5}. Проверить решение, подставив корни в уравнение.

В отчете представить график функции.

Вариант 22

В функции: сформировать матрицу *B(m,n)* в соответствии с исходной матрицей *A(m,n)* по правилу $b_{ij} = \dfrac{\left(\sum_{k=1}^{n} a_{ik}\right) - a_{ij}}{\left(\sum_{k=1}^{m} a_{kj}\right) - a_{ij}}$.

В главной программе: прочесть из файла две матрицы разных размеров и применить к ним функцию. Вывести исходные и результирующие матрицы.

Вариант 23

В подпрограмме: на интервале [*a, b*] разбить кривую линию, заданную уравнением, на *n* участков и вычислить ее длину на этом интервале.

В главной программе: используя подпрограмму, вычислить длину линии, заданной уравнением $y = e^{-0.3x} - e^{-1.5x} + 3{,}8$ на интервале [–1, 5] при $n_1 = 20$ и при $n_2 = 40$. Сравнить результаты. В правильности вычислений убедиться средствами пакета *Agrapher*, вычислив длину линии на заданном интервале по формуле $D = \int_a^b \sqrt{1 + [f'(x)]^2}\,dx$.

Вариант 24

В функции: вычислить сумму элементов диагонали, параллельной главной диагонали матрицы. Диагональ задаётся координатами ее верхнего элемента.

В главной программе: применить функцию ко всем диагоналям, параллельным главной диагонали матрицы *A(6,6),* начиная с левого нижнего угла. Составить и вывести одномерный массив из полученных значений.

Вариант 25

В подпрограмме 1: определить количество и составить массив простых чисел в интервале [1; *N*], используя определение простых чисел.
В подпрограмме 2: определить количество и составить массив простых чисел в интервале [1; *N*], используя алгоритм «Решето Эратосфена».

В главной программе: последовательно применив обе подпрограммы, найти простые числа среди первых 200 чисел натурального ряда. Сравнить результаты применения подпрограмм.

Вариант 26

В функции: подсчитать количество отрицательных элементов внутри прямоугольного фрагмента матрицы *M,* диагонально противоположными вершинами которого являются элементы с индексами (i_1,j_1) и (i_2,j_2).

В главной программе: В каждой из четырех матриц разных размеров найти максимальный и минимальный элементы и применить функцию к прямоугольнику, вершинами которого они являются. Вывести таблицу с координатами найденных вершин и количеством отрицательных элементов в прямоугольнике для каждой матрицы.
В исходных данных предусмотреть различное взаимное расположение максимума и минимума для этих матриц.

Вариант 27

В подпрограмме: методом *половинного деления* вычислить действительный корень уравнения $y = (x - 0.6)^3 - (x + 4)^2 + x + 26.8$ в интервале [*a, b*] с абсолютной погрешностью ε. Логический выходной параметр равен *.true*., если корень существует (иначе . *false*.).

В пакете *Agrapher* построить график функции и определить интервалы нахождения корней.
В главной программе: дважды применить подпрограмму к каждому из найденных интервалов, приняв абсолютную погрешность равной 10^{-3} и 10^{-5}. Проверить решение, подставив корни в уравнение.
В отчете представить график функции.

Вариант 28

В функции: вычислить произведение трех наименьших по модулю ненулевых элементов целочисленной матрицы. Если в матрице меньше трех ненулевых элементов, результат приравнять нулю.

В главной программе: применить функцию к трем целочисленным матрицам, разным по размеру. Вывести соответствующие сообщения с именами массивов.

Вариант 29

В подпрограмме: Найти координаты точек, в которых прямая, заданная уравнением $kx+b$, пересекает стороны треугольника; координаты вершин треугольника расположены в двумерном массиве размером 2×3 (каждой вершине соответствует столбец массива, первая строка содержит координату *x*, вторая – координату *y*).
Логический выходной параметр равен *.true*., если прямая пересекает треугольник (иначе *.false*.)

В главной программе: ввести коэффициенты уравнений пяти прямых и координаты вершин треугольника.
Применив подпрограмму к каждой из прямых, составить таблицу (коэффициенты уравнений, координаты точек пересечения прямых со сторонами треугольника).
В отчете привести графическую интерпретацию.

Вариант 30

В функции: составить двумерный массив локальных минимумов матрицы (каждой вершине соответствует столбец массива, первая строка содержит координату *x*, вторая – координату *y*).. *Локальным минимумом* считать элемент матрицы, значение которого строго меньше его «соседей» по горизонтали и по вертикали. Учесть, что количество «соседей» элемента колеблется от 2 до 4 в зависимости от его положения в матрице.

В главной программе: применить функцию к трем матрицам разных размеров. Вывести полученные массивы.

2.7. Механизмы присоединения данных

Задание

1. Реализовать два проекта, которые содержат:
   a) ввод данных из файла *In.txt* на диске и форматный вывод их в результирующий файл *Out.txt*;
   b) вызовы процедур с различными данными для отладки;
   c) если возможно, проверку результатов работы процедур обращением к стандартным подпрограммам или функциям;
   d) форматный вывод в файл *Out.txt* результатов работы каждой процедуры с пояснениями выводимых величин (использовать формулировки из индивидуального задания).
2. В проекте *Container* – механизм присоединения объектов носителя.
3. В проекте *ProModule* – механизм присоединения объектов модуля.

Содержание отчета.

1. Название работы и номер варианта индивидуального задания.
2. Фамилия, имя, отчество и номер группы студента.

3. Текст варианта индивидуального задания.
4. Постановка задачи, методы вычислений.
5. Описание механизмов обмена данными.
6. Графическая интерпретация (если требуется), выполненная вручную или пакетом *Agrapher*.
7. Блок-схемы главной программы и подпрограмм.
8. Распечатки текстов всех программных единиц.
9. Распечатки файлов результатов *Out.txt*.

При подготовке к защите ответить на контрольные вопросы.

Комментарии к заданию

1. Приступая к выполнению задания, прочтите *раздел 5*.
2. В проекте *Container* расчетная процедура – это внутренняя процедура, присоединяющая объекты носителя, которым является внешняя программа.
3. В проекте *ProModule* расчетная процедура – это модульная процедура, присоединяющая объекты используемого модуля.
4. Расчетные процедуры в виде внутренней и модульной процедуры не являются самостоятельными программными единицами.
5. Вызов внутренней или модульной процедуры не требуют `Interface`.

Пример

В подпрограмме: ввести одномерный массив из файла *in.txt*; найти и вывести в файл *out.txt* номер и значение максимального по модулю элемента массива.

В главной программе: применить подпрограмму к нескольким массивам, предварительно вводя длину каждого массива из файла *in.txt*.

Применить механизмы присоединения объектов носителя и модуля.

Проект Container

Главная программа *Container* – единственная самостоятельная программная единица. В программу *Container* вложена внутренняя подпрограмма `Subroutine` *OneArray*. Программа *Container* является носителем данных для подпрограммы *OneArray*. В ней лишь одна глобальная переменная *dArr*, указывающая длину массива. Эту переменную заимствует подпрограмма *OneArray*.

Переменные *ARR*, *NumARR* локальны в подпрограмме *OneArray*. Динамический массив *ARR* локализован в подпрограмме *OneArray*. Его переменная длина *dARR* определена до входа в подпрограмму. Команды `allocate` и `deallocate` не нужны – динамический массив размещается автоматически при входе в подпрограмму и освобождается при выходе.

Программа

```
Program Container     ! внешняя программа
 Implicit none
  Integer dARR              ! длина автоматического массива ARR
 Open(1,File='In.txt') ! файл исходных данных
 Open(2,File='Out.txt')! результирующий файл
 Read(1,*)dARR; Call OneArray('B') ! ввод dARR - длины массива 'B'
 Read(1,*)dARR; Call OneArray('C') ! ввод dARR - длины массива 'C'
      contains
  Subroutine OneArray(ArrName)    ! внутренняя подпрограмма
   Implicit none
   character,intent(in)::ArrName !однобуквенное "имя массива"
    Real,dimension(1:dARR)::ARR ! локальный динамический массив
     Integer::NumARR      ! локальный номер max по модулю
   Read(1,*)ARR; Write(2,2)ArrName,ARR ! ввод/вывод массива ARR
   NumARR=sum(maxloc(Abs(ARR)))
   Write(2,3) ARR(NumARR), NumARR, ArrName
   2 format(/' Массив ', a/ ( 6F8.1) )
   3 format(' max по модулю элемент ', F8.1, &
    ' его номер', i3,' в массиве ', a)
  End Subroutine OneArray  ! конец внутренней подпрограммы
End Program Container
```

Результаты работы программы Container

```
Массив B
    1.0    -4.0     6.0    20.0    45.0   -71.0
    4.0   -17.0      .0      .0      .0    11.0
max по модулю элемент     -71.0 его номер  6 в массиве B

Массив C
 -122.0      .0    45.0    71.0     4.0   -17.0
     .0      .0      .0    11.0
max по модулю элемент   -122.0 его номер  1 в массиве C
```

Проект ProModule – использование общедоступных объектов модуля.

В этом проекте две самостоятельные программные единицы – модуль *Mo* и головная программа *ForModule*, которые:

1. могут быть отдельными текстами и компилироваться порознь, но модуль должен компилироваться раньше главной программы;
2. могут быть единым текстом, но модуль должен располагаться перед главной программой.

Модуль *Mo* содержит модульную процедуру *ProArray*, которая размещена внутри модуля. Процедура *ProArray – без аргументов*. Интерфейс модульной процедуры считается явным и его описывать не надо.

Все объекты, кроме массива *ARR*, размещены в модуле. Главной программе *forModule*, использующей модуль, доступны объекты *ArrN, dARR (public)* Локальная *(private)* переменная модуля *NumARR* доступна в модульной процедуре *ProArray* и недоступна в программе *forModule.*

Массив *ARR* локализован в процедуре *ProArray*, аналогичный динамический массив используется в предыдущем примере.

Программа

```
Module Mo   ! ФИО Лаб 7  № группы, №  варианта
 Implicit none
 Integer,public::ArrN,dARR !порядковый № массива, длина массива
 Integer,private::NumARR !локальная: № max по модулю элемента,
  Contains ! модульная процедура
  Subroutine ProArray
   Implicit none
   Real,dimension(1:dARR)::ARR ! динамический автоматич. массив
  Read(1,*)ARR; Write(2,2)ArrN,ARR ! ввод и вывод массива ARR
   2 format(/' Массив №', i5/ ( 6F8.1) )
  NumARR=sum(maxloc(Abs(ARR)))
  Write(2,3) ARR(NumARR), NumARR, ArrN
          3 format(' max по модулю элемент ',F8.1, &
               ' его номер', i3, ' в массиве №',i5)
  End Subroutine proArray
end Module Mo

Program forModule
 use Mo ! использует Mo,  доступны:  ArrN, dARR , процедура  proArray
 Open(1,file='in.txt'); Open(2,file='Out.txt') !файлы:данные,результат
  do  ArrN=1,10000        ! читаем  до 10000 массивов
     Read(1,*, end=10) dARR;  Call ProArray
  enddo
 10 Write(2,*)   ' Все данные прочитаны'
End Program forModule
```

Результаты работы программы forModule

```
Массив №    1
     1.0    -4.0     6.0    20.0    45.0    -71.0
     4.0   -17.0      .0      .0      .0     11.0
 max по модулю элемент     -71.0 его номер  6 в массиве №    1

Массив №    2
    22.0      .0    45.0    71.0     4.0   -17.0
      .0      .0      .0    11.0
 max по модулю элемент      71.0 его номер  4 в массиве №    2
 Все данные прочитаны
```

Контрольные вопросы к защите работы

1. Назовите четыре вида *программных единиц* в Фортране.

2. Назовите три вида *программ*.
3. Назовите два вида процедур.
4. Что называется *подпрограммой*?
5. Что такое *внешняя и внутренняя программа*?
6. Какие виды программных единиц могут быть только внешними? Какие виды программ могут быть внутренними?
7. Могут ли объекты внешней программы быть невидимыми во внутренней программе?
8. Чем модуль отличается от программ?
9. Каков механизм присоединения данных носителя? Примеры.
10. Опишите механизм присоединения объектов модуля, примеры.
11. Какие объекты модуля не видимы в программе, использующей его?
12. Когда интерфейс обязателен, когда желателен, когда не нужен?
13. Нужен ли оператор *interface* для модульной программы?
14. Когда возникает коллизия имен при использовании модуля?

Пример контрольного задания

Вычисление количества повторений цикла оформить как:

1. *внешнюю* функцию *function*;
2. *внешнюю* подпрограмму *subroutine*;
3. *внутреннюю* процедуру – функцию или подпрограмму;
4. *модульную* процедуру – функцию или подпрограмму.

Варианты индивидуальных заданий

Варианты индивидуальных заданий см. в 2.6.

3. Элементы языка программирования Фортран

3.1. Текст программы на Фортране

Программу на Фортране оформляют в виде простого (*plain*) текста:

- символ «!» открывает комментарий – до конца строки;
- упрощают чтение текста программы пустые строки и пробелы (вне имен и ключевых слов);
- каждая программа, каждый простой оператор, каждый составной оператор, каждый блок пишут с новой строки, при необходимости делая перенос на следующую строку (символ переноса «&»);
- компиляторы поддерживают две формы записи текстов программ: **.f90*, **.f03* – современная *свободная форма* для новых текстов, **.for* -устаревшая *фиксированная форма* (преемственность с Ф77).

Таблица 15.

Пример оформления текста программы на Фортране

1	**Program** FreeForm ! *тема: Площадь круга*	**Program -** Начало программы
2	! *ФИО №группы №варианта №работы*	строка-комментарий
3	**Real** R; **integer** i ! комментарий	Два простых оператора в строке
4	**Do** i=1,10 ! *цикл по 10 радиусам*	**Do** - Начало конструкции
5	**Write**(*,*)' R=?'; **Read**(*,*) R	начало **блока**
6	**Write**(*,*)'R =', R, & ! *& - перенос*	с переносом по &
7	' S =', 3.14* R**2	строка-продолжение
8	**End do**	**End+Do** = конец конструкции **Do**
9	**End Program** FreeForm	**End+Program** = конец программы

Привязка операторов и блоков к строкам традиционно осуществляется одним из двух символов *конца строки* – клавиша <Enter> и «;» :

– используя символ «;», в одной видимой строке (строка 3, строка 5) компонуют *простые* операторы;

– <*Enter*> предшествует программной единице (строки 1-9), ее телу (строки 2-8), конструкции (строки 4-8) и блоку do (строки 5-7);

– специальных символов окончания оператора и блока в Фортране нет.

В языках Си и Паскаль все делается не так – операторы *не привязаны* к строкам, а каждый оператор или блок должен заканчиваться спецсимволом «;», что затрудняет чтение сложной программы. Те, кто приходят из Си и Паскаль, испытывают облегчение и быстро привыкают к оформлению текста на Фортране.

Программу из Табл.15 можно заметно упростить, если описать R не как скаляр Real R, а как вектор Real,dimension(1:10) **::** R. В этом случае совсем не понадобится цикл по элементам массива R, так как Фортран умеет то, чего нет в других языках – выполнять действия с массивами:

```
Program R10 ! тема: Площади 10 кругов
 Real,dimension(1:10):: R
 Write(*,*)' R=?';    Read(*,*) R ! ввести 10 радиусов
 Write(*,*)' R =',R; Write(*,*)' S =',3.14*R**2
End Program R10
```

3.2. Синтаксис языка Фортран, выражения

К простейшим элементам синтаксиса языка относят: *константы, имена, знаки и имена операций, метки, ключевые слова*.

Имена и ключевые слова

Имя дают константе, переменной, операции, массиву, производному типу данных, классу, конструкции, процедуре, модулю, интерфейсу. Действует следующее соглашение об именах:

- имена состоят из латинских букв, цифр 0÷9, символов «_», «$»;
- первым символом имени может быть только буква или символ «$»;
- строчные и прописные буквы не различаются;
- пробелы и знаки препинания, кириллица, греческие буквы, знаки операций и другие специальные символы в именах недопустимы;
- длина имени не более 31 символа.

Объекты программ, в отличие от однобуквенных математических обозначений (*s=v*t*), принято именовать со смыслом: *Space=Velocity*Time*.

Ключевые слова (основные – смотри раздел 8.5) – это *имена,* используемые в Фортран для операторов, атрибутов, некоторых операций, конструкций, встроенных функций. Правила те же, кроме:

- `.true.` и другие ключевые слова в логике обрамляются точками;
- целое в диапазоне 1:99999 используют как метку (для `format`).

В Фортране соглашение о ключевых словах *гибкое:* нет резервирования, как в Си. Ключевые слова могут использоваться в качестве обычных имен, лишь бы не возникало двусмысленности. Программисту не обязательно помнить все ключевые слова – ему поможет компилятор.

Пример *двусмысленности* по ключевому слову *max*

```
Real :: a=2, b=3, c, max ! max – простая переменная
max=a; C = max(a,b)  ! здесь max()- встроенная функция
```

Ошибка: inconsistent usage of MAX - неподходящее использование слова **max**.

Константы и переменные встроенных типов

Константу нельзя изменить в процессе исполнения программы, она может быть записана явно как 3.14 или иметь имя, например *Pi*. Переменная – это именованный объект, значение которого можно изменить.

Имеется пять встроенных типов:

- числовые: целые *integer*, вещественные *real*, комплексные *complex*;
- логические *logical*;
- строковые (символьные) *character*.

Таблица 16. Примеры оформления констант разных типов

Тип	Примеры явных констант	Разновидности и диапазоны порядков	Пояснения
integer	–2147	**integer*1 *2 *4** $\pm 10^{10}$	Точное значение
Real	-1.76 1e7 1.2e7 1.3D+123 1.2Q-1234	**real* 4** $10^{\pm 38}$ 7 цифр **real* 8** $10^{\pm 308}$ 17 цифр **real*16** $10^{\pm 4932}$ 33 цифры	Неточное представление, неточное вычисление
Complex	(-1.76 , 1)	Пара вещественных чисел	Неточно
Logical	.true. .false.	**Logical***1 *2 *4	1 бит
character	*'x='* "O'K"	**character***1 до *32767 символов	Кодировка символов ANSI

Каждому типу соответствует своя область допустимых значений, особый способ хранения в памяти ПК, объём памяти, набор операций ПК.

Атрибут *именованной* константы *parameter*, тип и значение входят в ее описание: `real,parameter :: Pi =3.141593`.

Выражения

Выражение – осмысленная математическая формула, записанная по правилам языка программирования. В отличие от математики формула записывается в виде строк текста; объекты именуются, значки функций заменены именами функций, которые указаны в описании языка и доступны в помощи. В отличие от других языков программирования, выражение может включать не только переменные, но и массивы. В выражение могут входить:

- константа, переменная, элемент массива;
- массив, секция массива;
- вызов функции с аргументами в скобках, например, *cos(x-0.2) ;*
- операция с одним операндом, например, *–x* ;
- операция с несколькими операндами, например, *x+y+z ;*
- парные круглые скобки ( ), квадратных и фигурных скобок нет.

В зависимости от типов объектов, входящих в выражения, различают:

- *арифметические* (*числовые*) выражения с *числовым* результатом – выполняются операции над числами целого, вещественного и комплексного типа, выражения могут быть смешанными;
- *логические* выражения с *логическим* результатом: *логические* операции выполняются над логическими величинами, операции отношения – над числами или строками;
- *строковые* выражения со *строковым* результатом – выполняются строковые функции и операция сцепления строк, обозначаемая *s//p*

Арифметическое выражение

Операнды *арифметического* (числового) выражения могут быть целого, вещественного или комплексного типа.

Таблица 17. В числовом выражении допускаются операции

Операция	Порядок	Пример	Примечания
Вычисление функции	1	**sin**(X)	одноместная (унарная) операция
Возведение в степень	2	X**2	несколько подряд – *справа налево*
Смена знака	3	–X	одноместная (унарная) операция
Деление, умножение	4	X/Y, X*Y	несколько подряд – *слева направо*
Сложение, вычитание	5	X+Y, X-Y	несколько подряд – *слева направо*

Знаки двуместных операций - возведение в степень, умножение, деление, сложение и вычитание записываются между операндами. В отличие от математики, знак операции умножения опускать нельзя. Численное зна-

чение результата зависит от порядка выполнения операций. Если надо, порядок действий изменяют скобки; разрешаются только круглые. В выражении без скобок операции выполняются в порядке старшинства.

В компьютере имеются два комплекта арифметических операций – для целых и для вещественных чисел. Действия над комплексными числами выполняются программно по правилам, известным из математики с помощью команд для вещественных чисел. Фортран автоматизирует выбор комплекта операций, руководствуясь типом операндов. Если операнды однотипны, то этого же типа будет операция и результат. Для операций с операндами разных типов результат всегда в наиболее широком классе из числа операндов. Множества чисел расширяются так:

– целые – *1 *2 *4 байта;

– шире – вещественные - *4 *8 *16 байтов;

– шире – комплексные - дважды по *4 *8 *16 байтов.

Обратите внимание, что выбор из двух разных операций деления производится по общему правилу:

- при делении вещественных чисел в результате – вещественное число, например, 2./3.=0.666667;
- при делении целых чисел в результате – целое число, равное целой части частного, например, 8/3=2 или 2/3=0;
- при делении разнотипных операндов в результате – вещественное число, например, 2./3=>2./3.=0.6666667 или 2/3.=>2./3.=0.666667 .

Операция возведения в степень a^n и a^x вычисляется программно:

- при *integer n* операция *a**n* выполняется как *n-кратное умножение* *a*a*a* ..*a*, поэтому *a**n*=0 при *n*<0 и *integer a*, например, 3**(-2)=1/3**2=1/9=0;
- при *real x* операция a**x выполняется как $a^x = e^{\ln a^x} = e^{x \ln a}$, следовательно, должно быть a > 0 ;
- в общем случае для *комплексных a,b* $a^b = (re^{\theta i})^b = (e^{\mathrm{Ln}(r)+\theta i})^b = e^{(\mathrm{Ln}(r)+\theta i)b}$. Здесь $a=r+\theta i$ представлено в экспоненциальной форме, Ln – комплексный логарифм.

Все переменные, используемые в выражении, должны быть определены к моменту его вычисления.

При вычислении по формулам следует анализировать ОДЗ, чтобы избегать числовых операций с неопределенным результатом:

- деление на нуль *x*/0;
- деление нуля на нуль 0/0, 0.0/0.0, 0/0.0, 0.0/0;
- возведение нулевого основания в нулевую или отрицательную степень 0**0 0**(-2);
- недопустимо возведение отрицательного основания в вещественную степень, как в (-8)**(1./3), но -(8**(1./3)) для нечетного 3.

Строковое выражение

Операнды строкового выражения должны быть строкового типа. В строковом выражении допускается только одна операция – *сцепление* строк (*конкатенация*). Сцепление обозначается «//», например s=s1//s2, символы в строке s нумеруются слева направо как 1,2,…`len`(s), `len` (s1//s2)=`len`(s1)+`len`(s2). Подстрока - несколько символов строки подряд. Пример: результат вычисления ab = ab`(14:26)//`ab`(1:13)` – строка, в которой меняются местами **первая** и вторая половины алфавита:

```
Character*26 :: ab='abcdefghijklmnopqrstuvwxyz'
```

Строка после: `ab='nopqrstuvwxyz`**`abcdefghijklm'`**

Обратите внимание на то, что оператор s1=s1//s2 с повторением s1 слева и справа от "=" имеет смысл только с подстроками. Нужно, чтобы суммарная длина не превышала `len`(s1). В Фортране строки имеют постоянную длину, а именно ту, что заявлена в её описании. В то же время оператор s1=`trim`(s1)//s2 вполне осмысленный, если в s1 хвостовых пробелов, не менее чем символов в s2. Функция `trim` (s1) обрезает хвостовые пробелы.

Логическое выражение

Логическое выражение имеет значение «истина» или «ложь», состоя из:

- логических констант и переменных;
- пяти логических операций с операндами логического типа;
- шести отношений, которые сравнивают либо два числовых выражения, либо два символьных выражения.

Результат вычисления отношения – либо **.**`true`**.** – «истина», либо **.**`false`**.**– «ложь».

Таблица 18. В Фортране шесть операций отношения

Отношения	**Фортран 90, сейчас пишут так**	**В стиле Фортран 77, устаревающее**
меньше чем *A*<*b*	a<b	a .**lt**. b
меньше или равно *A*≤*b*	a<=b	a .**le**. b
равно *A*=*b*	k==m	a .**eq**. b
не равно *A*≠*b*	k/=m	a .**ne**. b
больше или равно *A*≥*b*	a>=b	a .**ge**. b
больше чем *a*>*b*	a>b	a .**gt**. b

Отношения вычисляются после вычисления их операндов. Например, при *A*=2, *B*=0 для отношения *A*+3 >*B* порядок действий такой: сначала вычисляется *A*+3 и получается 5; затем 5 сравнивается с *B,* которое равно 0; результат вычисления выражения 5>0 – «истина».

Логические операции выполняются после вычисления отношений

Таблица 19.

В Фортране пять логических операций

Операция	Обозначение	Старшинство	Пример
Отрицание НЕ	**.not.**	1	**.not.***a*
Конъюнкция И	**.and.**	2	*a***.and.***b*
Дизъюнкция ИЛИ	**.or.**	3	*a***.or.***b*
Эквивалентность	**.eqv.**	4	*a***.eqv.***b*
Неэквивалентность	**.neqv.**	4	a**.neqv.***b*

Операция .not.*a* является, как и (–*x*), одноместной (унарной) и пишется перед операндом, остальные знаки логических операций записываются между операндами. Две логические операции могут следовать в выражении непосредственно друг за другом, только если второй операцией является операция отрицания. Например, допустимо *a*.and. .not.*b*.

Операция **.**not**.** – одноместная и дает результат «истина», если значение операнда «ложь», и результат «ложь», если значение операнда «истина».

Введем обозначения: *T* – «истина» и *F* – «ложь».

Таблица 20.

Таблицы истинности пяти логических операций выглядят как

.AND.	***F***	***T***
F	*F*	*F*
T	*F*	*T*

.OR.	***F***	***T***
F	*F*	*T*
T	*T*	*T*

.EQV.	***F***	***T***
F	*T*	*F*
T	*F*	*T*

.NEQV.	***F***	***T***
F	*F*	*T*
T	*T*	*F*

.NOT.	***F***	***T***
	T	*F*

Строки и столбцы таблиц истинности помечены значениям операндов, а на пересечении строки и столбца записан результат двухместной операции

Пример логического выражения: *a*>3 ***.and.*** *a*<5. Выражение истинно, когда оба отношения истинны, то есть $a \in (3,5)$.

Сложные выражения вычисляются в следующем порядке: числовые операции или строковые операции; операции отношения; логические операции.

3.3. Оператор присваивания

Оператор присваивания записывается в виде: *переменная = выражение*

В зависимости от типов переменной и выражения различают: числовое присваивание, символьное присваивание, логическое присваивание. Любое присваивание, в отличие от других языков программирования может быть векторным, матричным, конформным. Конформность относится как к опе-

рандам *выражения,* так и к *переменной* и к *выражению* между собой. Скаляр конформен массиву любого ранга.

Выполнение оператора состоит в вычислении выражения справа от «=» и присвоения результата переменной слева от «=». После выполнения присваивания прежнее значение переменной утрачивается. В отличие от математических формул знак «=» трактуется не как тождество, а как присваивание. Поскольку выражение вычисляется раньше, чем выполняется присваивание, одно и то же имя может быть указано в операторе одновременно слева и справа от знака «=». Например, *k=k+1* означает, что текущее значение переменной *k* увеличивает на 1. К моменту выполнения оператора переменные, входящие в выражение, должны быть определены.

Тип числового *выражения* назначается автоматически и если он не совпадет с типом *переменной*, то выполняется преобразование значения вычисленного выражения к типу переменной.

3.4. Ветвления *If* и циклы *Do*

Ветвления – конструкции *If*

Под *блоком* в Фортране понимают один или несколько выполняемых операторов внутри составного оператора. Передача управления извне внутрь блока запрещена. Конструкции `if` различаются по числу блоков 0, 1, 2, 3, …. Для реализации многоблочного разветвления имеется расширение `elseif` *(условие)* `then` , а также конструкция, называемая переключателем `Select case` по целочисленной или символьной переменной.

Таблица 21.

Двухблочный (*структурный*) условный оператор

Блок-схема	**Конструкция *If***

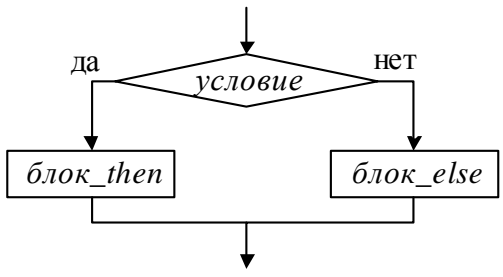
	**If** (*логическое_выражение*) **then**
	блок_then между *then* и *else*
	else
	блок_else между *else* и *endif*
	endif

Таблица 22.

Одноблочный условный оператор (нет *блока_else*)

Блок-схема	**Конструкция *If***

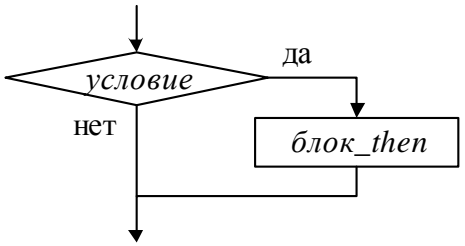
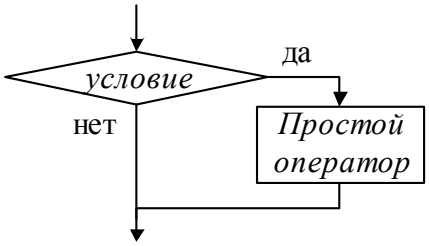
	**If** (*логическое_выражение*) **then**
	блок_then между *then* и *endif*
	endif

Таблица 23.

Безблочный, по-другому, логический условный оператор

Блок-схема	**Конструкция *If***
	**If** (*логическое_выражение*) *оператор* Пример **If** `(x<0) y=abs(x)` По виду блок-схема та же, но в прямоугольнике – единственный простой оператор

Циклы – конструкции *DO*

Циклом в программе называют группу (блок) многократно выполняемых операторов. Конструкция `do` обрамляет цикл

do …

…блок_do…

enddo

Имеется три разновидности циклов в Фортране:

- бесконечный цикл;
- итеративный цикл (цикл по условию);
- цикл по переменной (с заранее известным числом повторений).

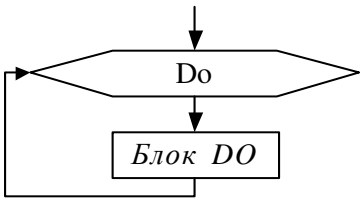
Вспомогательные операторы `Exit` и `Cycle`

Во всех трех разновидностях циклов (и только в циклах) можно использовать:

- `Exit` – досрочный выход из цикла на оператор, непосредственно *следующий* в тексте за `enddo` ближайшего охватывающего цикла либо указанного *именованного* цикла `Exit` *имя_цикла*;
- `Cycle` – переход на `enddo` *ближайшего охватывающего цикла* либо указанного *именованного* цикла `Cycle` *имя_цикла* .

Таблица 24.

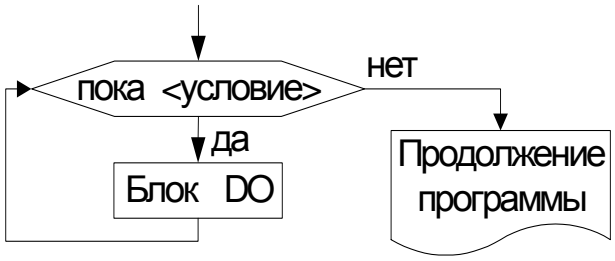
Бесконечный цикл

Блок-схема	Конструкция *Do*
	**do** *блок_do, в том числе* **if** *(на_выход)* **exit** **enddo** ! *только повтор* do

Чтобы выполнение цикла когда-либо закончилось, среди операторов *блока_DO* бесконечного цикла должен быть хотя бы один *exit* с условием выхода из цикла.

Таблица 25.

Цикл по условию (итеративный цикл)

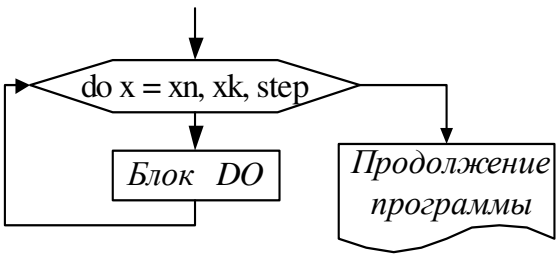
Блок-схема	Конструкция *Do*
	Пока *условие* выполняется, цикл продолжается **Do while**(*условие*) *блок_do* **enddo** ! *только повтор* do

Цикл по переменной

Переменная в операторе `Do` – предпочтительнее целая, но также допускается вещественная или вещественная двойной точности. Формула расчета числа повторений цикла $kp = \mathrm{Max}\left(0, \mathrm{Int}\left(\frac{xk - xn + step}{step}\right)\right)$.

Начало *xn*, конец *xk* и шаг *step* – константы, переменные или выражения одного из перечисленных типов. Функция `int( )` вычисляет целую часть указанного частного, отбрасывая дробную, без округления. Результат *max* **:** **max** (0, +) => + **max** (0, –) => 0 **max** (0, 0) => 0 .

Таблица 26. Цикл по переменной. Опуская подробности, цикл рисуют так

Блок-схема	**Конструкция `Do`**
	**do** *переменная* = xn, xk, step *блок_do* **enddo**

Цикл не выполнится ни разу, то есть kp=0, если

$$\begin{cases} xn > xk \\ step > 0 \end{cases} \quad \text{или} \quad \begin{cases} xn < xk \\ step < 0 \end{cases}$$

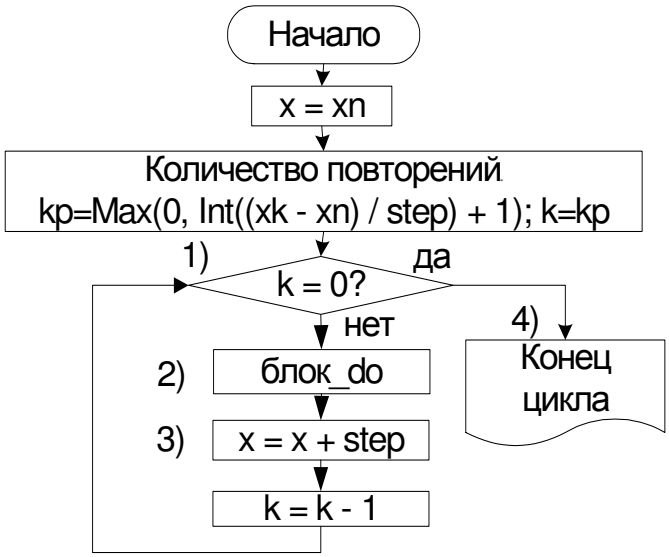
Шаг цикла *step* регламентируется следующим образом:

- step≠0;
- эквивалентны `Do` *i= 1,10,1* и `Do` *i = 1,10* – по умолчанию step=1;
- если step>0, *x* возрастает, по окончании цикла *x > xk;*
- если step<0, *x* убывает, по окончании цикла *x < xk.*

Переменная цикла *x* изменяется по закону арифметической прогрессии. Для понимания того, как это происходит, покажем подробную блок-схему, в которой проясняются все детали и правила для цикла по переменной.

Таблица 27.

Цикл по переменной. Подробная блок-схема цикла.

Подробная блок-схема цикла по переменной	
Блок-схема	**Пояснения**
	Здесь x – переменная цикла; xn, xk, step – константы, переменные или выражения. 1) **do** `x=xn,xk,step` 2) *блок_do* 3) **enddo** 4) конец цикла - *продолжение программы* **enddo** – это пункт (3), включающий в себя три действия: – изменение переменной *x*, – уменьшение *k* на 1, – переход к (1).

В соответствии с подробной блок-схемой цикла по переменной выполнение цикла регламентируется следующими правилами:

- переменная *x* в цикле `do` изменяется автоматически по закону арифметической прогрессии – изменять ее в *блоке_do* запрещено;
- не разрешается входить внутрь *блока_do* извне, поскольку в этом случае не определено число повторений цикла *kp* и переменная *x*;
- переменные *xn, xk*, step можно изменять в цикле – это не повлияет на число повторений уже запущенного цикла;
- оператор `cycle`, встретившийся в *блоке_do,* прерывает пункт (2) и вызывает переход к *end do* – пункту (3);
- оператор `Exit`, встретившийся в *блоке_do,* прерывает цикл, завершает его досрочно и вызывает переход к пункту (4) – на оператор непосредственно следующий в тексте за `end do`;
- при нормальном завершении цикла переменная x достигает значения *x = xn + step *kp,* а при досрочном выходе сохраняется достигнутое значение *x* из [*xn,xk*].

Цикл по *вещественной* переменной *x* имеет неустойчивый характер:

- при некоторых *xn, xk*, *step* цикл выполняется для значения *x = xk;*
- при других *xn, xk*, *step* цикл пропускает последнее значение *x = xk.*

Причина: ошибка в вычислении *kp* и накопление погрешности округления в *x=x+step*, особенно при сравнительно малом *step* по отношению к *x*. Такой цикл можно скорректировать, возможны три варианта:

- переделать радикально – построить цикл по целой переменной `inta`, через которую выразить вещественную переменную *x*. Количество повторений *kp* при этом следует рассчитать самостоятельно перед циклом; *int* будет изменяться точно, *x* не станет накапливать погрешность, хотя разовая погрешность сохранится

```
integer inta, kp;  real x
 kp = . . .  ! количество повторений цикла
 do int = 0, kp-1  !  kp – число повторений цикла
   x = xn + inta * step
   ! содержимое прежнего блока_do
 enddo
```

- «подправить» цикл по вещественной переменной, «обманув» `do` и переписав его так: `do` x = xn, xk + step/2, step
- выбрать шаг в виде числа, которое является степенью двойки и точно представляется в памяти компьютера, например, 4, 8, 0.25

Среди операторов *блока_do* может встретиться другая конструкция `do`. В этом случае говорят о вложенном цикле. Оценивая количество повторений оператора `write` внутри вложенного цикла в примере, показанном ниже, необходимо помнить, что при каждом повторении внешнего цикла, внутренний цикл повторяется заново.

Пример вложенных конструкций do

```
DO y = -1., 1., 0.125     ! 17 раз
  do x = -1., 1., 0.125   ! 17 раз
    WRITE(1,*) x,y        ! 289=17*17 раз
  enddo
ENDDO
```

В файл будет выведена 289 пар значений *x* и *y*.

3.5. Параллельные конструкции *where* и *forall*

Условия во вновь появившихся конструкциях Фортрана-95 стали векторными, матричными (в общем случае многомерными). Появилась возможность их варьировать покомпонентно. Прежние управляющие конструкции Do While, If и Select case, *ориентированные на скаляры,* для этих целей не подошли. Условия в управляющих конструкциях должны быть *конформны* проводимым вычислениям, см. раздел 5. Новые управляющие конструкции как раз реализуют *параллельные* действия над компонентами векторов и матриц (вообще, массивов) под управлением компонент *конформной им маски,* заданной логическим выражением. Имеется лишь две конструкции: where и forall (появилась в Fortran-95).

Оператор и конструкция *where*

Ветвление where по форме записывается аналогично if в трёх разновидностях с количеством блоков два, один, ноль

Таблица 28.

Сравнение операторов *if* и *where*

Оператор if	**Оператор where**
if (скаляр-условие) **then** ! *2 блока* **...!** *блок ДА* **else if** **....!** *блок НЕТ* **end if**	**where** (конформное_условие) ! *Два блока* **....!** *блок ДА - _действия, конформные условию* **else where** **....** ! *блок НЕТ - действия, конформные условию* **end where**
if (скаляр-условие) **then** ! *1 блок* **...!** *блок ДА* **end if**	**where** (конформное_условие) ! *Один блок* **....!** *блок ДА - действия, конформные условию* **end where**
if (скаляр-условие) !*один простой* ! без блоков	**where**(конформное_условие) !*один конформный* ! *без блоков*

Отметим внешнее сходство if и where – оба *ветвления*. Однако между ними есть кардинальные отличия:

- if задает *скалярное ветвление,* группируя действия над *скалярами* или *векторами целиком;*
- where – это *векторное покомпонентное ветвление,* которое группирует *параллельные* действия над *компонентами* векторов, матриц, многомерных массивов, секций с непременным требованием *конформности условия и действий.*

Имеется также определенное сходство where с do – оба могут обрабатывать массивы. Правда, do может задавать закон изменения лишь одного индекса массива по какому-либо измерению, а where обеспечивает работу по всем компонентам массива независимо, не важно, сколько в нем измерений. Следует подчеркнуть, что имеется принципиальное различие do и where:

– *Do* обрабатывает компоненты вектора последовательно;
– where обрабатывает компоненты вектора независимо, параллельно.

Так что where - это вовсе не цикл, а перечисление потенциально *параллельных процессов.* Реально параллельность достигается только в случае распараллеливающего компилятора и наличия многих процессоров. В случае, когда компилятор не умеет распараллеливать, конструкция where для одномерного массива как бы заменяет do и if.

Порядок действий, соответствующих where**:**

– вычисляется конформная маска;
– затем над всеми элементами конформных массивов выполняются конформные действия - одно или несколько в последовательности заданной в блоке where**;**
– действия выполняются только для истинных значений в маске;
– действия выполняются *параллельно.*

Как всегда, *параллельность* – это прежде всего независимость процессов, потенциальная способность – в том смысле, что это станет реальностью:

– если будет использован распараллеливающий компилятор IFC;
– если процессоров будет предостаточно;
– если процессоров будет недостаточно, то поочередно пачками, как решит компилятор и операционная система;
– ввиду многих задач и многих пользователей ситуация по числу свободных процессоров может меняться, что затрудняет наши измерения времени решения задачи;
– если компилятор не распараллеливает или процессор единственный, то действия выполняются поочередно по всем элементам, правда, программа выглядит короче.

Пример: найти сумму обратных величин матрицы. Приведем три варианта решения.

1. Рассмотрим сначала на первый взгляд простое решение, которое можно охарактеризовать как “ловушку для начинающих”

```
Real   SumObr
Real,dimension(1:10,1:10) :: A
SumObr=Sum( 1/A, A/=0 )
```

Ошибка состоит в непонимании порядка действий

- сначала вычисляем аргумент 1/A и рискуем ”поделить на 0” для нулевой компоненты вектора A;
- потом суммируем по маске A/=0.

2. Вычисление обратной величины *Obr* логично выполнить в виде двухблочной конструкции where

```
Real   SumObr
Real,dimension (1:10,1:10):: A, Obr
where (A/=0 ) ! с двумя блоками
  Obr=1/A      ! где можно – вычислить обратную величину
else where
  Obr=0        ! где нельзя – обнулить обратную величину
end where
SumObr=Sum(Obr)
```

3. Вычисление обратной величины можно выполнить и в виде простого оператора where

```
Real   SumObr
real,dimension(1:10,1:10) :: A,Obr
  Obr=0;
  where (A/=0) Obr=1/A ! безблочный оператор
    SumObr=Sum(Obr)
```

Оператор и конструкция forall

В конструкции where управляющая маска задается в векторной форме, а в конструкции forall маска задается более гибко в виде индексированных переменных. Аналогично и выражения в блоке задаются по отношению к индексированным переменным. Другими словами выполняются маскированные конформные действия в пространстве некой секции. Пространства изменения индексов такой виртуальной секции задаются в заголовке forall триплетами, аналогично заданию триплетами секций массивов. Заголовок конструкции forall задает триплеты по ряду измерений и маску отбора к полученной секции по смыслу аналогичную where**.**

Аналогично Do While конструкция forall управляет блоком операторов, но в отличие от Do While управляющая маска векторная, в общем случае многомерная, а не скалярная. Действия над компонентами секции выполняются параллельно и независимо.

Триплеты, указывающие законы изменения индексов в секции, и маску для секции задают

- в простом безблочном операторе
  forall (*индексИимярек*=*триплет*, **..**[, *маска_секции*]) *единственный_простой_конформный*
- или в составном одноблочном операторе
  forall (*индексИмярек*=*триплет*, .. &

индексИмярек=триплет [, *скалярная_маска_секции*])
! *маскированные действия, конформные секции, построенной*
! *по всем возможным комбинациям индексов из триплетов*
end forall

Сначала рассмотрим четыре варианта решения простой задачи. На этих решениях покажем разницу в понимании do и forall.

Пример: найти *diaPro* – произведение ненулевых элементов главной диагонали.

1. Правильное решение с традиционными конструкциями do и if

```
real,dimension(1:10,1:10) :: A
real diaPro; integer i
diaPro =1
do i=1,10
  if( A(i,i)/=0)   diaPro = diaPro* A(i,i)
enddo
```

2. Заменив do на forall, получим *неправильное* решение

```
real,dimension(1:10,1:10) :: A
real Diapro; integer i
Diapro =1
Forall(i=1:10, A(i,i)/=0) Diapro = Diapro* A(i,i)
```

Допущена ошибка в левой части оператора *Diapro = diapro* A(i,i)* – это присваивание, не конформное заголовку оператора *Forall*: *diaPro – скаляр*, в то время как *A(i,i)* определено в рамках секции 1:10.

3. Правильное решение с использованием оператора forall

```
real,dimension(1:10,1:10) :: A
real,dimension(1:10) :: Diagonal
real diaPro; integer  i
Diagonal =1;
forall (i=1:10, A(i,i)/=0)  Diagonal(i)= A(i,i)
diaPro= product(Diagonal)
```

4. Нет решения с помощью оператора where

У forall есть аналоги:

- where - оба задают работу с многомерным объектом;
- вложенными do - обе конструкции задают закон изменения индексов в виде арифметической прогрессии по триплету в заголовках;
- do - и в теле оператора forall элементы массивов адресуются по индексам, заданным в заголовке forall.

Однако смысл адресации элементов массива в do и в forall абсолютно разный:

- в do элементы массива выбираются *последовательно* по мере изменения индексов;

- в `forall` элементы массива выбираются независимо-*параллельно*, по предварительно построенным всем возможным комбинациям индексов, порядок обработки в языке не фиксирован:

Таким образом `do` - основа *последовательного* выполнения повторяющихся действий; `forall` - основа *параллельного* выполнения повторяющихся действий.

Любое присваивание массивов и `where` можно переписать как `forall`, но некоторые `forall` не могут быть записаны только на уровне манипуляций с массивами в `where`:

- например, через `forall` легко записать: `where` (A /= 0.0) B=1.0/A - сами напишите;
- однако, следующий пример `forall` нельзя записать на уровне манипуляций с массивами

  `forall` (i=1:n, j=1:n) H(i,j) = 1.0/(i+j-1)
- этот оператор устанавливает элемент массива H(i, j) равным значению 1.0/ (i + j - 1) для любых пар значений i и j между 1 и n.

`Forall` задает:

- список имен индексов, варьируемых в секции;
- каждому поименованному индексу сопоставлен триплет, определяющий закон его изменения;
- после списка задается необязательная маска, конформная секции, которая записывается в терминах варьируемых индексов;
- над какими компонентами массивов, какие именно действия и в какой последовательности - задается в блоке оператора `forall;`
- порядок обработки компонент в языке не фиксирован.

Порядок действий, соответствующий `forall:`

- по заданным триплетам создаются списки всех возможных значений индексов;
- далее формируются все комбинации индексов;
- по комбинациям индексов формируется секция и конформная ей маска;
- затем независимо и *параллельно* выполняются конформные действия в последовательности заданной в блоке;
- действия выполняются только для истинных значений конформной им маски.

Получается, что это не цикл, а перечисление потенциально параллельных процессов. Синтаксис записи триплетов в `forall` через двоеточие ещё раз подчёркивает:

- уместность аналогии с секциями, а не циклами;
- параллельность, а не последовательность;
- чтобы потенциально параллельная конструкция стала реально таковой, необходимы два условия:
  1. распараллеливающий компилятор, как IFC;
  2. наличие многих процессоров.

Различие скалярных и векторных конструкций

Поскольку `forall` не цикл, а перечисление параллельных процессов, то напрашивающаяся аналогия с циклом ограниченна.

В ряде случаев операторы `do` и `forall` одинаковы по действию. Тривиальный пример:

```
Integer, dimension(1:4) :: A
A=0    ! параллельные действия
forall (i=1:4)     ! параллельные действия
  A(i)=0
end forall
do i=1,4  ! последовательные действия
  A(i)=0
end do
```

Простой пример, из книги Немнюгина, иллюстрирующий разницу между `do` и `forall`

1. Используется оператор `Do`

```
integer,dimension(1:4):: A=(/ 1,2,3,4 /)
Do i=2,3     ! последовательные действия
  A(i) = A(i-1) + A(i) + A(i+1)
Enddo
```

Получим

A(2) = A(1) + A(2) + A(3)=1+2+3=6

A(3) = A(2) + A(3) + A(4)=6+3+4=13 ! A(2)=6 - *изменившееся значение*

2. Оператор *forall***,** перечисляющий параллельные процессы над исходными значениями массива, – те же по виду действия дают другой ответ

```
forall ( i=2:3 ) ! параллельные действия
  A(i)=A(i-1)+A(i)+A(i+1) !подставляется значение A( 2)=2
 ! то же по-другому A(i)=sum(A(i-1:i+1))  - проясняет суть дела
End forall
```

Получим

A(2) = A(1) + A2) + A(3)=1+2+3=6

A(3) = A(2) + A(3) + A(4)=2+3+4=9 ! A(2)=2 исходное значение

A(2:3) = (/ **sum**(A(1:3)) , **sum**(A(2:4)) - всё разъясняет: берем старые значения, а заносим новые значения A

и ни к чему здесь `forall` **-** это чисто академический стиль

Нетрудно заметить внешнее сходство `Do` - цикл и `forall` **-** как бы цикл по параллельным процессам. Не проходит аналогия `forall` с циклом *foreach* из языка PHP.

Есть кардинальное отличие цикла и перечисления параллельных процессов `forall`**:**

- `do` задает последовательность действий, которые выполняются над скалярами либо целиком векторами;

- `forall` задает параллельные действия, которые выполняются над компонентами векторов, матриц, многомерных массивов, секциями с непременным требованием *конформности* условия и действий;
- `do` работает с текущими значениями массивов, а `forall` как бы работает с двойной памятью - использует стартовые значения массивов, а формирует новые значения.

Парафлоид – один оператор вместо двадцати

В этом разделе показан более сложный пример вычисления парафлоида - квадратной матрицы минимальных расстояний между вершинами ориентированного графа с взвешенными дугами. Граф задан квадратной матрицей инцидентности. Строки и столбцы матрицы соответствуют вершинам графа. Инцидентные дуги соответствуют положительным компонентам матрицы. Схематично показано вычисление меры близости вершин графа по методу Флойда.

Таблица 29.

Пример вычисления парафлоида в операторе *forall*

A	**J**	
i	parafloid**(i,j) = minval(** A(i,:)+A(:,j))	**A(i,:)– строка i**
	A(:,j)– столбец j	

С целью упрощения на схеме не показаны условия положительной определенности, дополнительно прописанные в операторе *forall*.

Парафлоид - единственный выполняемый оператор

```
program parafloida
 implicit none
  integer,parameter:: N=10
   ! A - взвешенная входная матрица инцидентности
   integer,dimension(1:n,1:n) :: A=1, parafloid
      ! парафлоид parafloid - выходная матрица мин. расстояний
    integer i,j
      forall( i=1:n, j=1:n, i/=j ) &
       parafloid(i,j) = minval( A(i,:)+A(:,j), &
                 mask = A(i,:)>0 .and. A(:,j)>0 )
      write(6,1) (parafloid(i,:),i=1,n)
     1 format(1x,<n>i3)
end program parafloida
```

Попробуйте написать программу `paraseqva` с помощью традиционных конструкций – и Вы увидите, что она будет на порядок сложнее: вместо одного оператора - программа из 20 операторов.

```
program paraseqva ! взвешенная входная матрица инцидентности
 implicit none
  integer,parameter:: N=10
   integer,dimension(1:n,1:n) :: A=1, parafloid
                ! парафлоид - выходная матрица мин. расстояний
     integer i,j
     integer,dimension(1:n) :: str,stlb, summa
      integer   parafloidMin,i1
      open(6,file='parafloid.txt')
      do i=1,N
        A(i,i)=0
      enddo
      do i=1,N
       do j=1,N
        if(i/=j) then
         parafloidMin = huge(0) ! #7fffffff
         do i1=1,N
          str=A(i,i1);  stlb=A(i1,j);  summa(i1)=str(i1)+stlb(i1)
         enddo
         do i1=1,N
          if(str(i1)==0 .or. stlb(i1)==0) cycle
          if(summa(i1)<parafloidMin)  parafloidMin = summa(i1)
         enddo
         parafloid(i,j) = parafloidMin
        else
         parafloid(i,j) = 0
        endif
       enddo
      enddo
      write(6,1) (parafloid(i,:),i=1,N)
     1 format(1x,<N>i3)
end program paraseqva
```

Пример парафлоида – многогранный:

- `A(i,:)` - i-ая строка и A(:,j) - j-ый столбец заданы секциями двумерного массива, образующими векторы;
- конформные векторы *`A(i,:)`* и *A(:,j)* - суммируются *A(i,:)+A(:,j)*, образуя новый вектор положительно определенных крестообразных сумм;
- из сумм выбирается минимум при помощи итоговой функции `minVal`, формируя элемент парафлоида;
- `forall` задаёт перебор в пространстве секции, совпадающей с положительно определёнными недиагональными элементами матрицы;
- данный `forall` имеет объём вычислений, пропорциональный O (N^3).

4. Ввод и вывод в Фортране

Ввод и вывод осуществляется операторами `read` и `write` (по-русски «читаем/пишем»). После операторов `read` и `write` в скобках записываются аргументы. Первый аргумент (целое) – условный номер устройства, с которого/на который осуществляется ввод/вывод. Всегда открыта для ввода/вывода консоль, имеющая номер 0 или обозначаемая символом «*».

Второй аргумент – ссылка на формат ввода/вывода. Символ «*» в качестве второго аргумента означает, что ввод/вывод выполняется без формата. Бесформатную форму чаще всего применяют при вводе и для отладочной печати. Если важны не только сами значения, но и форма их представления, то используют форматный вывод.

Бесформатный ввод/вывод

Бесформатный ввод/вывод выполняется под управлением списка

read(u,*) список_переменных

write(u,*) список_переменных_констант_выражений

Пример: прочитать переменные p и k и вывести их для контроля:

```
integer k; real p
read(1,*)p, k ; write (2,*)'p=', p, 'k=', k
```

Обмен данными происходит в порядке следования элементов списка. В операторе *read* с устройства №1 читается сначала p, потом k. В операторе *write* на устройство №2 в одной строке выводятся последовательно: надпись «p=», значение p, надпись «k=», значение k.

Данные для ввода подготавливают через запятую, пробел или <Enter> – конец строки. Запись одинаковых чисел упрощает повторитель, например 8,8,8,8,8,8,8, легче записать как 7*8 (в данном случае «7*» – это повторить 7 раз следующее число).

Ввод/вывод по списку переменных, с каждой как «p= значение_p»: **Namelist** /имя_Namelist / список_переменных *! подробности – в помощи* **read**(u,имя_Namelist) или **write**(u,имя_Namelist)

4.1. Форматный вывод

Вывод без формата, применимый при отладке программ, дискомфортен для пользователя. Для создания привлекательного программного продукта требуется максимум внимания уделить оформлению результатов. Сочетая гибкость и математическую строгость, форматный вывод тут незаменим.

Фортран выгодно отличается от других языков тем, что ввод и вывод – операторы языка, и можно выверить форматы еще при компиляции. Концепция форматного вывода Фортрана настолько продуманна, что практически не изменялась с момента его появления. Формат размещают в любом месте программы и пишут следующим образом

m **format**(СД), где m – метка, по которой один или несколько операторов `write` ссылаются на оператор `format`, СД – список дескрипторов.

Пример:

```
real radius = 1
   Open (2, file='Out.txt')
   write(2,55)  2*3.14*radius
55 format(1x,'длина окружности'/2x,'c=',F5.2)
```

Здесь оператор `write`, используя оператор `format` с меткой 55, выводит в файл *Out.txt* две строки (символ «˜» означает пробел)

```
длина окружности
˜˜c=˜6.28
```

Дадим разъяснения:

– «длина окружности» и «c=» – пояснительные надписи;

– символ «/» – признак перехода к новой строке при выводе;

– 2x – 2 пробела,

– F5.2 – поле вещественного числа шириной в пять позиций с двумя цифрами, выводимыми после десятичной точки.

Уникальный СД может быть записан непосредственно внутри оператора *write* в виде строковой переменной или константы, например,

```
write(2, "(1x,'c=',F5.2 ) ") c
```

Внимание! Такой СД не контролируется компилятором, и ошибки выявятся на этапе счета, в отличие от использования оператора *format*.

Независимо от того, где СД записан – в операторе *format* или в операторе *write,* должны соблюдаться следующие правила:

1. СД заключен в скобки, элементы СД разделены запятыми или символом «/» по концу выводимой строки;
2. повторяющуюся последовательность дескрипторов заменяет повторитель, записанный перед дескриптором; например, вместо i4,i4 пишут 2i4;
3. повторяющуюся группу дескрипторов заменяет повторитель, записанный перед скобками, заключающими эту группу; например, вместо 1x,i4,i4,i4, 1x,i4,i4,i4 пишут 2(1x,i4,i4,i4) или, еще короче, 2(1x,3i4).
4. Все повторители, ширины полей – это явные натуральные константы или целочисленное выражение в угловых скобках < >;
5. дескрипторы в Фортране обозначены одной, реже двумя латинскими буквами, и подразделяются на:
   - дескрипторы числовых данных I, F, E, D и универсальные G, Z;
   - дескрипторы нечисловых данных A, L, и универсальные Z, G;
   - оформительские дескрипторы: ”текст”, ’текст’, X, P, SS, SP, T, TR, TL, «:», «\», «/».

4.2. Дескрипторы данных

Таблица 30.

Тип	Дескрипторы	Примеры
integer k	iw	**write**(1,"(1x,I5)")k
real x	Fw.d - без порядка Ew.d - с порядком Gw.t - с порядком и без	**write**(1,"(1x,F7.2)")x **write**(1,"(1x,E11.2)")x **write**(1,"(1x,G11.2)")x
double precision d	Dw.d - двойная точность	**write**(1,"(1x,D20.13)")b

Пояснения к форматному выводу чисел:

1. *w*-общая ширина поля, задаваемая по усмотрению программиста, число будет прижато к правому краю и дополнено слева пробелами, если число не поместится в отведенное поле, то увидим «*****»;
2. для вещественных чисел количество цифр задается параметрами:
   d – сколько цифр после десятичной точки в дескрипторах *F, E, D*;
   t – сколько значащих цифр в универсальном дескрипторе Gw.t, компьютер «решит», выводить число с порядком, как *E* или без как в *F*;
3. для вывода чисел с большим по модулю или неизвестным порядком рекомендуется применять *E, D, G*.

Выводимая строка прижимается к левому краю поля.

Таблица 31.

Дескрипторы нечисловых данных

Тип	Дескриптор	Примеры	Примечания
Logical M	Lw	**write**(1,"(1x,L5)")M	Вывод только «T» или «F»
Character[*c] S	Aw	**write** (1,"(A5)") S	Усекается или дополняется справа пробелами в зависимости от соотношения *w* и *c*.
m format [*c] S	A	**write** (1,"(1x,A)")S	ширина поля *c*, указана в описании строки Character[*c].

4.3. Взаимодействие операторов write и format

1. Оператор *write*
   write(*номер устройства, ссылка на формат*) *список вывода*
   пояснения:
   – *куда выводить* – направление вывода в виде *номера устройства*;
   – *что выводить* – в *списке вывода* перечислены выводимые данные, этот список просматривается однократно;
   – *как выводить* – данные из списка вывода запрашиваются по мере просмотра СД оператора format.

2. Оператор `format` описывает, *как* выводить данные, интерпретируя список дескрипторов

   метка `format(`*список дескрипторов*`)`

   пояснения:

   – по дескриптору данных запрашивается и выводится элемент списка вывода; если дескриптор данных не подходит элементу списка вывода, то сообщается *expected descriptor* об ожидавшихся дескрипторах;
   – по другим дескрипторам выводятся надписи, пробелы, переносы;
   – по концу СД – просмотр повторяется на участке сканирования.

Длина списка вывода может быть равной, больше или меньше длины СД. Когда список вывода длиннее, чем СД, просмотр автоматически зацикливается на участке сканирования, который определяется однозначно:

– оператор `format` без внутренних скобок

```
1 format ( .. .. )
          ↑____∞___|   – повторение по скобкам формата ;
```

– оператор `format` с внутренними скобками

```
2 format((..)..(..(..)..)..)
                ↑____∞_______|
```

повторение по последней паре скобок внутри оператора `format` .

Принцип сканирования реализуется по следующим правилам:

– каждый новый оператор *write* начинает вывод с новой строки, устанавливает начало списка вывода и начало списка дескрипторов;
– если встретился дескриптор данных, запрашивается и превращается в текст очередной элемент списка вывода;
– попутно исполняются встретившиеся оформительские дескрипторы: по дескриптору «X» вставляются пробелы, вносятся текстовые константы, по дескриптору «/» переносятся строки;
– участок сканирования циклически просматривается и интерпретируется до тех пор, пока не будут обработаны все элементы списка вывода;
– при возврате к началу участка сканирования вывод продолжается с новой строки;
– по концу списка вывода продолжится просмотр и исполнение оформительских дескрипторов до ближайшего дескриптора данных или до дескриптора «:» .

Пример 1 *– на понимание процесса сканирования.*

Вывести два целых по формату с одним числовым полем i2.

```
integer::x=12,y=100
Open (2, file='Out.txt')
write(1,33) x, y    ! x, y – список вывода
33 format(i3)   ! (i3) – участок сканирования
```

Поскольку длина списка вывода больше длины списка дескрипторов, возникает сканирование, то есть повторение формата и перенос строки. Повторяется весь список дескрипторов, так как внутри формата скобок нет. В файл *Out.txt* выведется две строки, начинающиеся с пробела – дескриптор *1x* (символ «˜» означает пробел)

```
˜12
100
```

Формат пригоден для вывода любого количества целых чисел из диапазона [-99 , 999], которые вмещаются в 3 позиции.

Пример 2 *– число не помещается в отведенное поле*.

Вывести два целых по формату с одним числовым полем i2.

```
integer::x=12,y=100
Open (2, file='Out.txt')
write(1,33) x, y    ! x, y – список вывода
33 format(1x, i2)   ! (1x, i2) – участок сканирования
```

Поскольку длина списка вывода больше длины списка дескрипторов, повторяется весь список дескрипторов, так как внутри формата скобок нет. В файл *Out.txt* выведется две строки, начинающиеся с пробела – дескриптор *1x* (символ «˜» означает пробел)

```
˜12
˜**
```

Формат пригоден для вывода любого количества целых чисел из диапазона [-9 , 99]. Вместо трехзначного числа 100 выведется «**», поскольку ширина поля вывода равна 2 (дескриптор *i2*).

Пример 3 *– бесконечно глупый вывод*.

Вывести целое по формату, в котором нет дескрипторов данных.

```
write(1,33)12 ! целое 12 бесконечно ищет дескриптор i в 33 format()
33 format('ищем целый дескриптор, а его нет')
```

Внимание: просмотр фактически зациклился !

5. Массивы

5.1. Характеристики массива

В современном Фортране свойства объектов, в частности, массивов описывают так:

список_атрибутов **::** *список_объектов*

Характеристики и атрибуты массива.

1. *Имя* массива, как объекта.
2. *Тип* массива – одного из базовых типов *integer*, *real*, *complex*, *logical*, *character* или определяемого типа *type*(имя_типа).
3. *Количество измерений* или *ранг* массива – допускается от 1 до 7, причем по каждому измерению указывается пара целых чисел [*нижняя*:]*верхняя* границы индекса:

   a. либо *диапазон* индекса как *нижняя*:*верхняя* границы индекса *протяженность* = *верхняя* – *нижняя* + 1 ; *протяженность* – это неотрицательное число, указывающее на количество возможных значений индекса по выбранному измерению;
   b. либо только *верхняя* граница, с нижней границей по умолчанию равной 1; в этом случае *верхняя* граница - это *протяженность*, например, число строк или число столбцов матрицы.
4. *Форма* массива, по-английски *Shape* – целочисленный вектор, составленный из протяженностей в порядке следования измерений. *Форма* характеризует порядок размещения элементов в памяти компьютера. Массивы, одинаковые по форме, называют, *конформными*. *Форма* измеряется функцией `shape(`*Array*`)` **.**
5. *Число элементов* – это произведение протяженностей по всем измерениям массива, измеряется функцией `size(`*Array*`)` `size(`*Array*`)=product(shape(`*Array*`))` **.**
6. *Объем памяти,* занимаемый массивом <*объем_памяти* > **=** < *память* на один элемент >* *size(array)* – ограничивается лишь объемом памяти компьютера.
7. Когда и как массив размещается в памяти компьютера – статический или динамический (*allocatable*); в процедуре, дополнительно – динамический автоматический, передаваемый по адресу.
8. Назначение (*intent*) массива как аргумента процедуры: `intent(in)` – входной аргумент, `intent(out)` – выходной аргумент, `intent(inout)` – входной-выходной аргумент.
9. Прочие атрибуты: *public* | *private*, *pointer* | *target*, *parameter*, *save*.

Предупреждение. Выбор элемента неописанного массива компилятор воспринимает как вызов неизвестной функции, причем об этом сообщает не компилятор, а компоновщик приложений.

Пример. Подсчитать *s* – среднее значение и *D* – среднее отклонение от среднего значения в массиве *x*

$$S = \frac{1}{10}\sum_{k=1}^{10} x_k \text{ и } D = \frac{1}{10}\sum_{k=1}^{10}\left|x_k - S\right|.$$

В раннем Фортране массив суммировали поэлементно, в Фортране-90 всё выглядит проще – «читаем – считаем – печатаем». Добиться краткости позволяет важный принцип современного Фортрана – практически везде, где используется простая переменная, можно использовать массив, как в `abs(`*X*`-`*s*`)`, кроме того, функция `sum(`*X*`)` суммирует массив.

```
Program sumX ! sum(X)-встроенная функция суммирования вектора
 Real,Dimension(1:10)::X;  real s
 read(1,*)X; s=sum(X)/10; print *, 'среднее =',s
! читаем – считаем – печатаем
 print *, 'среднее отклонение D=',sum(abs(X-s))/10
End Program sumX
```

5.2. Размещение массива в памяти компьютера

Говоря о размещении массива в памяти, следует прояснить:

- какой *объем памяти* требуется;
- *в каком порядке* элементы массива хранятся в памяти;
- *когда* массив размещают в памяти.

Какой объем памяти требуется?

Объем памяти, занимаемый массивом, определяется просто из-за того, что все элементы однотипны и занимают одинаковый объем памяти. <*объем_памяти*> = <*память* на один элемент>***size**(array)* Размер массива ограничивает лишь объем памяти компьютера.

В каком порядке элементы массива хранятся в памяти?

В большинстве случаев программист об этом может не задумываться. Однако, в ряде случаев при передаче массива целиком подразумевается именно этот порядок:

- при подготовке данных для ввода;
- при написании форматов вывода;
- при задании начальных значений для элементов массива.

Порядок хранения стандартизован в Фортране: для одномерного массива – по возрастанию индекса, для матрицы – по столбцам, для многомерного массива – самый быстрый первый индекс, медленнее второй, наконец, последний – самый медленный, каждый из индексов пробегает все значения по возрастанию.

Пример. В каком порядке надо готовить данные?

Всего в трехмерном массиве *P(1:2,1:2,1:2)* 8 элементов. Его элементы располагаются в памяти, как указано выше, и порядок чтения разъясняется вторым оператором *read*. Именно в этом порядке и надо готовить данные. Так как *read* бесформатный, можно, не нарушая порядка следования, либо все числа перечислить в одной строке через пробел или запятую, либо – каждое число – в отдельной строке.

```
dimension P(1:2,1:2,1:2)
read(1,*) P ! читать весь массив – понимают как
read(1,*) P(1,1,1),P(2,1,1),P(1,2,1),P(2,2,1),P(1,1,2),P(2,1,2),P(1,2,2),P(2,2,2)
```

Предупреждение. Размещение массива в памяти стандартно и никак не зависит от порядка ввода.

Когда массив размещают в памяти?

Форма *статическ*ого массива, (предыдущий пример), известна при компиляции – *shape(P)=(/2,2,2/)*; массив размещается в памяти при запуске программы. Форма *динамическ*ого массива не определена при компиляции, например,

```
Real,allocatable,Dimension(:,:) :: Matrix, Matr1, Matr2, X(:)
```

Известно лишь количество измерений массивов: два для *Matrix* и один для X. Запустив программу, сначала определяют протяженности, а затем динамически размещают массивы в памяти оператором

```
allocate( X(1:N), Matrix(1:M,1:N),&
                Matr1(1:N ,1:M), Matr2(1:N,1:N) )
```

Освобождают память оператором

```
deallocate( X, Matrix, Matr1, Matr2 )
```

Массив *как аргумент процедуры* должен быть размещен в памяти до входа в процедуру; в нее передается его адрес, независимо от способа написания размерности:

Real,intent,Dimension(1:N)::Mas - *в виде переменной N;*

Real,intent(in),Dimension(1:10)::Mas - *в виде константы 10;*

Real,intent(in),Dimension(:)::Mas – *без указания размерности;*

Real,intent(in),Dimension(*)::Mas–*массив, перенимает форму;*

Dimension(1:N,1:M) – *в виде массива с перераспределением памяти между строками и столбцами,* например, *3*4=4*3=2*6=6*2=12.*

В любом случае количество элементов можно узнать с помощью функции size*(array)*, а для матрицы число строк – size*(Matrix,dim=1)*, число столбцов – *size(Matrix, dim=2)*.

Пример (массивы *X, Y, P, Q конформны*):

```
real,dimension(1:11):: X,Y ! массивы X,Y по 11 элементов
real,dimension(-5:5)::P,Q ! P,Q – по 11 элементов с номерами 5,..0,..5
   Y=sin(X+0.5)/4 + 2.1; p= Y/2; Q=P-1
```

5.3. Секции массивов и неявный цикл в списках ввода/вывода

Требуя гибкой организации, ввод/вывод массивов часто подразумевает строчную форму написания цикла в списке ввода/вывода. Неявный *Do* – это конструкция вида read(..) (*v*, *i*=i_H,i_K,i_S); write (..) (*v*, *i*=i_H,i_K,i_S) , где *v* -список переменных, выражений, элементов массива, зависящих от *i*. Скобки как бы заменяют do .. enddo , ограничивая область действия цикла по переменной *i*. Возможно вложение циклов. Поскольку речь идет об индексах, то i, i_H, i_K, i_S – целые величины. Триплетом называют тройку целых величин iн, iк, is , записанных в отличие от цикла через двоеточие $i_н$:$i_к$:i_s. Как можно заметить из примеров, приведенных ниже в Табл.32, триплеты можно употреблять вместо индексов, как в M(i,1:3:2). Известно также, что шаг is=1, если он опущен M(i,1:3). В отличие от цикла в триплете можно опускать не только шаг, но и $i_н$ – M(i,:3), и $i_к$ – M(i,1:) и оба – M(i,::2), и даже все три элемента триплета вместе – M(i,:). Поэтому, например, M(i,:) и M(i,1:3) эквивалентны. Опущенные $i_н$-начало и $i_к$-конец выбираются из описания массива. Понятно, что M(:,:) писать не стоит, потому что это эквивалентно M. Секция массива – это тоже массив, который можно применять почти везде, где применяют массивы. Как массив секция имеет свою форму.

Часто наряду с секциями используют *неявный цикл* `Do`, когда требуется изменить стандартный порядок ввода/вывода. И ту, и другую конструкцию применяют, в частности, для ввода-вывода.

Таблица 32.

Ввод-вывод на матрице **`Integer,dimension`**`(1:3,1:4)::`M

Задание	**Неявный `Do`**	**Используя секции**
Ввод 1-ой строки	**`read`**(*,*) (M(1,j), j=1,4)	**`read`**(*,*) M(1,1:4)
Вывод по строкам	**`read`**`(1,*)` ((M(i,j),j=1,4),i=1,3)	**`read`**(1,*) (M(i,:),i=1,3)
Вывод матрицы по строкам	**`write`**(*,7) (i, M(i,:),i=1,3) 7 **`format`**`(1x,'№',i1,':',4i3)`	
То же, но длиннее, с номерами	**`write`**`(*,7)` (i, M(i,1:4),i=1,3)	
Ввод 2-го столбца	**`read`**`(*,*)` (M(i,2),i=1,3)	**`read`**(*,*) M(:,2)
Ввод диагонали матрицы	**`read`**(*,*) (M(i,i),i=1,3) В данных - диагональ матрицы	

6. Программы, модули и механизмы обмена данными

Простейшая задача - главная программа. Сложную задачу разбивают на подзадачи, и реализуют в виде программ и модулей. Главная программа вызывает *процедуры*, те в свою очередь другие процедуры, и т.д. Объекты, обрабатываемые многими программами, помещают в централизованное хранилище – *модуль*. Программы и модули, объединяют термином *программная единица* (ПЕ) - конструкции четырех видов в Табл. 33, 34.

Таблица 33. Виды программных единиц

`Программные единицы`			
Программы – носители данных для внутренних процедур			**`Модуль`** – носитель данных для тех, кто его использует, и для модульных процедур
`Главная программа`	**Процедуры**		
	`Подпрограмма`	**`Функция`**	
`Program` P **`use`** M описания действия **Contains** внутренние процедуры импортируют описания внешнего носителя **`end Program`** P	Subroutine S(аргменты) **`use`** M описания действия **Contains** внутренние процедуры, с импортом описаний внешней программы-носителя **`EndSubroutine`** S	**Function**F(аргументы) **`use`** M описания, включая F действия, включая F=выражение **Contains** внутренние процедуры импортируют описания внешнего носителя **`End Function`** F	**`Module`** M **`use`** другой_модуль описания ~~действия~~ **contains** модульная процедура **contains** *внутренние процедуры* **`end`** мод. процедуры .. модульные процедуры **`End Module`** M

Программы состоят из описаний объектов и действий над ними, модули – только из централизованных описаний объектов и процедур доступа к ним из других программ и модулей. Процедуры вызывают, чтобы выполнить предусмотренные действия, среди которых также могут быть новые вызовы процедур. Помимо возникающих цепочек вызовов программ в сложной задаче строятся цепочки использования модулей. Циклы в цепочке вызовов программ (*рекурсия*, то есть вызов самое себя) допустимы, а циклы в цепочке использований модулей (*тавтология*) недопустимы.

Таблица 34.

Программная единица может быть .. внешней, внутренней, модульной

П р о г р а м м ы			**Модуль** как хранилище данных (только внешний)
Главная программа	Процедуры		
	Подпрограмма	Функция	
Program P	**Subroutine** S(аргументы)	**Function** F(аргументы)	**Module** M
Program может быть только внешней	**Subroutine** может быть внешней, внутренней, модульной	**Function** может быть внешней,внутренней, модульной	Модульные процедуры с внутренними процедурами
Объект модуля доступен по имени в ПЕ, где **модуль** использовали			

6.1. Двухуровневая структура программ

Решая сложную задачу, программист выстраивает систему обозначений объектов, комбинируя компоненты проекта:

- *внешние* программы (с пачкой внутренних программ) имеют независимые системы обозначений и могут компилироваться отдельно;
- *внутренние* процедуры, будучи вложены во внешние программы, не являются самостоятельными и импортируют обозначения *носителя* данных, а потому не могут компилироваться отдельно;
- *внешние* и, транзитом, внутренние процедуры импортируют общедоступные обозначения модуля;
- сопоставляются аргументы вызывающей/вызываемой программ.

Во *внешнюю* программу после оператора `contains` может быть вложена одна или несколько внутренних процедур. Внутренняя процедура компилируются только в составе внешней ПЕ, внутреннюю процедуру нельзя вызывать извне. Кроме того, внутренние процедуры импортируют обозначения *носителя* данных – внешней ПЕ. Программа может быть без оператора `contains` и без внутренних процедур, как в Фортране-77, где не было внутренних процедур.

Разновидности программ.

В проекте одна главная программа и любое количество процедур и модулей. В таблице 35, приведенной ниже, объекты пронумерованы [1-7], а курсивом выделены части, необязательные в той или иной из перечисленных разновидностей программных единиц:

1. у *внешней* функции (`Function`) аргументы обычно только входные, в наличии все виды объектов [1-7], но *видимыми* являются [1-4];
2. у *внешней* подпрограммы (`Subroutine`) аргументы могут быть входными, входными - выходными и выходными, и с ее именем [2] не связывают результатов;
3. *модульная* процедура (*Function, Subroutine*) аналогична внешней процедуре, но вложена в модуль и потому не является самостоятельной. В ПЕ, использующей модуль, можно вызывать модульные процедуры, но нельзя вызывать их внутренние процедуры;
4. *главная* программа (*Program*) может быть только внешней, не имеет аргументов [3] и с ее именем [2] не связывают результатов;

внутренняя процедура не самостоятельна – она вложена во внешнюю или в модульную программу, импортируя её обозначения. Саму её не компилируют и не вызывают извне.

Таблица 35.

Видимыми объектами `внешней_программы` являются [1-7].

```
Заголовок имя_внешней_программы (аргументы) ! уровень 1
 [1] use имя_модуля                    ! делает объекты модуля видимыми
 [2] описание имя_внешней_программы ! только для функции
 [3] описание аргументов                ! видимы из внутренних процедур
 [4] описание переменных                ! видимы из внутренних процедур
  действия над всеми видимыми объектами [1-4] внешней программы
  имя_внешней_программы = выражение  ! только для функции [2]
contains  ! уровень 2 - внутренние процедуры

    Заголовок имя_внутренней_процедуры (аргументы)
     [5] описание  имя_внутренней_процедуры  ! только для функции
     [6] описание аргументов
     [7] описание переменных
    действия над всеми (внутри и вне) видимыми объектами [1-7]
     имя_внутренней_процедуры=выражение ! для функции [5]
    end Заголовок имя_внутренней_процедуры
           . .  . .  пачка внутренних_процедур.. ..

end Заголовок имя_внешней_программы
```

6.2. Трёхуровневая структура модуля

Модуль, как трёхуровневая структура, может содержать (**contains)** модульные процедуры, которые, в свою очередь, могут содержать (**contains)** внутренние процедуры. Имена общедоступных **public**-объектов модуля - заимствуют его модульные и их внутренние процедуры. Эти имена также заимствуют все, кто использует этот модуль, как-то: другие модули, внешние программы и, транзитом, их внутренние процедуры. Атрибут **public** по умолчанию имеют все объекты модуля. Модульные процедуры также имеют атрибут **public** и их можно вызывать из используемого модуля. В то же время их внутренние процедуры недоступны в программной единице, использующей модуль. **Private**-объекты модуля доступны только модульным процедурам и их внутренним процедурам. Глобальный объект проекта можно однократно описать в модуле. Затем в проекте любой сложности, используя модуль, можно получить доступ к его объектам и модульным процедурам. Важно иерархическое описание объектов, которое реализуется через иерархию использования модулей.

Таблица 36. Видимыми объектами модуля являются [1-6]

Module *имя_модуля* **!** ***уровень 1***
[1] ***use*** имя_другого_модуля и его объекты
[2] описания производных типов данных
[3] **public** ! *описания общедоступных переменных (по умолчанию)*
[4] **private** ! *описания переменных, доступных только в модуле*
[5] описания интерфейсов
[6] описания списков Namelist
! *действий в модуле нет*
contains

процедура МодПроцИмя(*параметры*) **!** ***уровень 2,*** *модульная*
[7] *описание* функции МодПроцИмя ! *только для функции*
[8] *описания параметров* ! *видимы из внутренних процедур*
[9] описания переменных ! *видимы из внутренних процедур*
действия над всеми видимыми объектами [1-9]
МодПроцИмя = выражение ! *только для функции* [7]
contains

процедура ВнтПроцИмя*(параметры)* ! ***уровень 3****, внутренняя*
[10] *описание* функции ВнтПроцИмя ! *только для функции*
[11] *описания параметров*
[12] описания переменных
действия над всеми видимыми объектами [1-12]
ВнтПроцИмя = выражение ! *только для функции* [10]
End ***процедура*** ВнтПроцИмя
.. **..** *пачка внутренних процедур* **...**

End ***процедура*** МодПроцИмя

.. **..** *пачка модульных процедур* **....**

```
End module имя_модуля
```

6.3. Вызов процедур и использование модулей

Различают два вида *процедур*:

- `Function` – *процедура-функция*, вызываемая в выражении, например, *p=имя_функции(аргументы)*, аргументы только входные с атрибутом `intent(in)`; способы определения функции разнообразны – встроенные, операторные, внутренние, внешние, модульные, чистые, однако вызов любой из них аналогичен вызову простейшей встроенной функции *max* z = (x+y) / `max`(x,y);
- `Subroutine` – *процедура-подпрограмма*, вызываемая оператором вызова `call` *имя(аргументы)*, аргументы входные – `intent(in)`, выходные – `intent(out)`, изменяющиеся – `intent(inout)`.

Неисполняемый оператор использования модуля `use` *имя_модуля* пишут сразу после заголовка ПЕ. Любая ПЕ, как программа, так и другой модуль может использовать модуль, присоединяя общедоступные объекты с атрибутом `public` и делая доступными модульные процедуры.

Передача данных через аргументы процедуры.

Объекты вызывающей программы в скобках после имени вызываемой процедуры называют *фактическими* аргументами. Объекты вызываемой программы в скобках после имени в заголовке процедуры, называют *формальными* аргументами. Количество фактических и формальных аргументов должно совпадать. Попарно в порядке следования фактические и формальные аргументы должны соответствовать по типу, назначению и форме. Совпадения имен сопоставляемых фактических и формальных аргументов не требуется, но и не запрещается.

Память под формальные аргументы, в том числе и под массивы, в процедуре не резервируется: из вызывающей программы передается адрес объекта, с которым работает процедура.

Возвращаемое значение функции связано с её именем и по форме может быть скаляром, вектором, матрицей, многомерным массивом, описываемым атрибутом *dimension*.

На аргументы процедур в связи с их назначением накладывается ряд ограничений:

1. формальные аргументы могут быть только переменными и массивами;
2. фактические аргументы, соответствующие входным формальным аргументам с атрибутом `intent(in)`, могут быть:
   - переменными, константами, выражениями,
   - массивами, секциями массивов, элементами массивов,
   - именами внешних или встроенных функций;
3. фактические аргументы, соответствующие формальным выходным аргументам с атрибутом `intent(out)` и изменяющимся аргументам с атрибутом `intent(inout)`, могут быть только переменными, элементами массивов, массивами, секциями массивов.

Оператор `Interface`

В вызывающей ПЕ, а лучше в используемом модуле, для внешней процедуры пишут оператор `interface`, который описывает правила её вызова. Для модульных и внутренних процедур интерфейсы считаются явными и их описывать не надо. Оператор *interface* описывает:

- вид процедуры `Function` или `Subroutine`;
- имена и порядок следования формальных аргументов;
- формальный аргумент пишут с атрибутом `intent` – назначение, имеющим одно из трёх значений: `intent(in)`, `intent(out)`, `intent(inout)`.

Оператор `interface` – это самостоятельный составной оператор описания с локализованными внутри описаниями формальных аргументов

```
Interface
        описание процедуры и её формальных аргументов
        .. ..
        описание процедуры и её формальных аргументов
end Interface
```

Особенностью оператора `Interface` является то, что вложенные в него описания процедур и их аргументов локализованы в каждой из процедур и независимы друг от друга и от описаний той программной единицы, где написан оператор интерфейса. До Фортрана-90 оператора интерфейса не было. Спрашивается, надо ли его описывать? – Ответ не односложный:

1. для встроенных, внутренних и модульных процедур интерфейс считается явным, и его описывать не требуется;
2. интерфейс внешних процедур описывать рекомендуется - это помогает исключить ошибки вызова; рекомендуется собрать интерфейсы в модуле, а не дублировать их в вызывающих программах;
3. интерфейс нужно описывать или он должен быть явным:
   – для процедур с необязательными и ключевыми аргументами,
   – для функций с атрибутом `dimension`,
   – для чистых (`pure`) функций и подпрограмм,
   – в случае *перегрузки* типов, формы, операций и присваиваний.

Операторы `End, Stop, Return`

К оператору `End` применимы следующие правила:

1. после оператора `End` рекомендуется повторять заголовок ПЕ
   ```
   Subroutine name(аргументы)
      Операторы
   End Subroutine name
   ```
2. `End` – это последний оператор ПЕ, не только конец текста, но и нормальное завершение программы, что означает:
   – в главной программе – закрытие файлов, освобождение памяти и возврат в ОС,
   – в процедуре – освобождение динамической памяти и возврат в вызывающую программу.

   Оператор `Stop` по действию похож на `End` главной программы, только может встретиться в любом месте проекта – это, скорее всего, прекращение приложения из-за ошибки, на консоли – *stop program terminated.*

Оператор `Return` по действию похож на `End` процедуры. Смысл оператора `Return` в следующем: он может появляться в любом месте любой из процедур – это всегда досрочное завершение активной процедуры.

6.4. Обмен данными в проекте

Программные единицы (ПЕ) проекта получают данные и формируют результаты. Помимо передачи данных через аргументы в Фортране-90 появились два новых способа доступа к объектам носителя данных по именам:

– вложенные процедуры импортируют объекты внешней ПЕ;
– объекты модуля общедоступны там, где модуль используют (`use`) и модуль является носителем данных, объекты модуля по умолчанию являются общедоступными (атрибут `public`), а локальные объекты модуля должны быть помечены атрибутом *private.*

Присоединение данных носителя

Носителем данных может быть:

– внешняя программа для своих внутренних процедур;

- модуль для своих модульных процедур;
- модульная процедура для своих внутренних процедур.

Пример: попадают ли пять точек из *Array* в круг?

```
Program Host
 integer :: i;  real :: R=5    ! радиус
 real,public,dimension(1:2)::x0=(/1,1/)! координаты центра
  real, dimension(1:2,1:5):: Array
  Open(1,File='In.txt'); Open(2,File='Out.txt')
  write(2,3) R,x0;  read(1,*) Array
  do  i = 1,5
   if(InCircle(Array(:,i))&
       write(2,2)Array(:,i),'внутри круга'
  enddo
2 format('Точка с координатами [',f6.2,' , ',f6.2,  '] - ', a)
3 format('Окружность радиуса ', f6.2,  &
                 ' с координатами центра [', f6.2, ',', f6.2, ']')
   contains
  logical function InCircle(X) ! модульная процедура: X в круге?
  real,intent(in),dimension(1:2)::X  ! X - координаты точки
   ~~real:: R=1 ! радиус~~
    InCircle = sum((X-x0)**2) < R*R
 end function InCircle
end Program Host
```

Функция *InCircle* является внутренней, а главная программа `Host` манипулирует с объектами так:

- `x0` присоединяется к числу видимых объектов внутренней функции *InCircle* из носителя данных `Host`;
- секция *Array(:,i)* поступает через аргументы, как вектор *X* в функции;
- *R*=5 главной программы и ~~`real:: R=1`~~ внутренней функции не входят в противоречие – это просто две одноименные переменные; чтобы в *InCircle* не было *R* надо *обязательно* удалить ~~`real:: R=1`~~.

Присоединение данных модуля

Модуль – централизованное хранилище общедоступных объектов, описываемых в проекте однократно. Оператор «`use` *имя_модуля*» размещают вслед за заголовком программной единицы – и тогда появляется доступ к объектам модуля: константам, переменным, массивам (атрибут `public` по умолчанию), интерфейсам процедур, спискам `Namelist`, производным типам данных, модульным процедурам, обслуживающим объекты модуля.

Пример: попадают ли 5 точек в круг радиуса 5 с центром в точке (1,2)?

```
Module Mo
 real,dimension(1:2)::p0=(/1,2/) ! x,y - координаты центра
  real,public:: R=1     ! инициализация радиуса
```

```
   real,dimension(1:2,1:5)::P   ! 5 точек (x,y)
    Namelist /R_P0/R,P0
contains
  function InCircle()
   logical,dimension(1:5)::InCircle
   InCircle=(p(1,:)-p0(1))**2+(p(2,:)-p0(2))**2 < R**2
  end function InCircle
end Module Mo
Program With_module
 use Mo
 R=5    ! изменение радиуса
 Open(1,File='In.txt'); Open(2,File='Out.txt')
 write(2,R_P0); read(1,*) P
 write(2,22) R,p0;   write(2,222) P
     22 format('Радиус:',f6.1,' центр:[',f6.1,',',f6.1,']')
                  222 format(  'Точки:'/(2f6.1) )
  write(2,*) InCircle() ! логический вектор для точек
  if( all( InCircle()) ) write(2,*)'все точки внутри круга'
end Program With_module
```

Результат выполнения программы:

```
&R_P0
 R       =   5.000000    ,
 P0      =   1.000000    ,   2.000000
 /
Радиус:   5.0  центр:[   1.0,   2.0]
Точки:
   1.0   2.0
   3.0   4.0
   0.0   1.0
   2.0   3.0
  40.0  50.0
    T T T T F
```

Радиус `real::R=1` указан в модуле. В главной программе написать `real::R=5` нельзя – это вызовет ошибку, как повторное описание объекта *R*. Можно только перевычислить *R=5*.

Оператор `use` может исключить ошибки, вызванные коллизией имен:

- с помощью оператора `use` *Mo,* `only`*: x0, InCircle* можно ограничить список используемых имен модуля;
- заменив имя *R* на несуществующее имя *R_null (use Mo, R_null=>R)*, можем писать оператор `real::R` в главной программе.

Атрибут `private` – локальный, приписанный объекту модуля, запретит доступ к объекту вне модуля. Проектируя модуль, следует избегать общедоступных (`public`) тривиальных имен вроде *i, j, k, a* и т.п.

7. Встроенные функции Фортрана

Встроенные функции являются неотъемлемой частью языка Фортран и их имена входят в число ключевых слов. В учебниках на русском языке наиболее полное описание функций имеется в [2]. У каждой функции – уникальное имя, типы аргументов. У многих функций имеются необязательные и ключевые аргументы. Многие функции заимствуют у одного из своих аргументов форму (такие функции называют элементными) и тип возвращаемого значения (такие функции называют *родовыми*). В Фортране-90 числовые функции стали *элементными и родовыми*: если аргумент – массив, то и результат – *конформный* массив того же типа, полученный применением функции поэлементно. Ряд математических функций, как *max, min*… являются *родовыми*, заимствующими тип у своего аргумента. В зависимости от выполняемых действий функции принято разбивать на группы. При работе с тригонометрическими функциями величину угла задают в радианах или в градусах, добавляя окончание *d* к имени функции.

7.1. Числовые функции

Таблица 37.

Функция	Возвращаемое значение
max(a1,a2,..)	Максимум из значений аргументов: **max**(-8.3,6.0,2.0)=6.0; *все* аргументы должны быть либо *все* целые, либо *все* вещественные.
min(a1,a2,..)	Минимум из значений аргументов: **min**(-8.0,6.0,2.0) = –8.0; *все* аргументы должны быть либо целые, либо вещественные
abs(a)	|*a*| – абсолютная величина аргумента.
mod(a,p)	Остаток от деления первого аргумента на второй; **mod**(1, 2)=1; **mod**(18,2)=0
int(a)	Целая часть аргумента; **int**(-5.7) = –5; **int**(0.9)=0
nint(a)	Ближайшее к аргументу целое число (округление): **nint**(-5.7) = –6; **nint**(0.9)=1
sign(a,b)	Абсолютное значение первого аргумента со знаком второго: **sign**(1.0,-1.0E-25) = –1.0
sqrt(x)	Квадратный корень из аргумента (аргумент ≥ 0)
exp(x) **log**(x) **log10**(x)	Экспонента e^x натуральный логарифм ln*x* (*x*>0) десятичный логарифм lg*x* (*x*>0)
sin(x),**cos**(x) **tan**(x) **cotan**(x)	sin *x* cos *x* угол – в радианах, *вещественное* значение tg *x* ctg *x*
sind(x) **cosd**(x) **tand**(x) **cotand**(x)	Sin*x* угол – в градусах, *вещественное* значение Cos*x* Tg*x* ctg*x*
asin(x)	arcsin*x* (|*x*|<1.0). Результат – в радианах. (-π/2<*asin*(x)<π/2)
acos(x)	arccos*x* (|*x*|<1.0). Результат – в радианах. (0<*acos*(x)<π)
atan(x)	arctg*x*. Результат – в радианах. (π/2<*atan*(x)<π/2)
sinh(x) **cosh**(x) **tanh**(x)	sh*x*, ch*x*, th*x* - синус, косинус, тангенс гиперболический

7.2. Функции редукции массивов

Редукция в переводе с английского – «сокращение». Функции редукции *сокращают* количество измерений исходного массива так, что у результата либо 0 измерений (это скаляр) либо число измерений меньше на одно указанное. *Ключевые слова* аргументов функций:

1. `Array` – числовой массив;
2. `Dim` - номер сокращаемого измерения, возможны два варианта:
   – если аргумент `Dim` не указан, результат – скаляр (0 измерений);

– Dim указывает номер измерения, исчезающего у результата.

Для одномерного массива в обоих случаях в результате – скаляр;

3. Mask – свойство массива Array для участия в подведении итогов.

Каждый аргумент может быть:

– *позиционным* – на своем по порядку месте в списке аргументов функции и без ключевых слов «*array*=»,« mask=» или « dim=»;
– *ключевым* – аргумент задается с ключевыми словами «array=», «mask=», «dim=» независимо от номера в списке аргументов.

Важное требование: *Array* и *Mask* – конформные массивы. Для одномерных массивов это означает, что у них равное количество элементов. Часто *Mask* задают в виде логического выражения от числового массива, чтобы гарантировать конформность, например, для массива Array=Ar задают mask=*Ar>0* или для Array=Ar(1:3) – mask=Ar>0.

Редукция одномерных массивов

Все ответы в примерах, приведенных ниже, получены для одномерного вещественного массива:

```
Real,dimension(1:7):: Ar
```

Таблица 38.Вещественный массив для редукции одномерных массивов

Массив Ar=[**2.1, –0.7, 0.0, 17., –12., 0.4, –35.5**]						
Номера элементов	1	2	3	4	5	6	7

Итоговые функции с редукцией массивов Sum, Product, MinVal, MaxVal

Здесь и далее в описаниях функций аргументы, указанные в квадратных скобках, *необязательны* (могут отсутствовать при конкретном вызове). Так, для одномерного массива аргумент №2 dim можно опускать или указывать dim=1 –для одномерного массива результат редукции – всегда скаляр, как итог по всему массиву. Аргумент dim почти во всех примерах отсутствует, поскольку для одномерного массива он не влияет на форму результата. *Итоговые числовые функции* перенимают тип у массива:

```
Real,dimension(1:5):: Ar;  Real Value
Value = Sum(array[, dim][, mask])    ! сумма
Value = Product(array[,dim][,mask]) ! произведение
Value = MinVal(array[,dim] [,mask]) ! минимум
Value = MaxVal(array[, dim][,mask]) ! максимум
```

Примеры

1. Найти значение минимального положительного элемента массива *Ar*

```
Real Zmin
```

Два варианта задания аргументов при вызове функции *MinVal*:

Zmin=**MinVal**(*Ar,1,Ar>0*) – имена аргументов не пишут, поскольку все они указаны по порядку на своих местах;

Zmin=**MinVal**(Ar,mask=*Ar>0*) – пропущен аргумент №2 dim и из-за этого необходимо указать имя *mask* для аргумента №3, этого будем придерживаться в последующих примерах для одномерного массива.

В результате: Zmin = 0.4

2. Найти значение максимального элемента *Ar* из интервала [0.1, 17.6]

Real Zmax; Zmax=**MaxVal**(Ar, **mask**=0.1<=Ar**.and.** Ar<=17.6)

В результате: Zmax = 17.0

3. Найти сумму квадратов положительных элементов секции *Ar(1:7:3)*

Real SumQu; SumQu=**Sum**(Ar(1:7:3)**2, **mask**=Ar(1:7:3)>0)

В результате: SumQu = $Ar(1)^2 + Ar(4)^2 = 2.1^2 + 17^2 = 293.41$

4. Найти произведение модулей ненулевых элементов секции *Ar(3:7)*

Real Prod; Prod=**Product**(**abs**(Ar(3:7)),**mask**=Ar(3:7)/=0)

В результате:

Prod = Ar(4) * Ar(5) * Ar(6)* Ar(7) = 17.0*12.0*0.4*35.5 = 2896.8

Внимание, типичная ошибка, когда, используя секцию массива, забывают эту секцию указать для маски Mask, и тогда *size(***Array***)≠size(***Mask***)* :

– ошибка, которая выявится при компиляции

Su=**Sum**(**Array** =**Ar**(1:3),mask=Ar>0)

– Su=**Sum**(**Array**=Ar(1:N), **mask**=Ar>0) - *нет ошибок при компиляции*

– при *N*=7 ошибки действительно нет

– при неизвестном *N*< 7 ошибка есть

– правильно так: Su=**Sum**(**Array** =Ar(1:N), **mask**=Ar(1:N)>0)

Any, all– итоговые логические функции с редукцией формы массива

Итоговые логические функции возвращают логическое значение:

Any(mask) – результат «true», если в массиве есть элементы со свойством mask; иначе говоря, это логическое ИЛИ элементов маски;

All(mask) – результат «true», если все элементы массива обладают свойством mask; иначе говоря, это логическое И элементов маски.

Примеры

1. Есть ли в массиве Ar элементы, по модулю большие 30 ?

Logical L1; L1=**Any**(**abs**(Ar)>30)

В результате: *L1* = .true. так как |-35.5|>30

Другой пример

if(**Any**(**abs**(Ar)>30))**print** *,' Ar > 30 presents'

2. Все ли элементы с пятого по седьмой отрицательны?

Logical L2

L2 = **All**(Ar(5:7)<0) ! *В результате: L2=.false.*

Другой пример:

if(**.not.All**(Ar(5:7)<0))**print** *,'not all Ar(5:7)< 0'

***Итоговый счетчик* Count**

Возвращается целое число – количество элементов, со свойством *mask*
int=**count(***mask***)** – *сколько* элементов удовлетворяет *mask***.**

Пример

Найти количество элементов массива *Ar*, по модулю больших 11.
```
Integer K11;  K11 = Count(abs(Ar)>11)
```
В результате: K11=3 – три элемента: |Ar(4)|>11, |Ar(5)|>11, |Ar(7)|>11
Внимание:

- аргумент Mask задают в виде логического выражения от числового массива, в то время как самого числового массива нет среди аргументов функций all, any, count;
- аргумент *dim* почти во всех примерах отсутствует, поскольку для одномерного массива он не влияет на форму результата.

Положение экстремума – MaxLoc MinLoc

Функции MaxLoc, MinLoc этой группы являются справочными и не редуцируют исходный массив.
```
Integer,dimension(1:1):: Num
Num=MaxLoc(array[,mask])! Num(1)- номер максимального элемента
Num=MinLoc(array[,mask])! Num(1)-номер минимального элемента
```
Для одномерного массива функции MaxLoc, MinLoc возвращают целочисленный однокомпонентный вектор с номером минимального (MinLoc) или максимального (MaxLoc**)** элемента для массива Array = *Ar*.

Примеры

1. Найти номер максимального элемента одномерного массива *Ar*.
```
Integer,dimension(1:1):: Num; Integer N
Num=MaxLoc(Ar);  N = Num(1)
```

Можно к однокомпонентному вектору MaxLoc*(Ar)* применить любую из четырех *числовых* функций редукции, тогда получим скаляр:
```
N=Sum(MaxLoc(Ar))
```
В результате: N = 4 – для максимального элемента Ar(4) = 17.0

2. Найти номер минимального положительного элемента массива Ar.
```
Integer,dimension(1:1)::Num; Integer N
Num=MinLoc(Ar,mask=Ar>0); N=Num(1)
 или  N=Sum(MinLoc(Ar,mask=Ar>0))
 или  N=Sum(MinLoc(Ar,1,Ar>0))
```
В результате: N=6 для минимального положительного элемента Ar(6)=0.4

Редукция двумерных массивов

В примерах везде, где функции применяются к матрице array=*Matr,* целочисленный двухкомпонентный вектор *Num* используется для хранения местоположения экстремума: **Integer,dimension**(1:2)::Num
```
Real,dimension(1:2,1:5) :: Matr
```

Таблица 39.

Вещественный массив для функций редукции двумерных массивов

		1	*2*	*3*	*4*	*5*
Вещественный массив	1	**-11**	**-12.**	**-13.**	**-14.**	**-15.**
(матрица `Matr`*)*	2	**-21**	**-22.**	**-23.**	**-24.**	**0.**

Говоря о матрицах, прежде всего, обсудим аргумент `dim`, и условимся говорить о *суммировании*, имея в виду, что для других операций редукции все делается аналогично. Для двумерного массива аргумент `dim` может отсутствовать или принимать значения из {1,2}:

- нет `dim`: возвращаемое значение – скаляр, *сумма* по *всему* массиву;
- `dim` =1: возвращаемое значение – вектор-строка, полученная в результате *суммирования* строками;
- `dim` =2: возвращаемое значение – вектор-столбец, полученный в результате *суммирования* столбцами.

Числовые функции заимствуют тип результата у массива **`array`**:

```
Value = Sum(array[, dim][, mask])    ! сумма
Value = Product(array[,dim][,mask]) ! произведение
Value = MinVal(array[,dim] [,mask]) ! минимум
Value = MaxVal(array[, dim][,mask]) ! максимум
```

Примеры

1. В матрице *Matr* найти минимум среди элементов, больших -20
   `Real` `S;  S = MinVal(Matr, mask=Matr>-20)`
   Аргумент *dim* отсутствует. В результате число: S = -15.0
2. Найти max элементы в каждом столбце *Matr* (вектор-строка)
   `Real, dimension``(1:5) S;  S = MaxVal(Matr, dim=1)`
   В результате – вектор-строка: S(1:5)= [–11., –12., –13., –14., 0.]
3. Найти вектор-столбец с построчными суммами для элементов матрицы *Matr*, больших –22.5
   `Real,dimension``(1:2)S;S=Sum(Matr,dim=2,mask=Matr>-22.5)`
   В результате – вектор-столбец: S(1)=– 11. – 12. – 13. – 14. – 15.,
   S(2)=21. – 22. + 0. => S(1:2)= [– 65., – 43.]
4. Найти вектор-столбец с построчными произведениями для ненулевых элементов матрицы *Matr* с нечетными индексами столбцов**.**
   `Real, dimension``(1:2) S`
   `S=product(Matr(:,1:5:2),2,mask=Matr(:,1:5:2)/=0)`
   В результате – вектор-столбец: S(1) = (-11)*(-13)*(-15) = –2145.
   S(2)=2 (-21)*(-23) = 483.

Логические функции для двумерного исходного массива возвращают логическое значение или одномерный массив:

any(**mask**[,**dim**]) – результат «истина», если в массиве есть элементы со свойством *mask*, то есть логическое ИЛИ элементов маски;

all(**mask**[,**dim**]) – результат «истина», если все элементы со свойством *mask*, то есть логическое И элементов маски.

Аргумент Mask задают в виде логического выражения от числовой матрицы, в то время как самой числовой матрицы нет среди аргументов.

Примеры

1. Есть ли в матрице *Matr* элементы, по модулю превышающие 30?
   Logical L; L=**Any**(**abs**(Matr)>30)
   Аргумент *dim* отсутствует. Результат – скаляр, L = .FALSE.
2. По каждой строке матрицы *Matr*: есть ли элементы > (–1)?
   Logical, dimension(1:2) L; L=**Any**(Matr>-1, **dim**=2)
   Результат логический массив (вектор-столбец) L(1:2) =[.false.,.true.]
3. О каждом столбце матрицы *Matr:* все ли элементы отрицательны?
   Logical, dimension(1:5) L; L=**All**(Matr<0, **dim**=1)
   Результат – логический массив (вектор-строка)
   L(1:5)=[.true.,.true.,.true.,.true., .false.]

Итоговый счетчик возвращает целое число или целочисленный массив – количество элементов, удовлетворяющих условию mask

Count(**mask**[,**dim**])

Примеры

1. Подсчитать количество отрицательных элементов в матрице *Matr*.
   Integer K; K=**Count**(Matr<0)
   Аргумент dim отсутствует. В результате – число K=9
2. Пересчитать отрицательные элементы каждой строки матрицы *Matr*
   Integer,dimension(1:2)K; K= **Count**(Matr<0, **dim**=2)
   Результат – целочисленный массив (вектор-столбец) K(1:2)= [5, 4]

Положение экстремума **MinLoc MaxLoc**

Функции возвращают целочисленный двухкомпонентный вектор с номером строки и столбца, где расположен экстремум.

Num **=** **MaxLoc**(**array**[,**mask**]) ! *координаты максимума*

Num **=** **MinLoc**(**array**[,**mask**]) ! *координаты минимума*

Примеры

1. Найти координаты максимального элемента массива *Matr*
   Integer, dimension(1:2):: Num
   Num = **MaxLoc**(Matr)
   В результате: Num= [2, 5] для Matr(2,5) = 0.
2. Найти координаты максимального отрицательного элемента *Matr*
   Integer, dimension(1:2):: Num
   Num = **MaxLoc**(Matr, Matr<0)
   В результате: Num = [1, 1] для Matr(1, 1) = –11.
3. Найти номер max по модулю элемента *первой* строки массива *Matr*

```
Integer, dimension(1:1):: Num
    ! поиск максимума в первой строке (одно измерение)
Num = MaxLoc(abs(Matr(1,1:5))); N = Num(1)
```

В результате: Num(1) = 5; N = 5 для Matr(1, 5) = –15.

4. Найти номер min элемента > (-20) в *третьем* столбце массива *Matr*

```
Integer, dimension(1:1):: Num
    ! поиск минимума в третьем столбце (одно измерение)
Num=MinLoc(Matr(1:2,3),Matr(1:2,3)>-20));
N=Num(1)
```

В результате: Num=[1]; N = 1 для Matr(1, 3) = –13.

Внимание: Фортран-95 определяет результат и при наличии аргумента `dim`, которого не было в Фортран-90.

8. Справочные материалы

Редакторы простых текстов для набора программ

Традиционно текст программы, адресованный компилятору с Фортрана или Си, готовится редактором простых текстов (plain-текстов), т.е. текстов без рисунков, мультимедиа и прочих встроенных объектов. Редакторы:

- в составе Developer Studio, Visual Studio для Windows;
- редактор в составе FAR (кодировки Windows/ANSI и DOS/ASCII);
- многооконный MULTI-EDIT 8.0 для Windows и DOS;
- блокнот Windows;
- Geany под Linux и другие.

Инструменты – мышь и клавиатура. Объекты, с которыми работают редакторы – файлы на диске, окна на экране и текстовые блоки в окнах. При создании текста на Фортран-90 следует называть файл *имя.f90* (на Фортран-03 – *имя.f03*). Работая в среде Windows, пользуются редакторами, которые придерживаются кодировки ANSI, принятой в Windows.

Выбирая *кодировку* для набора текста программы, следует знать:

- в DOS используется только кодировка ASCII;
- в Windows для файлов используются обе кодировки ANSI и ASCII и только ASCII для общения с консолью.

Кодировки ASCII и ANSI для Windows в английской части совпадают, и сообщение на английском языке `write(*,*) 'waiting for data'` выведется, независимо от кодировки. В русской части кодировки различаются, поэтому сообщение на русском языке `write(*,*) 'жду данных'` выведется успешно, если его набирали в кодировке ASCII, как положено для вывода на консоль в Windows. Если русский текст подготовлен в среде Windows, на экране увидим *абракадабру* (кодировка ANSI – не для консоли).

Рекомендации:

1. текст, выводимый на экран, писать по-английски, например, в диалоге вместо «*Введите*» писать «*Input*», либо пользоваться редактором *far для смены кодировок*;

2. текст результирующего файла с русскими словами печатать для отчета прямо в *DevStudio* или переносить в *word* транзитом через *far*.

Традиции оформления текста на Фортране

В Фортране придерживаются следующих правил оформления имен:

– прописные и строчные буквы не различаются компилятором;
– кириллицу используют в оформлении результатов и в комментариях;
– пробелы вне ключевых слов и имен, а также пустые строки используются свободно по усмотрению программиста.

В Фортране операторы, конструкции и блоки привязаны к строкам текста. Каждая программная единица, каждый составной оператор, каждый блок составного оператора пишутся с новой строки – после нажатия клавиши <Enter>. Чтобы короткие простые операторы не удлиняли текст по вертикали, их разделяют невидимым концом строки «;».

Традиционно вложенные (подчиненные) конструкции выделяются уступами. Кроме того, управляющие конструкции разрешается именовать, что упрощает их восприятие в сложном тексте, например,

```
C1: DO  ! имя обрамляет конструкцию – пишется дважды в начале «C1:»
    do
      if( условие 1 ) cycle c1  ! к следующей итерации цикла  C1
      if( условие 2 ) exit c1 !  выход сразу из двух циклов
    enddo
ENDDO C1  ! имя обрамляет конструкцию – в конце «C1»
```

На один и тот же формат по метке могут ссылаться несколько операторов вывода: **write**(*,23) ‘начало’; **write**(*,23) ‘конец’; 23 **format***(2x,a)* В современном Фортране использование меток практически ограничивается форматами и ссылками на них. Оператор перехода по метке «`goto` *метка*» считается устаревшим, хотя из языка он не исключен. Не рекомендуется в новых программах использовать любые разновидности `goto`.

8.1. Пакет Agrapher для построения графиков

Назначение пакета:

– вывод на экран графиков функций одной переменной
– печать графиков функций одной переменной
– экспорт графиков в виде *jpg*-файла изображения
– построение сглаживающих прямых и кривых.

Пакет прост, его можно применять при выполнении лабораторных, курсовых и дипломных работ:

– *Agrapher*, *http://www.alentum.com* – российский, для обучения распространяется в России свободно; установите его с русским меню.
– вызов пакета *Agrapher* в Windows – через ярлык или меню *программ* в кнопке *Пуск*. Имеется панель инструментов и контекстное меню.

Освоение пакета *Agrapher* облегчается обратной связью через картинку.

Более давний *Grapher* - американский пакет, без русификации; даёт возможность исчерпывающей настройки графика, имеет презентабельный вид.

Семь способов задания графиков в *Agrapher*:

1. по формуле Y(X);
2. по формуле X(Y);
3. параметрическое задание X(t),Y(t), например, для окружности;
4. по таблице X,Y, задаваемой в самом пакете *Agrapher;*
5. по системе неравенств, задаваемой логической формулой, как в работах по логике – лучше показывать со штриховкой (>0);
6. по импортируемой таблице – меню *Файл* | *импорт таблицы* (расширение *txt* по умолчанию);
7. как результат сглаживания (прямой, полиномом, сплайном).

В процессе открытия или после него:

- к показу можно привлечь как линии, так и точки;
- можно сменить цвет, размер и форму вывода точек;
- можно сменить цвет, толщину и стиль линий.

Настройки графика можно сохранить как используемые по умолчанию. Есть сглаживание, численное интегрирование и аналитическое дифференцирование. *Agrapher* предложит свой вариант масштабирования, редко когда удачный, поэтому есть ряд ручных инструментов для этого:

- выбрав в меню «свойства графика», можно изменить его настройки, в пункте «построение» можно сменить интервалы по X и Y;
- после изменения масштаба следует подправить дробные *засечки* и *метки* на осях в «*свойствах*» графика, лучше это сделать в конце
- график масштабируется *парными стрелками* панели инструментов;
- *одинарные стрелки* панели вызывают смещение графиков;
- утопленная кнопка выделения фрагмента позволит увеличить обведенный нажатой мышкой прямоугольный фрагмент, как на картах.

Таблица 40.

Справочник простейших функций для заданий

Уравнения прямой		
Прямая параллельна оси *X*, пересекает ось *Y* в точке (0,*b*).	$y=b$	
Прямая параллельна оси *Y*, пересекает ось *X* в точке (*a*,0).	$x=a$	
Прямая проходит через точку (x_1,y_1) под углом α к оси *X*.	$(y-y_1) = k(x-x_1)$	$k = \operatorname{tg}\alpha$
Прямая пересекает ось *Y* в точке (0,*b*) под углом α к оси *X*.	$y = kx+b$	$0 \le \alpha < \pi$
Прямая проходит через (0,0) под углом α к оси *X*.	$y = kx$	$\alpha \ne \pi/2$
Прямая проходит через две точки (x_1,y_1) и (x_2,y_2), где $x_1 \ne x_2$, $y_1 \ne y_2$.	$\frac{y-y_1}{y_2-y_1} = \frac{x-x_1}{x_2-x_1}$	
Прямая пересекает ось *X* в точке (*a*,0), ось *Y* в точке (0,*b*) (уравнение прямой в отрезках).	$\frac{x}{a}+\frac{y}{b}=1$ или $y = -\frac{b}{a}x+b$	

Длина отрезка, соединяющего две точки (x_1,y_1) и (x_2,y_2)	$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$
Уравнения окружности	
Уравнение окружности радиуса R с центром (x_0,y_0).	$(x-x_0)^2+(y-y_0)^2=R^2$
Параметрические уравнения окружности радиуса R с центром в точке (x_0,y_0). Аргумент α: $0 \le \alpha < 2\pi$	$\begin{cases} x = x_0 + R\cos\alpha \\ y = y_0 + R\sin\alpha \end{cases}$
Уравнения эллипса, оси которого совпадают с осями координат	
Каноническое уравнение эллипса с длинами полуосей a и b и центром в точке (x_0,y_0).	$\frac{(x-x_0)^2}{a^2}+\frac{(y-y_0)^2}{b^2}=1$
Параметрические уравнения эллипса с длинами полуосей a и b и центром в точке (x_0,y_0) $0 \le \alpha \le 2\pi$	$\begin{cases} x = x_0 + a\cos\alpha \\ y = y_0 + b\sin\alpha \end{cases}$
Уравнения параболы	
Уравнение параболы, ось которой параллельна оси X, а вершина – в точке (x_0,y_0).	$x = \pm (y-y_0)^2 + x_0$
Уравнение параболы, ось которой параллельна оси Y, а вершина – в точке (x_0,y_0).	$y = \pm (x-x_0)^2 + y_0$

8.2. Типичные ошибки в арифметических выражениях

Таблица 41. Типичные ошибки в написании арифметических выражений

Формула	В программе		Пояснения
	Правильно	**Неправильно**	
x^{-y}	**`x**(-y)`**	`x**-y` – *синтаксическая ошибка.* Знаки операций записывать подряд не разрешается.	Запись формулы требует дополнительных скобок.
$\cos^2 x$	**`cos(x)**2`** После имени функции – аргумент в скобках, а не операция	`cos**2(x)` *Синтаксическая ошибка.* *cos* воспринимается компилятором как переменная, а не как функция	Сначала вычисляется функция, затем результат возводится в степень
$2x+b$	**`2*x+b`**	`2x+b` *синтаксическая ошибка.*	знак операции опускать нельзя
$\frac{a}{bc}$	**`a/(b*c)`**	`a/bc` Пропущен знак умножения – вместо переменных b и c используется переменная bc	Сначала должно вычисляться произведение в знаменателе (требуются дополнительные скобки).
$\cos x^2$	**`cos(x**2)`**	`cos(x)**2` соответствует формуле $\cos^2 x$.	Косинус от квадрата x

Формула	В программе		Пояснения
	Правильно	Неправильно	
$\frac{a}{(b+c)d}$	**a/((b+c)*d)**	a/(b+c)*d соответствует формуле $\frac{a}{b+c}\cdot d$	Запись формулы требует дополнительных скобок.
X^{2Y}	**X**(2*y)**	x**2*y соответствует формуле X^2Y	Запись формулы требует дополнительных скобок.
Дробь $\frac{4}{7}$	**4./7.** или **4./7** или **4./7.** для получения значения 0.5714	4/7 Операнды – целые числа, результат – целая часть частного, для 4/7 это 0	Для получения вещественного результата хотя бы один из операндов должен быть вещественным
$\sqrt[3]{x}$	**x**(1./3.)** при x>0. Функции кубический корень нет	x**(1/3) Результат деления 1/3 равен 0 и общий ответ x^0=1.	Возведение в *вещественную* степень $\frac{1}{3}$ выполняется по формуле $e^{\frac{1}{3}\cdot\ln x}$
$\sqrt[3]{x}$	**x**(1./3.)** при x>0. Функции кубический корень нет	x**1./3. соответствует формуле $\frac{x^1}{3}$	Запись формулы требует дополнительных скобок для дробного показателя степени.
$\sqrt[3]{-27}$	**-27**(1./3.)** Сначала дробь, потом степень и перемена знака	(-27)**(1./3.) В вещественную степень нельзя возводить отрицательное число	$\sqrt[3]{-27} = -\sqrt[3]{27}$

8.3. Типичные недочеты и ошибки в работе с массивами

На массивы приходится наибольшее число нововведений и оригинальных возможностей современного Фортрана, поэтому в помощь начинающим методичка дополнена данным разделом. Массив – не самый простой объект, поэтому требуются практические разъяснения.

Таблица 42. Статический массив

Нехорошо, неправильно	Правильно – статический массив	Пояснения
Real *X(10)* **Real,& dimension(1:10)::Y**	**Real, dimension** (1:10) **::*X*,Y**	Конформные массивы дают единым списком
Integer::N=10 !переменная **Real,dimension**(1:N)::X *Неверно*: размерность-*переменная*	**Integer,parameter::**N=10 ! размерность *N – константа* **Real,dimension**(1:N)::X	Размерность статического массива должна быть *константой*.

Real X(10) **Real,dimension**(1:10)::**Y** нехорошо, размерность *10* надо менять *во многих местах*	**Integer,parameter::**N=10 ! размерность *N – константа* **Real,dimension**(1:N)::X**,Y**	Меняем только N – перекомпилируем
Real,dimension(1:10)::X .. *x* (i)= **Y**(i) Ошибка не при компиляции, а при компоновке **:** **Y** – unresolved external	**Real, dimension**(1:10)::***X,***Y** *x*(i) = y(i) *x,*Y – оба массивы	не описали **y** – воспринимается не как массив, а как неизвестная функция от одного аргумента **y(i)**
Real,dimension(1:4)::***X,***Y **Y**(i) = *x* (i) Ошибка при компиляции	**Real,dimension**(1:4)::***X,***Y ***i =4***; y (i) = *x* (i) *x,***Y** – оба массивы	не описали **y** – воспринимается как ошибка
Real,dimension(1:4)::***X,***Y x(i)=y**(i)** ! индекс ***i = ?***	**Real,dimension**(1:4)::***X,***Y **do** *i* =1,4 *x* (i)= y(i) **enndo**	Ошибка при выполнении: защита записи – индекс не определен
Real,dimension(1:4)::***X,***Y x(i)=y**(i)** ! индекс ***i = ?***	**Real,dimension**(1:4)::***X,***Y *X* = Y	Ошибка при выполнении: – индекс ***i*** не определен

Таблица 43.

Динамический массив

Нехорошо, неправильно	**Правильно**-динамический массив	**Пояснения**
Real,dimension(:)::*X* Неправильно – надо **allocatable**	**Real,allocatable,&** **dimension(:)**::*X*	нужен атрибут **allocatable** размерность свободна ! форма (1:4) – *неправильно*
Real,allocatable,& **dimension(1:4)**::X		
Real,dimension(1:4)::**X**=1 **real,allocatable,**& **dimension**(:)::Y ~~**!allocate(y(1:size(x)))**~~ Y = X	**Real,dimension**(1:4)::***X,***Y**=1 **real,allocatable,** & **dimension**(:)::y **allocate(y(1:size(x)))** **y = x**	забыли **allocate:** адрес *y* не определен
Real,dimension(1:4)::***X***=4 **Real,dimension**(1:8)::z=8 **Real,allocatable,&** **dimension(:)**::Y **allocate**(y(1:**size**(x))) **Y=X** ~~**! deallocate**(y)~~ **allocate(y(1:size(z)))** **y = z**	**real,dimension**(1:4)::X=4 **Real,dimension**(1:8)::Z=8 **real, allocatable,&** **dimension**(:)::y **allocate**(y(1:**size**(x))) **Y = X** **deallocate**(y) **allocate**(y(1:**size**(z))) y = z	забыли **deallocate** В результате повторный **allocate** не выполняется
Real,allocatable,& **dimension(:)::X** **Integer** :: N=8 **allocate**(*X*(1:N))	**Real,dimension(1:8)**::X	ошибки нет, но нет смысла строить динамический массив постоянного объема

Таблица 44. Выражения и присваивания с массивами

Нехорошо, неправильно	Правильно	Пояснения
Real,dimension(1:4)::P **Real,dimension**(1:8)::Q ***P*= Q*; Q= *P*** плохо: массивы разной длины	**Real,dimension**(1:4)::***X*,Y** ***X*=(/ 3.1, 3.7, -15. /)** **Y= *X**2+3*X+1***	*Массив = Выражение* и присваивают, и строят выражение только из конформных массивов, исключение – скаляры как *sin*(*z*) или 1 в выражении
Real,dimension(1:4)::X *Real*,dimension(1:4,1:3)::M **M= *X; X*= M** плохо: массивы с разным числом измерений не присваивают	**Real**:: *z=0.72* **Real,dimension**(1:4)::X **Real**,**dimension**(1:4,1:3)::M**=1** ***X*=sin(z) + 1; M(:,1)=X;** **M(:,2)=2**	
Real,**dimension**(1:4,1:3) :: M **Real,dimension**(1:3, 1:4)**::K** M=K !**shape**(M)≠**shape**(K)	**Real,dimension**(1:4,1:3)::M,K **Real,dimension** (1:4,-1:1)::R K = M ; M= R **shape**(M)=**shape**(K)=**shape**(R)	конформные массивы - это массивы одинаковой формы [1:4,1:3], нумерация может быть разная

Таблица 45. Встроенные функции для работы с массивами

Нехорошо, неправильно	Правильно	Пояснения
Real,dimension(1:8)::X **Print** *, **sum**(X, **X**>0)	**Real**,**dimension**(1:8)::X **Print** *, sum(X, **mask=**X>0)	Mask – 3-ий аргумент: -надо указать имя mask -задать после 2-го аргумента
	Print *, sum(X, **1,** X>0)	
Real,**dimension**(1:8)::X **Print** *,count(~~Array=~~X,X>0)	**Real**,**dimension**(1:8)::X **Print** *, *count*(**X**>0)	*У функции* **count** нет аргумента **array**
Real,**dimension**(1:8)::X **Print***,sum(X(1:4),mask=**X**>0)	**Real**,**dimension**(1:8)::X **Print***,sum(X(1:4),mask=X(1:4)>0)	На ошибку **Size(X(1:4))≠ Size(X)** укажет компилятор
Real,**dimension**(1:8)::X **Real** g*; **integer** N* g = sum(**X(1:N)**, mask=**X**>0) правильно только при N=8	**Real**,**dimension**(1:8)::X **Real** g*;* **integer** :: *N=4* g =**sum**(X(1:N), **mask**=X(1:N)>0) правильно при любом *N≤8*	**Size**(X(1:N))≠ **Size***(X)* компилятор не укажет на ошибку при неизвестном *N*
Integer N **Real**,**dimension**(1:8)::A N= **MinLoc**(A) **MinLoc** - возвращает вектор, который не конформен скаляру *N*	**Real**,**dimension**(1:8)::A **Integer** N N=**sum**(**MinLoc**(A))	Найти номер максимального элемента из массива A. Также правильно: **N=sum(MinLoc**(A))
	Real,**dimension**(1:8)::A **Integer** N; **Integer &, dimension(1:1)::**Num Num=**MinLoc**(A); N=Num(1)	

Таблица 46.
Ввод - вывод массивов

Нехорошо, неправильно	Правильно	Пояснения	
Real,**dimension**(1:4,1:3)::M **read**(1,*)**M** *Нехорошо*: читают, как хранится в памяти – по столбцам	**Integer** *i* **Real**,**dimension**(1:4,1:3)::M **read**(1,*) (M(i,1:3),i=1,4) Надо читать по строкам	Матрица ошибочно читается по столбцам, а обычно читают по строкам.	
Real,**dimension**(1:4,1:3)::M **write** *(1,7)* ***M*** 7 **format**(3*f5.1*)	**integer** *i ;* **Real**, & **dimension**(1:4,1:3)*::M* **write** (1,7) *(M(i,:),i=1,4)* 7 **format** (3 *f5.1*) вывод изменен – по строкам	Матрица выводится по столбцам, а надо выводить по строкам	
Real,**dimension**(1:4,1:3)::M **write***(1,7)(M(i,1:3),i=1,4)* 7 **format** (**4** *f5.1*) Ошибочно: **4** –число строк	**Real**,**dimension***(1:4,1:3)::M* **write** *(1,7) (M(i,1:3),i=1,4)* 7 **format** (***3*** *f5.1*) Надо **3** – число столбцов	1-й индекс – строка *2-ой индекс - столбец* Перепутали числа: столбцов 3, а строк 4.	
Real,**dimension***(1:3,1:3)::K* **read**(1,*) *K* ! чтение транспонирует *K* *7* **format***(3 f5.1)* **write** *(2,7) K*	**Real**,**dimension**(1:3,1:3)*:: K* **read** *(1,7) (K (i,1:3),i=1,3)* **write** *(2,7) (K (i,1:3),i=1,3)* *7* **format***(3f5.1)*	Матрица K хранится по столбцам. При неправильном вводе и выводе – создается *иллюзия* правильности..	
Вывести левую половину матрицы (2 из 4 колонок) **Real**,**dimension**(1:2,1:4)::A **write***(2,7)*(A(i,1:2), i=1,2) *7* **format***('левая')****2f5.1****)* ! перед каждой строкой чисел – заголовок *левая*	Вывести левую **половину** матрицы (2 из 4 колонок) **Real**,**dimension**(1:2,1:4) :: A **write** *(2,7)*(A(i,1:2), i=1,2) *7* **format***('левая')* ***2(f5.1)*** *)*	Неправильно *левая* *4.0 4.0* *левая* *4.0 4.0*	Правильно *левая* *4.0 4.0* *4.0 4.0* *4.0 4.0*
Вывести **левую** половину матрицы (2 из 4 колонок) **Integer** *:: N=2* **Real**,**dimension**(1:2,1:4)::A **write***(2,7)*(A(i,1:2), i=1,2) 7 **format**('левая'/ & (<N>f5.1)) Ошибка компилятора FPS40	Вывести **левую** половину матрицы (2 из 4 колонок) **Integer***:: N* **Real**,**dimension**(1:2,1:4) ::A *N=2;* **write** *(2,7)*(A(i,1:2), i=1,2) *7* **format** *('левая' / <N>****(f5.1)*** *)* Только так работает в *FPS40*	Неправильно *левая* *4.0* *4.0* *4.0* *4.0*	Правильно *левая* *4.0 4.0* *4.0 4.0*

Таблица 47.

Описание массивов в процедурах

Нехорошо, неправильно	Правильно	Пояснения
Program *имя* **Real**,**dimension(:)** :: X Неправильно: в главной: размерность неизвестна, а надо **allocatable** или константу	**Function** имя(X) **real,intent(in),&** **dimension(:)::X** **Integer N** **N=size(X)** В процедуре отложенное (*deferred*) определение размерности. *N* – лучше измерить при помощи встроенной функции *N*=**size**(*X*)	Память под массив-аргумент распределена в вызывающей программе. В процедуре массив не статический, не динамический – он перенимает адрес и размерность у фактического аргумента..
Function *имя(N, X)* **Integer,intent(in)::***N* **Real,dimension(1:N) ::** *X* нехорошо, размерность *N* – величина, зависящая от *X*		
Function *имя(X)* **Real**,**dimension(:)** :: X для аргумента *X* нет ***intent(in)***, но сообщения нет	**Pure Function** *имя(X)* **real,intent(in),&** **dimension(:)::X** есть *pure* – будет сообщение	**intent** – атрибут аргумента процедуры
Program *имя* **Integer**::N **Real**,**dimension(1:N)** :: X Размерность статического массива – только константа	**Function** имя(N) **Integer,intent(in)::N** **Real**,**dimension(1:N)** :: X Размерность – *переменная, только в процедуре*	автоматический динамический массив *размещается* автоматически *при входе*, *освобождается – при выходе* из процедуры
Program unallocated **Real**,**dimension**(1:4)::X=1 **real,allocatable,** & **dimension**(:)::y **open**(1,file='su.txt') ~~**allocate**(y(1:size(x)))~~ **call** s(x,y) **write**(1,*) 'y=', y ..	**allocate**(y(1:**size**(x)).. **contains** **subroutine s (x, y)** **real,intent(in),&** **dimension(:)::X** **real**,**intent**(**out**),& **dimension**(:)::Y **y=x** **End subroutine s** **EndProgram** unallocated	Процедура правильная, а в главной забыли **allocate** В результате видим y= Вместо правильного y= 1. 1. 1. 1. при наличии **allocate**
Program unallocated **Real**,**dimension**(1:4)::X=1 **real,allocatable,** & **dimension**(:)::y **open**(1,file='su.txt') **! ~~allocate~~(y)** **call** s(x,y) **write**(1,*) 'y=',y **contains** ..	**allocate(y(1:size(x)))** **.. contains** **subroutine s(x,y)** **real,intent(in),&** **dimension(:)::X** **real,intent(out), &** **dimension**(1:**size(x)**)::y Y=X **End subroutine** s **endProgram unallocated**	Процедура правильная, а в главной забыли **allocate.** В результате массив y имеет неопределенный адрес.
Function si(X) **Real :: si** **real,intent(in),&** **dimension(:)::X** si=**sin**(X)	**Function** si(X) **real,dimension(:) ::** si **real,intent(in),&** **dimension(:)::X** **si=sin**(X)	массивоподобной функции *забыли* дать атрибут массива – **dimension**

8.4. Перевод ключевых слов, операторов и терминов Фортрана

Allocatable – динамический массив ↓
allocate – разместить в памяти
deallocate – освободить память
Array – массив, *Array*(1:7:2) – секция
Assignment – присваивание
Associate[F03] - связать с..
BackSpace – назад на 1 запись в файле
Call *SubrName* – вызов процедуры
Case и **CaseDefault** – это случаи в ↓
Select Case – перечисление случаев
Character – строковый тип, базовый
Close – закрыть файл
Common – общий блок памяти – устарел
Compiling – компиляция
Complex – комплексный тип, базовый
Contains–далее вложенные процедуры
Declarations – объявления
Descriptors **i f e l a g** дескрипторы данных
Cycle – на следующий проход цикла
Dimension – размерность *массива*
Double Precision - двойная точность
Do...**EndDo** 3 цикла: *бесконечный*
Do i =1,9,2 ... **End Do** *по переменной* i
Do While(условие) – *по условию*
Exit – выход из цикла
End – конец конструкции, как **enddo**
Enum [F03] – перечислитель, ООП
Equivalence – синонимы по описанию
Expression **-** выражение
Exponentials – возведение в степень
extends[F03] расширяемые типы, ООП
External Intrinsic имя процедуры
Forall [F95] – параллельный цикл
Format - формат ввода-вывода
Statement Function - операторная функция
Function – процедура-функция
GoTo – переход, *устарели* все его виды
If(условие)**then** оператор 0,1,2.. блоков
Else или **Elseif**()**then**-«иначе» для
End If двух- и много- блочного IF
Implicit None – всё описывать
Initializing-инициализация **real**::x=5.2
Data /.. /– инициализация, устаревшая
Inquire – справка о файле
Integer – целый тип, базовый
Intent(in)|(out)|(inout) –аргумент вх|вых
Interface – интерфейс процедуры..
Labels - метки в диапазоне 1:99999
Logical – логический тип, базовый
Module – модуль
Namelist – список для ввода-вывода
Nullify – отцепить указатель
Open – открыть файл
Optional – необязательный аргумент
Operator – операция *рус.* , не путайте
Parameter – *константа* с именем
Pause – пауза – устарела
Pointer – указатель
Print – вывод на консоль (экран)
Procedure – процедуры в интерфейсе
Program – главная программа
Private – локальный объект модуля
Public – доступный объект модуля
Pure – чистая процедура
Quadruple – число четверно́й то́чности
Read – чтение из файла или с консоли
Real – вещественный тип, базовый
Return – возврат из процедуры
Rewind – к началу файла
Save – хранить не в стеке
Single Precision – одинарная точность
Statement – оператор языка
Stop – конец работы приложения
Subroutine – процедура-подпрограмма
Target – на кого указывает **pointer**
.true. .false. .and. .or.
.not. .eqv. .neqv. – логика
Type поле.. **endType** -производный тип
Type(имя)структура.. структура %поле
Union..endUnion **map**-контейнер
Map -группа полей-синонимов структуры
Use – использовать модуль
Where – параллельный IF
Write – записать в файл, на консоль
[..] или (/ .. /) – конструктор массива
& *Continuation Lines* – перенос строк
! *Comment* – комментарий
«;» или <*Enter*> – конец строки в *Fortran*

9. Литература

1. Звягин В.Ф., Яньшина Н.А., Голыничев В.Н. Практикум по современному Фортрану в курсе Информатики. - СПб.: СПбГУ ИТМО, 2010 - 134 с.
2. Алгазин С.Д., Кондратьев В.В. Программирование на Visual Fortran – М.: Диалог МИФИ, 2008 - 472 с.
3. Артёмов И.Л. FORTRAN: основы программирования. - М.: Диалог МИФИ, 2007 - 304 с.
4. Горелик А.М. Программирование на современном Фортране. М.: - Финансы и статистика, 2006. 352 с.
5. Рыжиков Ю.И. Современный Фортран. Спб, Корона принт, 2004 - 288с.
6. Немнюгин С.А., Стесик О.Л. Современный Фортран. Спб, БХВ, 2003 - 496с.
7. Бартеньев О.В. ФОРТРАН для студентов. М: Диалог МИФИ, 1999 - 400 с.
8. Голыничев В.Н., Звягин В.Ф., Фрейман И.А., Щупак Ю.А., Яньшина Н.А. Практикум по информатике. Санкт-Петербург: - Санкт-Петербургский государственный институт точной механики и оптики (Технический университет), 2001 - 94с.
9. Рыжиков Ю.И. Программирование на Фортране Power Station для инженеров – Спб, КОРОНА принт, 1999 - 160с.
10. Программирование на Фортране 77: Пер. с англ./ Дж. Ашкрофт, Р.Элдридж, Р.Полсон, Г.Уилсон. - М.: Радио и связь, 1990 - 272 с.
11. ФОРТРАН 90. Международный стандарт. Пер. с англ. – М.: Финансы и статистика, 1998 - 416 с.
12. Мак-Кракен Д., Дорн У. Численные методы и программирование на Фортране.– 2-е изд.: Пер. с англ. – М.: Мир, 1977 – 584 с.