**ЛАБОРАТОРНЫЙ ПРАКТИКУМ**

**«Информатика и программирование»**

**Часть вторая – решение задач на Фортране.**

**Введение.**

Учиться программированию лучше всего на решении конкретных задач. Данный раздел практикума начинается с рассмотрения простейших основ программирования – решения задач и упражнений, необходимых начинающему программисту для дальнейшего написания достаточно сложных алгоритмов. Студенты, имеющих навыки написания программ на языке высокого уровня, могут начинать выполнение лабораторного практикума с решения задач, связанных с построением достаточно сложных логических конструкций.

Ниже представлены ***обязательные******требования*** к написанию и оформлению программ, практические советы, а также некоторые сведения по Фортрану (в самом кратком и минимальном объёме) необходимые для первых шагов программирования. Подробное описание языка дано в соответствующей литературе [1] и на сайте accel.ru.

Практикум базируется на работе в лаборатории кафедры «Электрофизические установки» НИЯУ МИФИ (сервер accel.ru).

**Требования к написанию и оформлению программ.**

Практическую работу в лаборатории студенту следует начинать с построения файловой структуры выполняемых лабораторных работ в своём личном каталоге. Все необходимые знания получены в первой части практикума - работе в операционной среде UNIX.

1. Каждая лабораторная работа должны быть оформлена в отдельной папке.
2. В этой папке, в конечной стадии работы, должны находиться только окончательные варианты исходных, исполняемых и текстовых файлов (с результатами расчётов). *Все файлы пробных и промежуточных вариантов должны быть удалены.*
3. Имена исходных и исполняемых файлов данной лабораторной работы должны совпадать и, по возможности, отражать её смысловое содержание. *Пример: Лабораторная работа по рассмотрению циклов*.

Папка с именем «CYCLE». Поскольку циклы бывают трёх типов, то для каждого типа создать свою папку, например, «CYCLE/ CYCLE\_F» и т.п. Имена исходных файлов могут быть, например, «cycle\_f.f95», а исполняемых файлов просто «cycle\_f». Текстовый файл (с результатами расчётов) - «cycle\_f.txt».

1. Результаты расчётов должны быть оформлены в удобочитаемом виде с необходимыми текстовыми пояснениями и выведены в соответствующий текстовый файл.

*Практический совет:* Программа должна быть легко читаема, а для этого:

1. Должны быть даны в начале программы краткие, но содержательные комментарии, определяющие суть задачи и производимые действия.
2. Все используемые данные должны быть явно объявлены и сгруппированы по типам.
3. Особое внимание именам переменных и массивов. Они должны отражать физическую или математическую суть переменной.

*Пример:* Напряженность электрического поля – *E*. Ускорение свободного падения – *g*. Для математических выражений нужно использовать символы близкие к написанию математических формул. Если есть выражение , то нужно использовать имена близкие по смыслу - и т.п. Для индексов нужно использовать принятые в математике символы - и их вариации - и т.п. Желательно не употреблять длинных и безликих имён, что затрудняет чтение текста.

1. Для записи управляющих конструкций использовать *правило рельефа* –операторы, вложенные внутри конструкции записываются правее операторов образующих эту конструкцию.

*<операторы\_0>*

*do i=0,n*

*<операторы\_1>*

*if <условие> then*

*<операторы\_2>*

*else*

*<операторв\_3>*

*endif*

*<операторы\_1\_1>*

*enddo*

**Программа.**

Прежде чем приступать к написанию программы нужно чётко представлять себе смысловую и техническую стороны программирования. Смысловое содержание программы – это её физическая, математическая или логическая суть, т.е. нужно хорошо осмыслить конечный результат расчётов и исходные предпосылки необходимые для решения данной конкретной задачи. Техническая сторона проблемы заключается в ясном представлении процесса создания исполняемого файла, т.е. понимания этапов преобразования исходного текста, написанного на языке программирования высокого уровня, в исполняемый файл, преобразованный в текст на языке понятном компьютеру.

В самом общем виде процесс создания исполняемого файла представлен на рис.1.

Сама программа – это исходный текст, записанный в текстовом редакторе. Всё остальное относится к работе операционной системы, каждая часть которой отвечает за свой этап вычислительного процесса.

1. Программа (исходный модуль) готовится с помощью текстового редактора в виде текстового файла, который подается на вход транслятора.
2. На стадии трансляции исходный модуль преобразуется в промежуточную форму объектного модуля. Процесс компиляции включает в себя проверку правильности написания программы с точки зрения синтаксиса языка, выявления несоответствий типов и структур переменных, функций и процедур. Последний этап трансляции – генерация объектного кода. Объектный модуль – это текст программы, написанный на машинном языке.
3. Исполнительный модуль реализуется редактором связей, построителем задач, компоновщиком, основной задачей которых является объединение объектных и загрузочных модулей в единый загрузочный модуль с последующей записью в библиотеку пользователя или файл.

Современные интегрированные системы программирования позволяют удобно переходить от одного этапа к другому. Они содержат в себе текстовый редактор, компилятор, компоновщик, встроенный отладчик и другие, удобные для пользователя средства написания и отладки программы.

**ЭВМ:** информация в строго регламентированном виде

***ОС:*** ввод, доступ, хранение, обработка, вывод информации

***Язык*:** Все правила написания программы.

Текстовый редактор

Транслятор (компилятор)

**ЗАДАЧА**

Исходный

текст

Исходный модуль

Объектный модуль

Компоновщик

Редактор связей

Загрузочный модуль

Загрузчик

Абсолютный модуль

Выполнение

Результат

Исходные данные

Библиотека системных программ

Библиотека программ пользователя

Выполнение

Рис.1

**Исходный модуль.**

Структура исходного текста программы подчиняется строгим правилам используемого языка программирования. Любую программу можно разделить на две части - *описания и тело программы.*

1. Часть описаний содержит определения имён используемых переменных, массивов, записей (тип, размер и т.д.).
2. Тело программы – последовательность исполняемых операторов.

*Примечание:* нельзя допускать, чтобы операторы описания «вклинивались» между исполняемыми операторами тела программы!

Стратегия написания программы заключается в выборе оптимального варианта, т.е. не нужно пытаться написать программу как «на все случаи жизни», так и программу «однодневку». Нужно выбирать разумную середину. Это, прежде всего, касается анализа входных данных.

* Программа должна быть, как минимум, рассчитана на расчёт вариантов, отличающихся друг от друга только численными значениями некоторых входных данных. Следовательно, такие данные должны считываться с клавиатуры (или из внешнего файла).
* Другие данные, которые являются одинаковыми для всех вариантов, могут быть заданы в теле программы.
* Такой подход к составлению программы исключает необходимость перетранслировать её при задании новых числовых вариантов.
* Не перегружайте программу, по возможности, сложными логическими конструкциями, что может привести к трудно определяемым ошибкам и неправильным вычислениям (особенно для начинающего программиста).

Результаты расчётов – выходные данные – должны быть записаны в удобочитаемом виде с соответствующими краткими текстовыми пояснениями. Выходная информация может быть двух видов промежуточная и конечная. Промежуточная информация необходима как для отладки программы, так и для изучения работы тех или иных алгоритмов, реализованных в данной программе. Например, при рассмотрении алгоритмов сортировки можно рассмотреть процесс перестановки элементов сортируемого массива по отдельным шагам алгоритма.

**Типы данных.**

Из всех типов данных рассмотрим четыре стандартных типа.

Целый - INTEGER

Вещественный - REAL

Символьный - CHARACTER

Логический - LOGICAL

Форма объявлений следующая:

INTEGER список переменных, разделённых запятыми

REAL список переменных, разделённых запятыми

CHARACTER

LOGICAL

**Переменные и Массивы.**

*Переменная* – поименованный объект какого-либо типа.

*Массив* – поименованный набор конечного числа объектов одного типа. Массивы могут быть одномерные и многомерные. Объявление типа и размерности массивов производится в разделе описаний программы.

Переменная или массив становятся *определёнными* после того, как они получают значения в результате присвоения или выполнения ввода.

Операторы объявления типов размещаются в начале программы в разделе объявлений.

*Пример:*

INTEGER i, j, i1, ind, mas(10), mas1(5,10) - и т.п

REAL a, r, tt(25), par(5,25) - и т.п

CHARACTER(n), где n - длина символьной строки

*Пример:* character(10) sim, где sim=’ text ‘ – текст из 10 символов, включая пробелы. Тестовая переменная заключается в апострофы.

LOGICAL может принять только два значения **.true.** (правда) или **.false.** (ложь)

Для описания массивов могут быть использованы различные формы записи.

* INTEGER mas(10), mas1(5,10)

REAL tt(25), par(5,25)

*или*

* INTEGER mas, mas1

REAL tt, par

DIMENSION mas(10), mas1(5,10), tt(25), par(5,25)

Форма записи mas(10), par(5,25) означает, что элементы массивов могут иметь индексs от 1 до 10 и от 1 до 5 (первый параметр), от 1 до 25 (второй параметр).

Форма записи mas(-5:10), par(0:5,5:25) означает, что элементы массивов могут иметь индексы от -5 до 10 и от 0 до 5 (первый параметр), от 5 до 25 (второй параметр). Форма записи с разделителем **(:)** определяет нижнюю и верхнюю границы изменения индекса.

*Примечание.* В Фортране действует правило «*по умолчанию*» - переменные, имена которых начинаются с букв i, j, k, l , m, n – являются переменными целого типа. Все остальные имена – вещественного типа. Тип переменных можно изменить, задавая их в явном виде в операторах объявления INTEGER или REAL.

**Ввод и вывод данных.**

Рассмотрим действия операторов READ и WRITE, PRINT – операторов ввода (чтения) и вывода (записи, печати). Общая форма записи:

READ(\*,\*) - список ввода !<ввод с клавиатуры>

READ\*, - список ввода !<ввод с клавиатуры>

WRITE(\*,\*) - список вывода !<вывод на экран>

PRINT\*, - список вывода !<вывод на экран>

Список ввода - перечень вводимых величин.

Список вывода - перечень выводимых величин.

Общая форма операторов следующая -

READ (<идентификатор устройства>, < идентификатор формата>)

WRITE (<идентификатор устройства>, < идентификатор формата>)

PRINT\*, < идентификатор формата>)

* Идентификатор устройства задаётся выражением целого типа (числом)
* Идентификатор устройства – при значениях \*, 0 и 5 присоединяется клавиатура (ввод с клавиатуры).
* При других значениях идентификатора устройства – связь с внешним файлом, который инициируется в программе оператором OPEN в начале программы.
* Идентификатор формата - при значениях \* происходит бесформатный (по умолчанию) вывод информации.
* При других значениях идентификатора формата (метки формата - целого числа) происходит форматный вывод информации.
* В позиции <идентификатор формата> может быть записан сам формат.

**Форматный ввод и вывод данных.**

Перевод данных из внутреннего представления в текстовое задаётся дескрипторами преобразований. В таблице 1. приведены некоторые дескрипторы.

Таблица 1.

|  |  |  |
| --- | --- | --- |
| Дескриптор | Тип аргумента | Внешнее представление |
| Iw[.m] | Целый | Целое число |
| Fw[.m] | Вещественный | Вещественное число без экспоненты |
| Ew.d[Ee] | Вещественный | Вещественное число с экспонентой |
| A[w] | Символьный | Строка символов |
| Lw | Логический | **.**true**.** и **.**false**.** |

где I, F, E, A, L – коды форматов

w – размер поля, отведённого для ввода/вывода данного;

d – количество цифр после десятичной точки;

m - количество цифр в поле вывода целого числа;

e- количество цифр в поле вывода для экспоненты числа.

*Пример: бесформатныйй и форматный вывод.*

*Дано* : x=0.1

y=1.5

ind=125

1. Бесформатный вывод

write(\*,\*) x, y, ind

0.1000000 1.5000000 125

2. Бесформатный вывод с поясняющим текстом

write(\*,\*) 'x=',x,'y=',y,'ind=',ind

x= 0.1000000y= 1.5000000ind= 125

3. Бесформатный вывод с поясняющим текстом (с пробелами в тексте)

write(\*,\*) 'x=',x,' y=',y,' ind=',ind

x= 0.1000000 y= 1.5000000 ind= 125

1. Форматный вывод с поясняющим текстом (с пробелами в формате ***nx***),

n - количество пробелов

10 format(2x,a2,f5.2, 2x,a2,f5.2, 2x,a4,i3)

write(2(,10) 'x=',x,'y=',y,'ind=',ind

x= 0.10 y= 1.50 ind=125

1. Форматный вывод с поясняющим текстом (во внешний файл)

Открытие внешнего файла для записи

open(2,file='d:\study\lab\_work\writ.txt', status='unknown')

…..

10 format(2x,a2,f5.2, 2x,a2,f5.2, 2x,a4,i3)

Вывод во внешний файл

write(2,10) 'x=',x,'y=',y,'ind=',ind

x= 0.10 y= 1.50 ind=125

Сравните насколько варианты 4 и 5 (форматные выводы на экран и во внешний файл) более читаемы и информативны.

**Вывод массивов.**

Результаты выводятся во внешний файл, например, открытый как

open(3,file='d:\study\lab\_work\w\_mas.txt',status='unknown')

*Вывод одномерного статического массива*.

1. Нужно вывести созданный одномерный массив, определённый как real mas0\_x(0:50), но заполненный только 11 числами, во внешний файл, располагая по 5 элементов массива в строку. Вывод проводится по формату 10, где 5 – повторитель формата (т.е. количество выводимых в строку чисел по формату (2x,f6.4), 2*x* – количество пробелов между числами.

write(3,\*)' mas0\_x ' ! Имя массива

write(3,10)(mas0\_x(i),i=0,10)

10 format(5(2x,f6.4))

В результате вывода получаем

mas0\_x

0.1000 0.1500 0.2000 0.2500 0.3000

0.3500 0.4000 0.4500 0.5000 0.5500

0.6000

2. Нужно вывести тот же массив в столбец (в строке одно число). Вывод проводится по формату 11,

write(3,\*)' mas0\_x ' ! Имя массива

write(3,11)(mas0\_x(i),i=0,n)

11 format(2x,f6.4)

В результате вывода получаем

mas0\_x

0.1000

0.1500

0.2000

0.2500

0.3000

0.3500

0.4000

0.4500

0.5000

0.5500

0.6000

3. Нужно вывести таблицу, состоящую из 3 одномерных массивов. (например, массив аргумента , массивов и ). Созданные одномерные массивы, определённые как real mas1\_x(0:50),mas1\_y(0:50),mas1\_z(0:50), но заполненные только 11 числами, выводятся во внешний файл и располагаются соответственно в три столбца. Вывод проводится по формату 20, где 3 – повторитель формата (т.е. количество выводимых в строку чисел по формату (2x,f6.4).

write(3,\*)' mas1\_x ',' mas1\_y ',' mas1\_z '

write(3,20)(mas1\_x(i),mas1\_y(i),mas1\_z(i),i=0,n)

20 format(3(2x,f6.4))

В результате вывода получаем

mas1\_x mas1\_y mas1\_z

0.1000 0.0998 0.9950

0.1500 0.1494 0.9888

0.2000 0.1987 0.9801

0.2500 0.2474 0.9689

0.3000 0.2955 0.9553

0.3500 0.3429 0.9394

0.4000 0.3894 0.9211

0.4500 0.4350 0.9004

0.5000 0.4794 0.8776

0.5500 0.5227 0.8525

0.6000 0.5646 0.8253

*Вывод двухмерного статического массива*.

Нужно вывести созданный двухмерный массив (матрицу ), определённый как real mas2\_x(10,5) во внешний файл, располагая по 5 элементов матрицы в строку. Вывод проводится по формату 30, где 5 – повторитель формата (т.е. количество выводимых в строку чисел по формату (2x,f6.4). *m* =5, *n* =10.

write(3,\*) ' mas2\_x '

write(3,30) ((mas2\_x(i,j),j=0,m),i=1,n)

30 format(5(2x,f6.4)

В результате вывода получаем

mas2\_x

0.1500 0.2000 0.2500 0.3000 0.3500

0.2000 0.3000 0.4000 0.5000 0.6000

0.2500 0.4000 0.5500 0.7000 0.8500

0.3000 0.5000 0.7000 0.9000 1.1000

0.3500 0.6000 0.8500 1.1000 1.3500

0.4000 0.7000 1.0000 1.3000 1.6000

0.4500 0.8000 1.1500 1.5000 1.8500

0.5000 0.9000 1.3000 1.7000 2.1000

0.5500 1.0000 1.4500 1.9000 2.3500

0.6000 1.1000 1.6000 2.1000 2.6000

Во всех выводах на печать используется скрытая форма организации циклов

write (3,11) (mas0\_x(i),i=0,n)

write(3,10) (mas0\_x(i),i=0,n)

write(3,20) (mas1\_x(i),mas1\_y(i),mas1\_z(i),i=0,n)

write(3,30) ((mas2\_x(i,j),j=0,m),i=1,n)

**Внешние файлы.**

*Внешний файл* – файл, существующий в среде, внешней по отношению к выполняемой программе. Внешние файлы должны быть открыты оператором OPEN.

Оператор OPEN создаёт устройство ввода/вывода с номером ***n*** и подсоединяет к нему внешний файл ***file*** (полное имя файла с путём доступа). Оператор имеет целый ряд спецификаторов, рассмотрим только три из них.

OPEN(n, file=’<*имя\_файла*>’, STATUS=’<*статус*>’).

*n* – устройство внешнего файла, к которому подсоединяется файл - <*имя\_файла>.*

<*имя\_файла>* - имя файла с расширением ***.txt*** с путём доступа (от корня).

<*статус*> - ‘unknown’ . Если файл существует, то он открывается,

если нет – создаётся.

*Пример:*

open(2,file='d:\study\lab\_work\writ.txt', status='unknown')

**Внешние процедуры.**

В фортране определены два типа процедур – *подпрограммы* (*subroutine*) и *функции (function)*. Функция вызывается из выражения обращением к ней по имени с заменой формальных параметров фактическими параметрами и возвращает результат, который используется в этом выражении, т.е. её результат можно записать в одну переменную. Подпрограмма вызывается оператором CALL <имя подпрограммы с заменой формальных параметров фактическими параметрами> и возвращает в вызывающую программу значения нескольких переменных (поэтому в отличие от функции не может использоваться в выражениях). Структура процедур такая же, как и у основной программы только перед концом процедуры *(end)* ставится оператор (*return)* возвращения в вызывающую программу.

*Пример:* В вызывающей программе существует массив – массив значений аргумента функции . Создаём функцию f=sin(x), где *x* – формальный параметр.

*real function f(x)*

*real x*

*f=sin(x)*

*return z*

*end*

В вызывающей программе запишем обращение к функции – замена формального параметра *x* на фактический параметр .

**Генератор случайных чисел.**

При решении различных научно-технических и учебных задач часто используется последовательность случайных чисел. В вычислительной технике эта последовательность реализуется генераторами случайных чисел (ГСЧ) – специального вида программами. В действительности имеем дело с последовательностью *псевдослучайных чисел*, так как для работы программы необходимо тем или иным способом задать стартовое число. В зависимости от версии Фортрана процедура формирования последовательности псевдослучайных чисел (ПСЧ) реализуется различными подпрограммами. В Gfortran’е получение СЧ реализуется следующим образом.

Подпрограмма RANDOM\_NUMBER(*ran*) возвращает псевдослучайное число (или массив ПСЧ) из равномерно распределённого в интервале . Тип *ran* – вещественный. Стартовая точка (затравка) для генератора случайных чисел устанавливается подпрограммой

RANDOM\_SEED(*[size],[put],[get]*).

Возможны два режима работы генератора:

* При очередном запуске программы выдаётся каждый раз новая последовательность случайных чисел.
* При очередном запуске программы выдаётся одна и та же последовательность случайных чисел.

Параметры подпрограммы RANDOM\_SEED:

*Size –* стандартное целое число равное размеру *n* создаваемого процессором затравочного массива.

*Put –* стандартный целый массив, используемый процессором для изменения затравки.

*Get -* стандартный целый массив, в который заносятся текущие значения затравки.

При вызове RANDOM\_SEED задаётся не более одного параметра.

*Пример:* Формирование двухмерных массивов вещественных и целых чисел в различных диапазонах значений.

real bb(25,10) - описание массивов СЧ.

integer aa(25,10), cc(25,10)

! --------------старт генератора--------------------------------------

Integer sv, ag(100), ap(100) - параметры для запуска ГСЧ.

sv=50

call random\_seed(size=sv)

call random\_seed(put=ap)

call random\_seed(get=ag)

! ---------------------------------------------------------

do j=1,10 - заполнение массивов размерностью .

do i=1,10

call random\_number(hav) – вещественное число в диапазоне от 0 до 1.

Это основной элемент – получение СЧ ***(hav***) в зависимости от стартового значения. Использование его в циклах позволяет заполнять массивы любой размерности.

*! генерация целых чисел в диапазоне от 0 до 10.*

aa(i,j)=anint(hav\*10) – умножение на 10 и отбрасывание дробной части (функция *anint)*. Заполнение в цикле массива aa(i,j)

*! генерация вещественных чисел в диапазоне от 0 до 1.*

bb(i,j)=hav - Заполнение в цикле массива bb(i,j)

*! генерация целых чисел в диапазоне от -1 до 1.*

cc(i,j)=anint(hav\*2)-1 ) – умножение на 2, отбрасывание дробной части (функция *anint)* и сдвиг на 1. Заполнение в цикле массива cc(i,j)

enddo

enddo

!----вывод сформированных массивов по заданному формату-------------

write(\*,10) ((aa(i,j),i=1,10),j=1,10)

write(\*,11) ((bb(i,j),i=1,10),j=1,10)

write(\*,10) ((cc(i,j),i=1,10),j=1,10)

10 format(2x,10i8)

11 format(2x,10f10.8)

read(\*,\*)

end

*Результат действий*

6 5 4 5 8 10 7 6 5 9

6 1 6 2 7 2 0 1 2 7

3 4 9 5 8 4 5 1 2 0

10 0 3 5 3 3 4 9 4 1

8 5 7 2 9 8 2 8 8 1

3 5 10 9 0 1 10 4 1 1

6 8 5 10 9 0 0 9 8 9

10 6 5 4 8 4 3 3 6 9

7 7 3 1 2 5 9 0 3 5

7 8 10 4 3 4 3 9 5 7

0.195 0.183 0.924 0.027 0.206 0.972 0.143 0.930 0.663 0.505

0.767 0.366 0.342 0.137 0.287 0.903 0.971 0.220 0.854 0.003

0.835 0.273 0.930 0.779 0.151 0.671 0.250 0.316 0.487 0.096

0.625 0.229 0.540 0.134 0.846 0.819 0.456 0.007 0.021 0.028

0.768 0.466 0.448 0.626 0.578 0.020 0.035 0.353 0.351 0.195

0.489 0.638 0.226 0.515 0.552 0.533 0.337 0.112 0.303 0.413

0.111 0.168 0.007 0.807 0.625 0.482 0.269 0.653 0.620 0.143

0.311 0.479 0.706 0.898 0.343 0.183 0.116 0.077 0.374 0.500

0.102 0.771 0.684 0.509 0.029 0.054 0.043 0.499 0.301 0.728

0.393 0.085 0.304 0.987 0.766 0.698 0.722 0.891 0.464 0.611

0 1 1 1 -1 -1 -1 1 -1 0

0 0 0 -1 0 0 -1 0 1 0

1 1 -1 1 -1 0 0 1 0 0

1 1 -1 0 0 0 -1 -1 1 0

1 0 0 0 0 0 -1 1 1 -1

0 0 1 0 0 0 -1 -1 -1 1

0 -1 0 1 -1 0 -1 0 -1 1

0 0 0 1 -1 1 0 1 0 0

1 -1 1 -1 -1 0 0 0 0 1

0 0 -1 0 1 1 -1 0 -1

Для генерации при очередном запуске программы одной и той же последовательности случайных чисел обращения к подпрограмме

call random\_seed(put=ap)

call random\_seed(get=ag)

нужно исключить.

**Итог**

Конечным результатом, полученным на основании выше изложенного материала, является корректно построенная структура программы.

Комментарии о содержании работы

Описание входных и выходных данных

Real ОПИСАНИЕ

Integer

Dimension

Подключение внешних файлов

Ввод начальных данных

Write - подсказка

Read – ввод

Программный код алгоритма задачи ТЕЛО ПРОГРАММЫ

Вывод конечных результатов

Write - вывод

Format – форматы вывода

End – конец программы

**Упражнения и задания.**

1. **Первая программа.**

**Упражнение 1. Процедура создания исполняемого файла.**

**Задание 1.**

Первый вопрос начинающего программиста – как ввести информацию с клавиатуры и как вывести информацию на экран.

В редакторе напишите программу, в которой используется только процедура вывода текста на экран - *write* (*текст*)*.* Выводимый текст - (‘Hello, computer’) - заключён в одиночные кавычки.

*program first*

*write(\*,\*) ‘Hello, computer’*

*end*

Исходный текст программы транслируется и в случае отсутствия синтаксических ошибок создаётся исполняемый файл, который посылается на исполнение. Результат – «Hello, computer», программа выполнена, но на экране результат работы может быть и не отражен.

*Порядок действий:*

* Наберите в командной строке команду-

gfortran <имя\_исходного\_файла> -o <имя\_исполняемого\_файла >

если трансляция прошла без ошибок, то создаётся исполняемый файл

* запуск исполняемого файла ***./имя\_исполняемого\_файла*** (без пробелов – точка, слеш, имя файла).

**Задание 2.**

Добавьте в предыдущую программу оператор *read.*

*program first*

*write(\*,\*) ‘Hello, computer’*

***read****(\*,\*)*

*end*

Теперь при выполнении программы на экране появится надпись

***Hello, computer.***

Чтобы вернуться к исходному тексту в окне редактора, нажмите клавишу *Enter*.

Разница между этими программами заключается в том, что при последовательном выполнении операторов программы, после оператора *write* - вывода на экран текста (‘Hello, computer’) - следует оператор ввода данных *read* без параметров (т.е. фиктивное чтение). Программа ожидает ввода данных. Нажимая клавишу *Enter****,*** осуществляется операция фиктивного чтения данных и возвращение к исходному тексту программы.

**Задание 3.**

Тексты (или отдельные символы) можно выводить на экран, используя переменные, объявленные как строки или символы – *character(\*количество символов)*. Тексты могут быть написаны на любом языке (например, на русском – «Привет, компьютер»).

*Character\*25 abc {Описание строковой переменной }*

*abc= ‘Привет, компьютер’ {Присвоение строковой переменной abc значения }*

*write(\*,\*) abc {Вывод значения строковой переменной на экран}*

*read(\*,\*)*

*end*

Результат выполнения этой программы такой же, как и предшествующей программе - «*Привет, компьютер*».

1. **Основные элементы программирования.**

**Лабораторная работа № 1. Линейные программы.**

Линейная программа – программа, в которой идет последовательное выполнение операторов без повторений пройденных участков программы и разветвлений.

**Задание 1.**

Напишите программу, в которой производятся простейшие арифметические вычисления, и результаты выведите на экран. Для этого нужно:

1. Объявить несколько переменных целого (*integer*) и вещественного (*real*) типов.

2. Ряд переменных использовать как входные параметры - ввод конкретных значений через процедуру ввода данных - *read(список переменных).*

*Примечание:* не забудьте написать «подсказку» с именами входных переменных- *read(список переменных).*

3. Записать простейшие арифметические действия.

4. Ряд переменных использовать как выходные параметры - вывод конкретных значений через процедуру вывода данных - *write* *(список переменных).*

**Задание 2.**

Задайте вещественное число *x*. Используя только операции умножения, сложения и вычитания, произведите вычисления приведённых выражений за указанное количество операций. Приведённые выражения нужно соответствующим образом преобразовать (в т.ч. с использование скобочных конструкций).

*  (не более 4-х умножений, не более 4-х сложений и вычитаний)
*  (не более 8-и операций)
*  (не более 8-и операций)

**Задание 3.**

Задайте значения параметров переменными типа *real*. Вычислите следующие выражения. Для тестирования задайте значения *a, b, c, d* в диапазоне 1  4 в алгебраических выражениях и  - в тригонометрических выражениях.

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 

**Лабораторная работа № 2. Разветвляющиеся вычислительные процессы:** **условный оператор *if*.**

Для того, чтобы в зависимости от конкретных значений данных, обеспечить выполнение двух (или более) различных последовательностей операторов, используются оператор ветвления программы – *if***.** Ветвление программы осуществляется в зависимости от вычисляемого значения логического условного выражения - «правда» или «ложь».

*if < условие > then*

*< оператор1 >* {Выполняется, если условие = ’правда’}

*else*

*<оператор2>*  {Выполняется, если условие = ’ложь’}

*endif*

Вначале вычисляется <*условие>*. Если оно имеет значение – «правда», то выполняются  *< оператор1 >*  и управление передается на конец оператора *if.* Если оно имеет значение – «ложь», то выполняются  *< оператор2 >.* Часть *else* оператора *if* может быть опущена и тогда, в случае значения «ложь», весь оператор *if* как бы пропускается.

*< Оператор1 >* и *< оператор2 >* - операторы любого типа, в том числе, и условные. В последнем случае возникает вложенная логическая конструкция. На некоторых уровнях вложенности может отсутствовать часть *else.* Чтобы не было путаницы уровней вложенности, существует правило – часть *else* соответствует ближайшей к ней «сверху» части *then* условного оператора *if*.



Структурная схема условного оператора

Логическое условное выражение может быть представлено простыми операциями отношения – равно, не равно, больше, меньше, меньше-равно, больше-равно – *a.eq.d, c.ne.s, m.gt.n, r.lt.p, k.le.z, f.ge.q*.

Сложные условные выражения (т.е. проверка нескольких соотношений переменных совместно) строятся с использованием логических операций – *и, или, не*.

Например, (*a.eq.d*) .*and.* (*c.ne.s*)*,* (*m.gt.n*). *or.* (*r.le.p*), (*s.ne.k*)*.*

При записи сложных условных выражений необходимо помнить, что логические операции имеют более высокий приоритет исполнения, чем операции отношения. Поэтому, чтобы не нарушалась логика конструкции, нужно элементы, содержащие операции отношения, заключать в круглые скобки. При построении сложных логических конструкций с использованием элементов логики – *и, или, не*, анализ структуры можно разложить на простейшие операции сравнения двух входных сигналов.



Схема действия логических операторов

*Совет начинающим программистам*. По возможности старайтесь избегать очень длинных и сложных логических конструкций (возможный источник логических ошибок). Как правило, их можно разделить на отдельные завершённые фрагменты.

Рассмотрим следующий пример применения условного оператора *if.*



Задача может быть решена по разным структурным схемам.



Структурные схемы алгоритмов

Алгоритм можно сформулировать следующим образом.

1. Ввести значение аргумента *x****.***
2. Проверить принадлежит ли это значение области определения функции.
3. Проверить в каком диапазоне лежит значение аргумента *x*и произвести вычисление функции *y* по соответствующей формуле.

*Практический совет*.

1. Если нет особых требований к быстродействию алгоритма, то при написании программы лучше отдавать предпочтение более простым и надежным алгоритмам.

2. При необходимости организовать разветвления вычислительного процесса, в зависимости от логических условий, по трем и более направлениям, нужно использовать алгоритмы с использованием вложенных структур оператора *if*.

**Задание 1.**

Задайте три вещественных числа и найдите наибольшее из них. Вывод результата оформите по описанным выше требованиям.

**Задание 2.**

Добавьте в предшествующую программу поиск минимального числа и выведите числа в следующем порядке – *«минимальное число, среднее число, максимальное число» -* числовые значения с соответствующими текстовыми пояснениями (*среднее число* - среднее по величине из трёх введённых чисел, а не среднарифметическое).

**Задание 3.** Задайте вещественное число *x* и вычислите значение функции, соответствующее заданному значению.

1. 
2. 
3. 
4. 
5. 

**Лабораторная работа № 3. Разветвляющиеся вычислительные процессы:**

**оператор выбора *case*.**

Для того чтобы в зависимости от конкретных значений исходных данных обеспечить выполнение двух (или более) различных последовательностей операторов, используются оператор выбора *case.*Оператор *case* – осуществляет ветвление программы по значению ключа. Он позволяет выбирать одно из нескольких возможных продолжений программы. Определение направления движения связано с *ключом* – выражением любого порядкового типа.

*Select case <ключи выбора>*

*case <символ(ы) из перечня ключи выбора>*

*<оператор\_1>*

*case <символ(ы) из перечня ключи выбора>*

*<оператор\_2>*

*case <символ(ы) из перечня ключи выбора> <список выбора>*

*<оператор\_3>*

*…*

*case <символ(ы) из перечня ключи выборcа>*

*<оператор\_n>*

*else*

*<оператор\_k>*

*End select*

Сначала вычисляется значение выражения *<ключ выбора>,* затем в последовательности операторов *<список выбора>*  отыскивается оператор, константа которого совпадает с ключом выбора. Одному оператору из списка выбора может соответствовать несколько констант выбора, которые записываются через запятую. Выбранный оператор выполняется, и оператор выбора в целом завершает свою работу. Если в списке выбора не будет найдена константа, соответствующая ключу выбора, то управление передается оператору, следующему за словом *else* , и затем оператор выбора завершает свою работу. Часть *else <оператор>* может быть опущена, тогда, если значение ключа не соответствует списку выбора, оператор *case* просто завершает свою работу и выполнение задания передается следующему за ним оператору.

*Примечание: <символ(ы) из перечня ключи выборcа>* можно задавать также в виде диапазона значений, например - *5:10* или . Допустимы и следующие формы записи *:10* или *5:* , т.е. первая запись справедлива для всех значений меньше или равно 10, вторая запись – для всех значений больше или равно 5.

**Задание 1.**

Определите время года по номеру месяца. Структура оператор *case* должна содержать четыре позиции по временам года – зима, весна, лето, осень – соответствующие им константы (метки) означают номер месяца в порядке следования.

**Задание 2.**

Промоделируйте работу лототрона. Выигрышные номера:

1. Квартира в Москве;
2. Автомобиль;
3. Туристическая путёвка;
4. Зубная щётка;

Все остальные номера – наилучшие пожелания в следующей лотерее.

Для выбора номера используйте ,оператор ввода данных.

**Задание 3.**

Задайте целое число в пределах n≤100 (возраст человека). Сформируйте правильную фразу, используя фрагменты определения - «…год» «…года», «… лет», и выведите её на экран. Например,

«возраст человека 1 год»

«возраст человека 22 года»

«возраст человека 15 лет»

**Лабораторная работа № 4. Операторы повторения.**

Используются два различных оператора, позволяющих запрограммировать повторяющиеся участки программы. Различия в их действиях следующие:

1. Повторение операций *заданное число раз* вне зависимости от условий.
2. Повторение операций пока логическое условие, проверяемое **до** выполнения основного тела цикла, *истинное*.
3. Повторение операций пока логическое условие, проверяемое **после** выполнения основного тела цикла, *ложное*.

Таким образом, перекрывается весь спектр возможных ситуаций повторения действий.

Циклы применяются как для работы с индексированными переменными (например, элементами массива), индексы которых совпадают по имени с переменной цикла, а также и для вычислений, когда в теле цикла изменение значений каких-либо переменных не связано напрямую с переменной цикла ( зависит от какой либо целочисленной переменной, значения которой изменяются в процессе выполнения очередного шага цикла).

Необходимо отметить, что переменная циклане может принимать значения вне диапазона, указанного в описании массива. Но в объявленном диапазоне индексов можно задавать произвольные начальные и конечные значения переменной цикла.

1. Оператор цикла - *DO*.

*do <переменная цикла>= < нач. знач> ,<кон. знач.> ,<шаг.>*

Сначала определяется начальное значение и присваивается переменной цикла. Производится выполнение оператора. Затем значение переменной цикла увеличивается на «*шаг*». Цикл завершается, когда текущее значение переменной цикла становится больше *<кон. знач.>.*  Если условие *переменная цикла* больше или равна *<кон. знач.>* изначально нарушено, то оператор не будет выполнен..

2. Оператор цикла с предусловием -  *do while*

*do while <логическое условие>*

*<оператор>*

Если логическое условие - *«правда»,* то *оператор* выполняется до тех пор, пока логическое условиене примет значение *«ложь»*. В теле цикла обязательно должны изменяться значения параметров логического выражения (иначе может получиться бесконечный цикл). Если логическое условие ложно изначально, то цикл не выполняется ни разу.

Циклы завершается оператором *end do*.

**Задание 1.** **Оператор цикла *DO*.**

Найдите сумму членов арифметической прогрессии



Значения величин *a, d, n* вводить с клавиатуры.

**Задание 2. Оператор цикла с предусловием.**

В цикле с клавиатуры вводятся произвольные числа до тех пор, пока не будет введено отрицательное число. Посчитать, сколько введено положительных чисел.

Использование оператора *do while <логическое условие>*

**Задание 3. Оператор цикла с постусловием.**

Задать стороны прямоугольника *целыми* числами. На сколько квадратов можно его разрезать, если каждый раз отрезать квадрат максимально возможной площади (Рис.1). Использование оператора *do* совместно с оператором *if<логическое условие>exit .*

*Примечание: Различие в выполнении Задания 2 и Задания 3 заключается в том, что логическое условие проверяется «до» или «после» выполнения вычислительного блока операций цикла.*



Рис.1. Схема выбора квадратов

Алгоритм задачи:

* Определяем наибольшую сторону.
* Определяем, сколько раз на ней может уместиться меньшая сторона.
* Для остатка большей стороны повторяется та же процедура.
* Цикл завершается, когда остаток становится равным нулю. (Остаток от деления целых чисел определяется оператором *mod –* большее/меньшее.)
* Ответ представить в виде (например, для a=17, b=4) –

4 квадрата 4\*4

4 квадрата 1\*1 всего 8 квадратов

**Лабораторная работа № 5. Одномерные массивы.**

Массив - набор данных, характерной чертой которого является то, что все компоненты его одного типа. Тип может быть любым. Элементы массива располагаются в памяти непрерывным блоком и имеют одно и то же имя.



Каждая из ячеек массива имеет свой порядковый номер. В данном примере нумерация идёт от 1 до 6. Элементы массива имеют имя – *a*. В принципе, нумерация тех же самых элементов массива может быть произведена в другом диапазоне индексов, например, – от 0 до 5, от -10 до -5 или от -2 до 3. Выбор индексов зависит от условий решаемой задачи.

Чтобы описать массив необходимо задать следующие параметры:

* количество элементов массива;
* тип элементов массива;
* нумерацию элементов.

Массив не является стандартным типом данных, поэтому он задаётся в части описаний.

**Задание 1.**

Используя генератор случайных чисел, создайте массив из 10 элементов целого типа и вычислите их сумму.

**Задание 2.**

Используя генератор случайных чисел, создайте массив из *N* элементов целого типа. Найдите элементы, имеющие минимальное и максимальное значения, запомните их местоположение в массиве (порядковые индексы) и вычислите количество элементов массива, которые расположены между ними. Для определения наибольшего элемента введите дополнительную переменную (например, *maxel*) и занесите в неё значение первого элемента массива. В цикле последовательно сравнивайте элементы массива с содержимым переменной *maxel*. Если какой-либо из очередных элементов имеет большее значение, чем *maxel*, то это значение нужно занести в *maxel,* заменив прежнее значение. Одновременно нужно запоминать местоположение наибольшего элемента в массиве (т.е. его порядковый номер). Аналогично определите наименьший элемент массива.

**Задание 3.**

Используя генератор случайных чисел, создайте массив из *N* элементов целого типа в диапазоне от -100 до +100. Вычислите количество положительных и отрицательных чисел.

**Лабораторная работа № 6. Многомерные массивы.**

Определение многомерного массива аналогично определению одномерного массива.

Чтобы описать массив необходимо задать следующие параметры:

* количество элементов массива по каждому измерению;
* тип элементов массива;
* нумерацию элементов по каждому измерению.

Рис.2. Двумерный и трёхмерный массив

При работе с многомерными массивами, как правило, используются структуры, построенные на вложенных циклах. Например, так можно заполнить двухмерный массив вещественными числами от 0 до 1, используя генератор случайных чисел.

Организация вычислений в структурах вложенных циклов следующая:

1. Выполнятся оператор внешнего цикла – задаётся начальное значение переменной цикла.
2. Для текущего значения переменной внешнего цикла выполняется тело внутреннего цикла в указанных пределах изменения переменной внутреннего цикла.
3. Возвращаемся к внешнему циклу, изменяем значение переменной внешнего цикла на 1 (шаг).
4. Вновь переходим к выполнению тела внутреннего цикла.
5. И т.д. до тех пор, пока не будет исчерпан весь диапазон значений переменой внешнего цикла.

**Задание 1.**

Заполните массив размерностью 5×6 вещественными числами (положительными и отрицательными). Найдите минимальный и максимальный по модулю элементы массива. Запомните их координаты. Массив вывести на экран в виде таблицы.

**Задание 2.**

Заполните массив размерностью 10×10 целыми числами в диапазоне от -5 до 5. Найдите нулевые элементы массива. Запомнить их координаты. Массив вывести на экран в виде таблицы.

**Задание 3.**

Сформируйте матрицу размерностью *N×N* (предельный размер 10×10) в соответствии с приведённой ниже схемой заполнения ячеек целыми числами. Элементы главных диагоналей матрицы равны нулю. Остальные секторы матриц заполнить в соответствии с Рис.3. Рассмотрите матрицы с чётным и нечётным значением *N*. Полученные матрицы вывести на экран в виде двумерных таблиц.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 1 | 0 |  | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 4 |  | 3 | 0 | 1 | 1 | 1 | 0 | 4 |
| 3 | 3 | 0 | 0 | 4 | 4 |  | 3 | 3 | 0 | 1 | 0 | 4 | 4 |
| 3 | 3 | 0 | 0 | 4 | 4 |  | 3 | 3 | 3 | 0 | 4 | 4 | 4 |
| 3 | 0 | 2 | 2 | 0 | 4 |  | 3 | 3 | 0 | 2 | 0 | 4 | 4 |
| 0 | 2 | 2 | 2 | 2 | 0 |  | 3 | 0 | 2 | 2 | 2 | 0 | 4 |
|  |  |  |  |  |  |  | 0 | 2 | 2 | 2 | 2 | 2 | 0 |

Рис.3. Схема заполнения матрицы

*Практический совет.*

Прежде чем приступать к «программированию» (т.е. к написанию каких-либо кодов) нужно выяснить какими характерными чертами обладает поставленная задача – структурными или математическими, физическими особенностями. На основе этого анализа можно приступать к разработке алгоритма. В предлагаемой задаче характерными особенностями является наличие четырёх секторов, следовательно, можно рассмотреть частные задачи заполнения секторов последовательностями одинаковых чисел.

*При разработке алгоритмов Заданий-3,4,5 полезно иметь пред собой заполненные числами таблицы, что облегчит выбор нужных диапазонов переменных циклов.*

Основная задача распадается на четыре подзадачи – заполнением по отдельности каждого из секторов по строкам (двойной цикл). Боковые сектора также нужно разделить на две части – верхнюю (от верхней строки сектора до самой длинной строки сектора) и нижнюю (оставшаяся часть сектора).

**Задание 4.**

Сформируйте матрицу размерностью *N×N* (предельный размер 10×10) в соответствии с приведённой ниже схемой заполнения ячеек ( по диагоналям целыми числами в диапазоне от 1 до *N 2* с шагом +1. Полученную матрицу вывести на экран в виде двумерной таблицы.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 4 | 7 | 11 | 16 |
| 3 | 5 | 8 | 12 | 17 | 22 |
| 6 | 9 | 13 | 18 | 23 | 27 |
| 10 | 14 | 19 | 24 | 28 | 31 |
| 15 | 20 | 25 | 29 | 32 | 34 |
| 21 | 26 | 30 | 33 | 35 | 36 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Рис.4. Схема заполнения матрицы

При формировании матрицы полезно рассматривать по отдельности две диагональные части – верхнюю, включающую в себя большую диагональ, и нижнюю.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Рис.5. Схема заполнения матрицы

Возможны различные алгоритмы заполнения матрицы. Предварительно нужно проанализировать принцип изменения индексов элементов, входящих, например, в строку и определить индексы начального и конечного элемента строки. Рассмотрим верхний сектор, его верхнюю строку.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 4 | 7 | 11 | 16 |
|  | *1* | *2* | *3* | *4* | *5* |

Первый ряд – элементы первой строки верхнего сектора таблицы *a(i,j)*.

Второй ряд – разница между соседнимиэлементами строки *r(j)*

Следовательно, алгоритм заполнения строки *a(i,j+1)=a(i,j)+r(j)*

По этому принципу рассмотрим и остальные строки. Следовательно, помимо основной матрицы необходимо сформировать дополнительный вектор *r(j)*.

По аналогии с верхним сектором нужно проанализировать нижний сектор массива *a(i,j)*.

**Задание 5.**

Сформируйте матрицу размерностью *N×N* в соответствии с приведённой ниже схемой заполнения ячеек целыми числами.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |
| 24 | 25 | 26 | 27 | 28 | 29 | 8 |  |
| 23 | 40 | 41 | 42 | 43 | 30 | 9 |  |
| 22 | 39 | 48 | 49 | 44 | 31 | 10 |  |
| 21 | 38 | 47 | 46 | 45 | 32 | 11 |  |
| 20 | 37 | 36 | 35 | 34 | 33 | 12 |  |
| 19 | 18 | 17 | 16 | 15 | 14 | 13 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Рис.6. Схема заполнения матрицы

Рассмотрите матрицы с нечётным значением *N.* Порядок заполнения матрицы – скручивающаяся по часовой стрелке спираль. Начало спирали левый верхний угол матрицы – элемент с координатами (1,1). Конец спирали - центральный элемент матрицы. Предлагаемая схема заполнения матрицы основана на записи элементов матрицы по виткам спирали. Каждый виток спирали разделён на четыре участка – верхний горизонтальный, правый вертикальный, нижний горизонтальный, левый вертикальный. Нужно записать алгоритм изменения диапазона индексов на каждом витке спирали и на каждом участке отдельной спирали. Задача решается с использованием двойного цикла – внешний цикл определяется количеством витков спирали, внутренние циклы обеспечивают заполнение горизонтальных и вертикальных участков каждого витка спирали. Исходные значения переменных внутренних циклов определяются из задания размеров матрицы *a(i,j)*.

1. **Алгоритмы поиска, выборки и сортировки**

В практике вычислений распространенным действием является поиск необходимого элемента в большом массиве данных и упорядочение элементов массивов по различным признакам. При работе с большими массивами возникают вопросы, связанные с быстродействием поиска (сортировки) и использованием памяти. Рассмотрим наиболее распространенные алгоритмы.

**Лабораторная работа № 7.Алгоритмы поиска.**

**Алгоритмы поиска.**

Списки данных могут быть двух типов – отсортированными или неотсортированными по какому-либо признаку (ключу). В зависимости от этого используется разная тактика поиска. Элемент списка, являющийся предметом поиска, называется ***целевым элементом (ключом)***. Поиск обычно проводится не для того, чтобы убедиться в наличии того или иного элемента, а с целью получит какие-либо данные соответствующие данному ключу. Например, в большом массиве данных сотрудников выделить однофамильцев и т.п. В задаче поиска нужно по ключу определить местоположение элементов в исследуемом списке. Элементы списка должны быть пронумерованы.

Последовательный поиск.

Поиск проводится в ***неотсортированном***списке. Последовательно просматривается список элементов, начиная с первого. Элементов списка, соответствующих ключу, может быть несколько. Самый простой анализ – запись порядкового номера элементов в исходном списке, соответствующих ключу, в отдельный массив. Элементы списка имеют номера от1 до *n*. В случае отсутствия в списке целевого элемента результатом поиска является 0.

Двоичный поиск.

Поиск проводится в ***отсортированном*** списке. Алгоритм поиска следующий. Выбирается средний элемент списка и сравнивается с ключом. Возможны три случая:

1. Средний элемент списка меньше ключа;
2. Средний элемент списка равен ключу;
3. Средний элемент списка больше ключа.

В соответствии с результатом сравнения ведутся последующие действия:

1. Исключается из рассмотрения левая (меньше ключа) половина списка;
2. Поиск завершен. Так как элементов, соответствующих ключу, может быть несколько, то необходимо просмотреть соседние элементы списка с индексами меньшими и большими индекса среднего элемента и определить, таким образом, все элементы, соответствующие ключу.
3. Исключается из рассмотрения правая (больше ключа) половина списка.

Поиск продолжается для уменьшенного вдвое списка данных до достижения нужного результата.

**Задание 1.**

1. Создать ***неотсортированный*** массив из 100 целых чисел в диапазоне значений от 0 до 10, используя ГСЧ. Первый элемент массива имеет индекс, последний элемент . Фактически имеется список из двух полей – численных значений элементов созданного массива, по которым ведётся поиск, и массива их порядковых номеров.
2. Задать значение ключа *key* (ввод с клавиатуры) в диапазоне от 0 до 10.
3. Найти элементы, соответствующие ключу, и поместить ***их индексы*** в отдельный массив. Таких элементов в данной постановке задачи может быть несколько. Для поиска использовать простую процедуру последовательного просмотра данных в цикле.

**Задание 2.**

1. Создать массив из 100 ***отсортированных***в порядке возрастания целых чисел в диапазоне значений от 0 до 10. Первый элемент массива имеет индекс, последний элемент [*можно воспользоваться ГСЧ и процедурой сортировки «пузырьком»*]. Фактически имеется список из двух полей – численных значений элементов созданного массива, по которым ведётся поиск, и массива их порядковых номеров.
2. Задать значение ключа (ввод с клавиатуры) в диапазоне от 0 до 10.
3. Найти элементы, соответствующие ключу, используя, описанную выше, процедуру двоичного поиска, перемещая соответствующим образом границы или на каждом шаге сокращения интервала. Таких элементов в данной постановке задачи может быть несколько. Поскольку список отсортирован, то после нахождения одного элемента, соответствующего ключу, нужно проверить соседние элементы на их соответствие ключу. Поместить ***индексы*** найденных элементов в отдельный массив.

**Лабораторная работа № 8. Алгоритмы выборки.**

**Выборка.**

Иногда бывает нужно выбрать из списка элемент, не имеющий какого-либо конкретного значения. Например, выбрать запись с большим, меньшим, средним по величине элементом или, в общем случае, с -*ым* по величине элементом.

Алгоритм 1.

Неотсортированный список фактически состоит из двух полей – численных значений элементов созданного массива, по которым ведётся выборка, и массива их порядковых номеров. Произвести сортировку массива в порядке убывания [*например, методом «пузырька»*]. Элемент, находящийся на -*ой* позиции отсортированного списка, будет соответствовать условию выборки. В зависимости от постановки задачи элементов с одинаковым значением может быть несколько, поэтому нужно сравнить соседние элементы с выбранным элементом и, в случае их равенства, они также являются решение выборки.

Алгоритм 2.

Неотсортированный список фактически состоит из двух полей – численных значений элементов созданного массива, по которым ведётся выборка, и массива их порядковых номеров. Процесс выборки носит итерационный характер и построен на сокращении интервала поиска на каждом последующем шаге итерации.

Алгоритм начинается с задания значения порядкового номера элемента (по его численному значению в упорядоченном списке элементов по возрастанию). Например, -ый элемент относительно наибольшего элемента. На первой итерации выбирается произвольный -ый элемент списка (можно просто средний по номеру). Затем заполняется пустой дополнительный массив равный размерностью исходному. Производится перестановка списка в новый массив (без сортировки) таким образом, что элементы с значениями меньшими значений -го элемента располагаются на местах с номерами меньшими номера ключевого элемента, а элементы большие - на местах с номерами большими номера ключевого элемента. Заполнение массива проводится с двух концов в зависимости от численного значения очередного переставляемого элемента. Тогда выбранный -ый элемент списка расположится в дополнительном массиве в -ой ячейке.

Если значения и совпадают, то выборка закончена. В противном случае процесс продолжается на следующих итерациях, когда интервалы поиска сокращаются. Если то анализируется часть списка расположенная справа от-ой ячейки, если , то анализируется часть списка расположенная слева.

**Задание 1.** Алгоритм 1.

1. Создать ***неотсортированный*** массив из 100 целых чисел в диапазоне значений от 0 до 10, используя ГСЧ. Первый элемент массива имеет индекс, последний элемент . Фактически имеется список из двух полей – численных значений элементов созданного массива, по которым ведётся выборка, и массива их порядковых номеров. (сохранить этот массив также и во вспомогательном массиве)
2. Задать -ый номер ячейки массива, в которой должен располагаться интересующий нас элемент списка (ключевой элемент).
3. Произвести сортировку исходного несортированного массива в порядке убывания [*например, методом «пузырька»*].
4. Поскольку в исходном массиве могут быть одинаковые элементы, то нужно создать дополнительный массив (из элементов сортированного массива), в котором собраны только элементы с различными значениями.
5. Элемент, находящийся на -*ой* позиции этого дополнительного отсортированного списка, будет соответствовать условию выборки.
6. В данной постановке задачи элементов с одинаковыми значениями может быть несколько, поэтому нужно в отсортированном и исходном неотсоиртированном массивах определить индексы элементов соответствующие ключу, они также являются решением выборки.

**Задание 2.** Алгоритм 2.

1. Создать ***неотсортированный*** массив из 100 целых чисел в диапазоне значений от 0 до 10, используя ГСЧ. Первый элемент массива имеет индекс, последний элемент . Фактически имеется список из двух полей – численных значений элементов созданного массива, по которым ведётся выборка, и массива их порядковых номеров.
2. Задать -ый номер ячейки массива (отсчёт от наибольшего элемента), в которой должен располагаться интересующий нас элемент списка (ключевой элемент).
3. В исходном списке выбирать произвольный -ый элемент списка (лучше выбрать просто средний по номеру
4. Нужно произвести перестановку списка (без сортировки) в дополнительный массив таким образом, что элементы с значениями меньшими значений -го элемента списка располагались на местах с номерами меньшими номера -го элемент (массив заполняется с левого края), а элементы большие - на местах с номерами большими номера -го элемента. (массив заполняется с правого края). Остаётся незаполненной одна ячейка массива, в которую помещается -ый элемент. Эта ячейка имеет индекс .
5. Если значения и совпадают, то выборка закончена. В противном случае процесс продолжается на следующих итерациях. Интервалы поиска сокращаются. Если , то анализируется часть списка расположенная справа от-ой ячейки (включая ячейку), если , то анализируется часть списка, расположенная слева (включая ячейку)
6. Процесс продолжается для последовательно сокращающихся интервалов пока значения и не совпадут, выборка закончена

*Пример:.*

Используя генератор случайных чисел формируем массив размерностью . Как видно из текста в массиве много элементов с повторяющимися значениями. Нужно избавится от дублирующих элементов

---s--start massiv---------------

**1 9 3 8 2**  3  **6 7**  9 7

7 2 3  **0**  0 0 6  **4**  6 2

8 2 1 2 4 8 6 3 4 7

8 7 4 4 0 6 3 8 7 0

**5**  1 7 1 5 6 9 3 2 8

1 5 7 9 7 2 1 0 2 3

0 7 3 4 2 6  **10**  8 9 7

5 3 1 4 6 9 10 2 7 6

9 4 1 2 3 1 1 9 5 4

6 7 8 9 5 7 9 8 3 5

*Примечание: для лучшего визуального анализа массива лучше выводить его на печать построчно по 10 элементов в строке.*

Для этого просматриваем весь исходный массив с начала до конца и присваиваем дублирующим элементам значения отличные от основного диапазона чисел (например, 100)

---s--start massiv--var----------

**1 9 3 8 2** 100 **6 7** 100 100

100 100 100 0 100 100 100  **4**  100 100

100 100 100 100 100 100 100 100 100 100

100 100 100 100 100 100 100 100 100 100.

**5** 100 100 100 100 100 100 100 100 100

100 100 100 100 100 100 100 100 100 100

100 100 100 100 100 100 **10**  100 100 100

100 100 100 100 100 100 100 100 100 100

100 100 100 100 100 100 100 100 100 100

100 100 100 100 100 100 100 100 100 100

Формируем массив элементов с отличающимися значениями в порядке их следования в исходном списке (в данном примере 11 чисел). Этот массив и является объектом выборки.

---sz--quantity--------------------

1 9 3 8 2 6 7 0 4 5

10

===================================.

Основной алгоритм выборки – итерационный процесс сокращения интервала поиска за счёт переноса границ и .

====== ПРОХОД 1 =============

Пробная ячейка p=6, значение элемента массива в ней sp=6.

Число элементов слева от ключевого значения j1=6. Индекс j1 в процессе заполнения массива слева пробегает значения от 0 до 6 (на этом шаге итерации).

Число элементов справа от ключевого значения j2=4 Индекс j2 в процессе заполнения массива справа пробегает значения от 0 до 4 (на этом шаге итерации)

*Примечание: на каждом шаге итерации оба индекса j1 и j2 вначале должны быть обнулены. При завершении прохода значение j1=4 соответствует значению p1 (p1=j4).*

Индекс левой границы n1=1

Индекс правой границы n2=11

Индекс ключевой ячейки nk= 6

---------szp-------

**1 3 2**  **0**  **4 5 6** 10 7 8

9

====== ПРОХОД 2 =============

p1= 4 sp= 0

j1= 0 j2= 6 n1= 1 n2= 7 nk= 6

---------szp-------

0 6 5 4 2 3 1

====== ПРОХОД 3 =============

p1= 4 sp= 4

j1= 4 j2= 2 n1= 1 n2= 7 nk= 6

---------szp-------

0 2 3 1 4 5 6

====== ПРОХОД 4 =============

p1= 6 sp= 5

j1= 1 j2= 1 n1= 5 n2= 7 nk= 6

---------szp-------

4 5 6

---- FINISH ---------------

element numder 6 quantity= 5

**Лабораторная работа № 9. Алгоритмы сортировки.**

Задача сортировки состоит в перестановке членов последовательности таким образом, чтобы выполнялось условие - , для всех *i* от 0 до *n*.

Возможна ситуация, когда элементы состоят из нескольких полей:

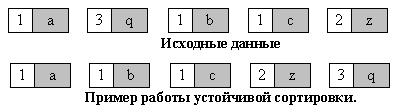
*y*

*x*

Если сравнение проводится только по полю ***x***, то ***x*** называют ключом, по которому производится сортировка. На практике, в качестве ***x*** часто выступает число (номер), а поле ***y*** хранит какие-либо данные, никак не влияющие на работу алгоритма.

Рассмотрим параметры, по которым будет производиться оценка алгоритмов.

1. *Время* сортировки - основной параметр, характеризующий быстродействие алгоритма.
2. *Память* - ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных. При оценке используемой памяти не будет учитываться место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы.
3. *Устойчивость* - устойчивая сортировка не меняет взаимного расположения равных элементов. Такое свойство может быть очень полезным, если они состоят из нескольких полей, а сортировка происходит по одному из них, например, по x.



дополнительными полями "a", "b", "c" осталось прежним: элемент с полем "a", затем - с "b", затем - с "c".  
  
Взаимное расположение равных элементов с ключом 1 и дополнительными полями "a", "b", "c" изменилось.

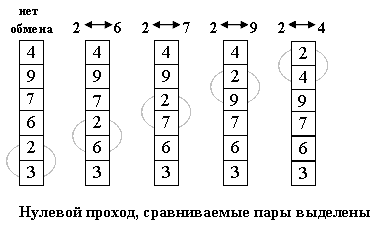


1. *Естественность поведения* - эффективность метода при обработке уже отсортированных, или частично отсортированных данных. Алгоритм ведет себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.

**Сортировка пузырьком**

Расположим элементы массива снизу-вверх, от нулевого элемента - к последнему элементу.

Идея метода: шаг сортировки состоит в проходе снизу-вверх по массиву. По пути прохода сравниваются значения пары соседних элементов. Если элементы некоторой пары находятся в неправильном порядке, то меняем их местами.



. Схема продвижения «лёгкого» элемента к началу массива

После нулевого прохода по массиву «вверху» оказывается самый «легкий» элемент - отсюда аналогия с пузырьком. Следующий проход делается до второго сверху элемента, таким образом, второй по величине элемент поднимается на правильную позицию.

Делаем проходы по все уменьшающейся нижней части массива до тех пор, пока в ней не останется только один элемент. На этом сортировка заканчивается, так как последовательность упорядочена по возрастанию.

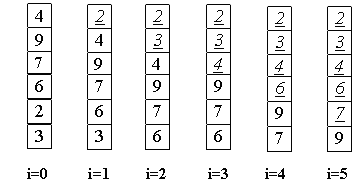


Схема сортировки элементов массива

**Сортировка вставками**

Будем разбирать алгоритм, рассматривая его действия на i-м шаге. Последовательность к этому моменту разделена на две части: готовую часть *a*[*0*]*...a*[*i*] и неупорядоченную - *a*[*i+1*]*...a*[*n*]*.* На каждом последующем (i+1)-м шаге алгоритма берем *a*[*i+1*] и вставляем на нужное место в готовую часть массива. Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним. В зависимости от результата сравнения элемент либо остается на текущем месте (вставка завершена) либо они меняются местами и процесс повторяется.

Можно использовать и другой алгоритм – очередной элемент *a*[*i+1*] заносим в буфер *x* и сравниваем значение *x* с содержимым уже отсортированной части массива до тех пор, пока не дойдём до элемента меньшего *x* или начала массива. Затем все элементы большие *x* сдвигаем на одну позицию вправо, а в освободившуюся ячейку заносим значение из буфера *x*. Этот алгоритм более эффективен, так как в нём меньше операций пересылки.

Пример: Последовательность на текущий момент. Часть a[0]..a[2 ] уже упорядочена (Рис.9.).



Рис.9. Последовательность элементов массива на текущий момент

Производим вставку числа 2 в отсортированную последовательность. Сравниваемые пары выделены.







Рис.10. Схема вставки элемента массива на «нужное» место

Таким образом, в процессе вставки мы «просеиваем» элемент *a*[*i+1*] к началу массива, останавливаясь в случае, когда

1. Найден элемент, меньший *a*[*i+1*]

или

1. Достигнуто начало последовательности.

**Задание 1.**

1. Создайте массив *вещественных* (*real*) случайных чисел .

В полученном массиве найти и вывести на экран элементы с максимальным и минимальным значением, а также их индексы (порядковые номера) в исходном массиве.

3. Отсортируйте массив по возрастанию (убыванию), используя изложенные выше методы, и выведите все результаты на экран.

**Задание 2**. **Анализ больших массивов чисел**.

Предположим что, сформирован массив N случайных чисел в диапазоне [0-1]. Этот диапазон можно разделить на 10 поддиапазонов [0-0.1, 0.1-0.2, 0.2-0.3, … 0.9-1.0]. В каждый поддиапазон попадёт определённое количество чисел – n1, n2, n3, … n10. Создайте массив вещественныхчисел, задав его размер с клавиатуры. Разделите диапазон значений чисел массива на 10 частей. Определите количество чисел массива, попавших в каждый поддиапазон. Рассмотрите массивы из 50, 500, 1000, 5000 чисел. Для того чтобы определить, в какой поддиапазон попадает число можно воспользоваться следующим алгоритмом. Диапазон значений элементов массива от 0 до 1. Поддиапазоны имеют номера 1,2, .. 10. Определим номер поддиапазона, куда попадает, например, число 0,31567. Разделим это число на ширину поддиапазона 0,31567/0,1 = 3,1567, прибавим 1 и отбросим дробную часть. Номер поддиапазона - 4 (значение типа *integer*).