Курс «Прикладное программирование и основы алгоритмов»

Литература

1. В.В. Фаронов Турбо Паскаль 7.0 Начальный курс. М., Изд. «Нолидж», 2001 г.
2. В.С.Зубов и др. Сборник задач по базовой компьютерной подготовке. Под общей редакцией И.Н. Котаровой. М., МЭИ, 1998 г.

Лекция №1

История развития вычислительной техники

Потребность в автоматизации вычислений и обработки информации возникла очень давно, когда люди считали с помощью камешков, палочек, счётов. Механические устройства вычислений появились в 17 веке. Работы по первой Аналитической машине появились в первой половине 19 века. Их проводил английский математик Чарльз Бэббидж.

Первые вычислительные машины появились в годы второй мировой войны (1941 г. – сделал немецкий инженер Конрад Цузе, 1943 г. – сделал американец Говард Эйкен – название «Марк-1»). Эти машины были сделаны на основе реле, а в 1945 г. была сделана первая машина на основе электронных ламп под названием ENIAC (Эниак). Научный доклад по этой машине сделал математик Джон фон Нейман, в котором изложил общие принципы работы компьютеров.

Принципы фон Неймана:

1. Компьютер должен состоять из следующих устройств:
2. Арифметическо-логическое устройство (АЛУ);
3. Устройство управления (для организации выполнения программ);
4. Запоминающее устройство (ЗУ) или память (П); (ОП – оперативная память);
5. Внешние устройства (ВУ) для ввода-вывода информации.

АЛУ ↔ УУ ↔ ВУ

ОП ↔

1. Процесс работы компьютера:
2. с ВУ информация поступает в ОП (ввод);
3. УУ и АЛУ обрабатывают информацию из ОП;
4. Результат обработки размещается в ОП;
5. Результат из ОП выводится на ВУ (вывод).

В современных компьютерах АЛУ и УУ объединены в центральный процессор (ЦП):

ОП↔ ЦП↔ ВУ

Внешних устройств достаточно много. ЦП тоже может быть несколько, работающих в параллель.

Компьютеры 40-х и 50-х годов 20 века были очень большими и занимали целые машинные залы (минимум 100 квадратных метров) и были очень дорогими в цене. Рынок заставлял людей изобретать более компактные и более дешёвые компьютеры. В 1948 г. был изобретён транзистор, который заменил электронную лампу. Во второй половине 50-х годов появились ЭВМ на основе транзисторов. Эти компьютеры были в сотни раз меньше ламповых. К середине 60-х годов были изобретены интегральные схемы (или чипы). В 1968 году появился первый компьютер на интегральных схемах, а с 1970 года фирма Intel начала продавать интегральные схемы памяти. В том же 70-м году появился первый микропроцессор Intel-4004 – прообраз ЦП современных персональных компьютеров.

В нашей стране первая ЭВМ появилась в 1949 году. Это тоже были большие и средние ЭВМ. Персональные компьютеры у нас появились с начала 80-х годов прошлого века.

Представление информации в компьютере

Компьютер обрабатывает только цифровую информацию. Вся другая информация (тексты, изображения, звуки, видео и др.) кодируется в цифровую форму. Единицей информации является 1 бит (двоичный разряд), принимающий значение либо 0, либо 1. Восемь бит составляет 1 байт. В одном байте можно закодировать 256 символов (28=256). 1 Кб=1024 (210) байта. 1 Мб=1024 Кб, 1 Гб=1024 Мб. Так как компьютер «понимает» только 0 и 1, то используется двоичная система счисления. На основе двоичной могут быть построены 8-ричная, 16-ричная, 10-ричная системы счисления.

Программы для первых компьютеров писались на машинном языке. Программа – это набор машинных команд. Машинная команда состоит из двух частей:

<код операции> и <адресная часть>

Адрес оперативной памяти – это номер байта ОП. Команда в трёхадресной ЭВМ выглядит следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| КОП | А1 | А2 | А3 |

Выполнение операции: (А1) КОП (А2) => А3

Где (А1) – содержимое ячейки по адресу А1

Пример: Y=a+b

Исходные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Адрес | Команда | Содержимое |  |  | Программа |
| 100 |  | a |  |  |  |
| 104 |  | b |  |  |  |
| 108 |  | Y |  |  |  |
|  |  |  |  |  |  |
| 200 | 01 | 100 | 104 | 108 | Y=a+b |
| 204 | 00 |  |  |  | Останов |

Примером машинного языка является язык Ассемблера. Машинные языки очень трудны и неудобны для человека. Поэтому в середине 50-х были изобретены языки высокого уровня – алгоритмические языки. Первым алгоритмическим языком был Фортран (1958 г.) для научных вычислений. В 1960 г. был придуман АЛГОЛ-60, а в 1964 г. – язык PL/1. Позже PL/1 был преобразован в Pascal. Кроме Паскаля сейчас известны и используются языки C, JAVA, ЛИСП, ПРОЛОГ и др. Современные объектно-ориентированные языки: Visual Basic, C++ и Visual C, Delphi и др.

Программы на языках высокого уровня преобразуются в программы из машинных команд с помощью программ трансляторов или компиляторов, либо интерпретируются с помощью программ-интерпретаторов.

Лекция № 2

Общие методики решения задач на ЭВМ

Первая методика

Программист

|  |  |  |
| --- | --- | --- |
| Программа |  |  |
| в машинных | Решение |  |
| кодах |  | Результат |
| Исходные | ЭВМ |  |
| данные |  |  |

Пользователь

Разновидности программ для компьютеров

Прикладные программы => Операционные оболочки => Операционные системы => системы программирования => языки программирования => машинные коды => ЭВМ

1. Прикладные программы – редактирование текстов (Word), рисование картинок (Paint) и др.
2. Системные программы – операционные системы, драйверы, операционные оболочки (пример: Norton Commander), утилиты (копирования, антивирусы, архиваторы, диагностики компьютера и др.)
3. Инструментальные системы (системы программирования), обеспечивающие создание новых программ для компьютера.

**Основные функции ОС:**

1. Управление данными;
2. Организация интерфейса между пользователем и ЭВМ;
3. Организация файловой системы;
4. Первоначальная загрузка программ.

**Популярные ОС:**

1. MS DOS;
2. Windows;
3. UNIX;
4. OS/2;
5. Система виртуальных машин (OS/370, OS/390).

Этапы прохождения задачи в ЭВМ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| программы | ЭВМ | программа |  |  |  |
| На АЯ | трансляция | в | Редак- | Загру- | Ре- |
| (паскаль) |  | машинных | Тирова- | зочный | Ше- |
|  |  | кодах | ние | модуль | ние |
| Исходные | транслятор |  | редактор |  | Резу- |
| данные |  |  |  |  | льтат |

1. Трансляция;
2. Редактирование;
3. Выполнение.

Имя программы – это <имя файла>.<расширение>

Примеры:

.PAS – на Паскале;

.C – на Си;

.BAS – на бейсике;

.ASM – на Ассемблере;

.FOR – на Фортране;

.DBF – база данных;

.EXE – абсолютный модуль.

Транслятор – это программа, которая переводит программы с алгоритмического языка на внутренний (машинный) язык (для каждого языка программирования свой транслятор/компилятор).

Этапы решения задачи на ЭВМ.

1 этап Физическая постановка задачи

Пример: Рассчитать расход краски на забор R=K\*S, где

K – расход краски на единицу площади,

S – площадь забора.

2 этап Математическая постановка задачи

R = f(x) = K\*

3 этап Выбор численного метода

1. Метод прямоугольников

S = \*h; h = (B-A)/n

Возьмём шаг h/2 и посчитать S1 |S1 – S| < eps

1. Метод трапеции (приближение – интерполяция):

S = ∑ h\*(f(x)+f(x+h))/2

1. Метод Симпсона

4 этап Разработка алгоритма решения численных методов

Алгоритм – конечная последовательность действий, которая применяется к исходным данным для получения результата.

Свойства алгоритма:

А) массовость;

Б) конечность;

Б) конечность;

В) детерминированность (применение одного и того же алгоритма к одним и тем же исходным данным должно дать один и тот же результат).

Алгоритмы описываются на языках: 1) блок-схем; 2) псевдокодов.

5 этап Написание программы на алгоритмическом языке

6 этап Компиляция (трансляция) программы – обнаружение ошибок в программе

7 этап Процесс отладки программ:

1. Исправление синтаксических ошибок;
2. Исправление семантических ошибок.

8 этап Тестирование программы

9 этап Эксплуатация программы

Приложение:

Работа в среде TP-7 – быстрые клавиши:

F2 – сохранить программу (записать на диск);

F6 – сделать активным следующее окно;

Alt+F9 – компиляция программы;

Ctrl+F9 – выполнение программы;

Alt+F5 – сменить окно редактора на окно вывода результатов работы программы и наоборот;

Alt+F3 – закрывает активное окно;

Alt+X – выход из TP-7.

Лекция № 3

Алгоритмический язык Pascal

Требования к любым алгоритмическим языкам:

1. Близок к языку математики;
2. Близок к естественному языку (например, к английскому);
3. Однозначность любых конструкций.

Язык Pascal был создан в 70-е годы прошлого века как язык структурного программирования. В 1984 году фирма Borland написала среду Turbo Pascal, работающую под управлением операционной системы (после выполнения очередной команды ОС автоматическое возвращение в среду).

Алфавит языка

1. Латинские строчные и прописные буквы: a, b, …, z и A, B, …, Z
2. Цифры: 0, 1, …, 9
3. Символы «\_», « » и управляющие символы, например: «конец строки», «новая строка», «табуляция».
4. Спец. символы: +, -, \*, /, =, <, >, [, ], ., ,, (, ), :, ;, ^, @, {, }, $, #, ‘.
5. Составные символы, воспринимаемые как один символ (пробелы между ними недопустимы): <=, >=, :=, (\*, \*), (., .), ..

В алфавит не входят: 1) русские буквы (кириллица); 2) символы |, %, “ и др.

При описании сложных вычислительных процессов построение программ на языке программирования является очень трудным процессом. Чтобы облегчить процесс, программа описывается на языке более высокого уровня, например на языке блок-схем (языке алгоритмов).

Язык блок-схем состоит из 6 операторов:

1. Оператор начала и останова программы
2. Оператор ввода-вывода –ввод исходных данных и вывод результатов работы программы
3. Оператор действия – реализует вычисления и запоминает результат вычислений
4. Условный оператор – если <условие> выполняется, то работает процесс А, если <условие> не выполняется, то работает процесс В.

1. Оператор-ссылка: для продолжения блок-схемы на другой странице
2. Оператор вызова подпрограммы:

Основные операторы языка Pascal:

1. Составной оператор: begin … end;
2. Оператор ввода/вывода: read/write;
3. Оператор присваивания: := ;
4. Условный оператор и оператор выбора: if и case of;
5. Оператор цикла (повторений): for … do; while … do; repeat … until;
6. Оператор перехода: goto;
7. Оператор процедуры: procedure и function;
8. Пустой оператор ;
9. Комментарии: { … }.

Определение программы

Программа = Алгоритм + Данные

Данные – это информация, обрабатываемая на компьютере.

Классификация данных:

1. По классу: а) входные; б) выходные; в) промежуточные.
2. По типу: а) целые; б) вещественные; в) символьные; г) логические.
3. По структуре: а) константы; б) простые переменные; в) массивы;

г) множества; д) записи; е) списки.

Алгоритмы:

Алгоритм – это последовательность действий вычислительного процесса, приводящая к вычислению верного результата. Алгоритмы (последовательности действий) могут быть различны, но результат должен быть один и тот же.

Алгоритмы бывают:

1. Последовательные;
2. Разветвлённые;
3. Циклические.

Последовательный алгоритм – это строгая последовательность операторов ввода/вывода и операторов действия.

Пример: определить y=Ax2+B

Разветвлённый алгоритм – обязательное использование как минимум одного условного оператора.

Циклический алгоритм – в языке блок-схем нет оператора цикла (повторений). Для отображения оператора цикла используется сочетание оператора действия и условного оператора.

Пример: определить сумму n чисел: S=x1+x2+x3+ … +xn=∑xi

начало

Ввод N

S=0

I=1

Ввод Xi

S=S+Xi

I=I+1

I>N Вывод S конец

Лекция № 4

Типы данных

Программа = Алгоритм + Данные

В компьютере все данные выглядят как огромные массивы ячеек (разрядов) памяти, а человеку более привычны математические понятия: числа, множества, последовательности (а лучше: тексты и изображения).

В языках программирования существует принцип, что любая константа, переменная, выражение или функция относится к некоторому типу данных. Фактически тип данных характеризует множество значений данных. Поэтому используется правило, по которому тип явно указывается в описании константы, переменной или функции. Число различных значений, входящих в тип Т, называется мощностью Т. Мощность задаёт размер памяти, необходимой для размещения переменной x типа Т. Принадлежность к типу обозначается x:T.

Существуют простые типы данных и составные. К простым типам данных относятся:

1. Перечислимый TYPE T=(c1,c2, …, cn) например: TYPE BOOLEAN=(FALSE, TRUE)
2. Целые числа – INTEGER
3. Вещественные (дробные) числа – REAL
4. Символьный тип – CHAR

К составным типам относятся:

1. Массивы
2. Записи
3. Множества
4. Последовательности

Общая схема:

Порядковые Целые

Типы Простые Вещественные

Структурированные Логический

Массивы

Указатели Записи Символьный

Строки Множества Перечисляемый

Процедурные Файлы Тип-диапазон

Объекты

Элементы языка Pascal:

1. Идентификаторы
2. Константы
3. Выражения
4. Операции

Идентификатор – это имя константы, переменной, метки, типа, объекта, процедуры, функции, модуля, программы, поля в записи. Идентификатор (имя) – это последовательность букв и цифр, начиная с буквы. Длина идентификатора до 63 символов. Пробел и специальные символы не могут входить в идентификатор (вместо пробела используют символ подчёркивания). Зарезервированные слова языка Pascal не могут быть использованы в качестве идентификатора. Зарезервированные слова – это названия операторов, операций и т.п. Примеры: a, ALPHA, beta.

Константа – это величина, значение которой постоянно. Константы бывают:

- числовые (целые и вещественные);

- символьные;

- строковые;

- логические;

- множественные;

- адресные.

Целые числа записываются со знаком и без него по обычным правилам. 0 – тоже целое число.

|  |  |  |  |
| --- | --- | --- | --- |
| Длина в байтах | Описание | Диапазон | Примечание |
| 2 | Integer | |N| <=32767 | ±215-1 |
| 1 | Shortint | |N| <= 127 | ±27-1 |
| 1 | Byte | N >= 0 | 0<=N<=255(28-1) |
| 2 | Word | 0<=N<=65535 | 216-1 |
| 4 | Longint | |N|<=2147483647 | ±232-1 |

Вещественные числа записываются со знаком или без него с использованием десятичной точки и/или экспоненциальной части (с помощью мантиссы и порядка). Вместо числа 10 записывается буква Е, например: 3.14Е5 = 3.14 умноженное на 10 в 5 степени (3.14 – мантисса, 5 – порядок). С десятичной точкой: 2.5, -3.75

|  |  |  |  |
| --- | --- | --- | --- |
| Длина в байтах | Описание | Диапазон 10n порядка | Количество значащих цифр в мантиссе |
| 6 | Real | -39:+38 | 11:12 |
| 4 | Single | -45:+38 | 7:8 |
| 8 | Double | -324:+308 | 15:16 |
| 10 | Extended | -4951:+4932 | 19:20 |
| 8 | comp | -263+1:263-1 |  |

Пример:

S – integer

S:=25786\*3+0.5 даст ERROR, так как выражение вещественное.

Символьная константа ‘A’ – char 1 байт A

Строковая константа ‘ABC’ – string 3ABC

Логическая константа – Boolean – true или false (1- true, 0- false).

Константы объявляются в разделе констант:

Const <идентификатор>=<значение>; … - нетипизированная константа

Const <идентификатор>:<тип>=<значение>; … - типизированная константа (это уже переменная).

Переменные задаются идентификатором и типом данных в разделе VAR:

Var <идентификатор>:<тип>; …

Выражения

Выражение – это правило получения нового значения из имеющихся констант, переменных и обращений к функциям с помощью знаков операций и скобок. Выражения бывают арифметические, логические, строковые.

Пример арифметического выражения: A+B + - операция, A,B – операнды.

Арифметические операции: +,-,\*,/.

|  |  |
| --- | --- |
| № приоритета | Операции |
| 0 | ( ) |
| 1 | not |
| 2 | and |
| 3 | Or / xor |
| 4 | +, - одноместные |
| 5 | \*, / |
| 6 | +, - двуместные |

Пример:

=> 2.5\*sin((a+b)/2)/(3\*cos(1.5\*x\*y))

Лекция № 5

Логические выражения. Данные типа Boolean.

True – 1

False – 0

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | A and B | A or B | Not A | Not B | A xor B |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Операции отношения: =, <>, >, >=, <, <= (равно, не равно, больше, больше либо равно, меньше, меньше либо равно).

В логических выражениях можно использовать логические операции (and, or, not, xor), операции отношения, логические переменные и логические функции.

В математике: A<=x<=B (отрезок) В Pascal: (x>=A) and (x<=B)

В математике: x2+y2<=R2 (окружность – 3-я четверть: x<0 и y<0)

В Pascal: (sqr(x)+sqr(y) <= sqr(R)) and (not((x>0) and (y>0)))

Структура программы на Pascal

[program <имя>;]

{Раздел описаний}

Begin

{Раздел описаний}

End.

Это короткая структура, а более полная:

[program <имя>;]

[uses <имя>, …;] {подключение модулей}

[label <имя>, …;] {описание меток}

[const <имя>=<значение>; …] {описание констант}

[type <имя>=<значение>; … ] { описание типов данных}

[var <имя>, …:<тип>; …] {описание переменных, массивов и др.}

[procedure <имя> …;

Begin {описание подпрограмм-процедур}

…

End; ]

[function <имя> …: <тип>;

Begin {описание подпрограмм-функций}

…

End;

Begin

{раздел операторов}

End.

Комментарием является любой текст, заключённый в круглые скобки:

{текст} или /\* текст \*/

Операторы языка Pascal перечислены в лекции № 3.

При рассмотрении операторов мы будем использовать два понятия: синтаксис оператора и семантика оператора.

Синтаксис оператора – это правила правописания оператора (как пишется оператор). Семантика оператора – это смысл оператора (как работает оператор). Семантику мы будем описывать с помощью языка блок-схем.

Пустой оператор.

; или ;

Пример: m1:; m2:; - после метки (переход на метку и ничего не делать). Перед end “;” можно и не ставить.

Оператор присваивания.

:= или ЛЧ:=ПЧ;

ЛЧ – левая часть, ПЧ – правая часть.

Обычно в качестве ЛЧ записывается переменная (простая или с индексом). В качестве ПЧ записывается выражение.

Примеры:

1. VAR S,a,b: real;

…

S:=a+b;

1. VAR b:Boolean;

C,d,x,y:real;

…

B:=(x>=y) and not (c-d>0);

В операторе «присвоить значение» первым вычисляется выражение, стоящее справа (с учётом типа); полученное выражение преобразуется к типу переменной, стоящей слева; Эта переменная получает вычисленное и преобразованное значение.

Оператор перехода

Синтаксис: GOTO <метка>;

Метка – это идентификатор или целое число без знака. Пример:

Label m1, 999, KON;

Семантика: меняет естественную последовательность действий. Пример:

Begin

X:=a+b;

GOTO m1;

999: GOTO KON;

M1: y:=2\*x;

GOTO 999;

KON: end.

Условный оператор

Синтаксис: IF <B> THEN <S1> [ELSE <S2>];

Семантика: Если <B> истинно, то выполняется оператор S1, иначе выполняется оператор S2.

S1

S2

S1

или

Составной оператор

Синтаксис: begin <S1>; <S2>; … ; <SN> end;

Семантика: …

S1

S2

SN

Лекция № 6

Условный и составной оператор

Пример1: если A>0 и B>0, то y:=1 и x:=1, иначе y:=0, а x:=2.

IF (A>0) and (B>0) THEN begin y:=1; x:=1; end

ELSE begin y:=0; x:=2 end;

Пример 2:

На языке Pascal:

IF <B1> THEN ELSE begin <S2>;

IF <B2> THEN begin <S3>; <S4>; end;

End;

Пример3: sin2(kx), если x<0

Вычислить значение y= ln(x+1), если 0<=x<=1

Cos(x/2), если x>1

На языке Pascal:

IF (x>=0) and (x<=1) THEN y:=ln(x+1)

ELSE IF (x<0) THEN y:=sqr(sin(k\*x)) ELSE y:=cos(x/2);

Оператор выбора

Синтаксис: case <i> of

<i1>, … :<S1>;

<i2>, … :<S2>;

…

<in>, … :<Sn>;

[ELSE <Sm>]

END;

Где I – это переменная счётного типа (все, кроме real).

Семантика:

Sm

Sn

S2

Пример: var i:integer;

Case i of

1,2,3: y:=0;

5,8: y:=1;

12: y:=5

End;

То же самое с помощью условного оператора IF:

IF (i=1) or (i=2) or (i=3) THEN y:=0;

IF (i=5) or (i=8) THEN y:=1;

IF i=12 THEN y:=5;

Пример: IF <B> THEN <S1> [ELSE <S2>];

Var f:boolean;

Begin

F:=B;

Case f of

True: <S1>;

False: <S2>;

End;

Массивы

Массив (структурированный тип данных) – это однотипная совокупность данных, поименованная, расположенная последовательно (в порядке возрастания индекса) в оперативной памяти.

A

|  |  |  |  |
| --- | --- | --- | --- |
| A[1] | A[2] | … | A[100] |

Индекс элемента массива записывается в квадратных скобках

1 способ описания массива:

Const n=100;

Var A: array [1..n] of real;

Память под элементы массива выделяется на этапе компиляции программы.

2 способ описания массива:

Type MAS1=array[0..100] of real;

VAR A:MAS1;

До любого элемента в массиве можно добраться по его индексу: A[i], где i – это индекс.

Оператор цикла с параметром.

Если нам известно число повторений одних и тех же действий, то удобно применять этот оператор.

Синтаксис:

1. FOR <параметр цикла>:=<начальное значение> TO <конечное значение> DO <оператор>;
2. FOR <параметр цикла>:=<конечное значение> DOWNTO <начальное значение> DO <оператор>;

<оператор> (тело цикла) может быть простым, составным и пустым.

<параметр цикла> может быть только целочисленного или перечислимого типа и должен быть описан как переменная.

Шаг цикла всегда постоянный (+1 или -1) – это недостаток. Если нужен другой шаг, то используются другие операторы цикла WHILE…DO или REPEAT…UNTIL.

Семантика:

i>n

I=i+1

оператор

I=1

да

1. Параметру цикла присваивается начальное значение;
2. Следует проверка: параметр цикла больше конечного значения?
3. Если проверка истинна, то цикл заканчивается, и управление передается следующему за циклом оператору;
4. Если проверка ложна, то выполняется оператор (тело цикла);
5. Параметр цикла получает следующее значение и всё повторяется со 2) пункта.

Более правильно:

I=i+1

оператор

i>n

I=1

Да

Нет

Нельзя! Изменять начальное, конечное значения и сам параметр цикла внутри цикла. Нельзя писать:

FOR i:=i-5 TO i+5 DO …

Не имеют смысла операторы:

FOR i:=5 TO 4 DO …

FOR i:=4 DOWNTO 5 DO …

Оператор FOR i:=N TO N DO … будет выполнен только один раз.

Пример: Дан массив из 100 элементов A[1..100]. Найти две суммы следующих элементов:

S1=A[1]+A[4]+A[7]+A[10]+ …

S2=A[2]+A[5]+A[8]+A[11]+ …

Program M1;

Type MAS=array [1..100] of real;

Var i,n:integer;

A:MAS;

S1,S2:real;

Begin

<ввод n>;

<ввод массива A>;

S1:=0;

S2:=0;

FOR i:=1 TO n div 3 DO

Begin

S1:=S1+A[3\*(i-1)+1];

S2:=S2+A[3\*(i-1)+2];

End;

{Печать результатов}

Writeln (S1,S2);

End.

Лекция №7

Оператор цикла с предусловием

Синтаксис:

WHILE <B> DO <S>;

B – логическое выражение;

S – оператор.

Семантика:

S

B

False

true

Оператор S будет выполняться циклически до тех пор, пока истинно <B>. <B> проверяется до начала выполнения цикла. Если <B> ложно, то тело цикла игнорируется, и управление передается оператору, стоящему сразу за телом цикла.

При программировании оператора WHILE … DO следует помнить о двух вещах:

1. Цикл обязательно должен завершиться, поэтому тело цикла S должно влиять на <B>;
2. <B> должно состоять из корректных выражений и значений, определенных до начала цикла, чтобы не привело к ошибкам:

А) неверное число повторений цикла;

Б) получение бесконечного цикла.

Пример: Вычислить S= или S=1+1/22+1/32+…+1/n2

Программа (первый вариант):

Program p1;

Var S:real; I,n:integer;

Begin

Readln(n);

S:=0;

I:=1;

While i<=n DO

Begin

S:=S+1/(i\*i);

I:=i+1;

End;

Writeln(S);

End.

Второй вариант:

…

Begin

Readln(n);

S:=0;

I:=n;

While i>=1 DO

Begin

S:=S+1/(i\*i);

I:=i-1;

End;

Writeln(S);

End.

Оператор цикла с постусловием

Этот оператор такой же, как и While …DO, только условие проверяется после выполнения цикла, поэтому While …DO может не сделать ни одного цикла, а REPEAT…UNTIL хотя бы один цикл сделает.

Синтаксис:

REPEAT

<S1>;

<S2>;

…

<Sn>;

UNTIL <B>;

Где S1, S2, …,Sn – операторы, а <B> - логическое выражение.

Семантика:

B

Sn

S2

S1

…

False

True

Операторы S1, S2, …,Sn будут выполняться до тех пор, пока <B> ложно. Как только <B> станет истинным, цикл заканчивается и начинает работать оператор после UNTIL.

Бесконечные циклы:

WHILE TRUE DO …;

REPEAT … UNTIL FALSE;

Циклы могут вкладываться друг в друга.

Матрица и описание матрицы.

Пример: A(10,15)

a11 a12 a13 … a1,15 j

I a21 a22 a23 … a2,15

aij

a10,1 a10,2 … a10,15

Описание матрицы:

1. VAR A:array[1..10,1..15] of <тип>;
2. TYPE matr=array[1..10,1..15] of real;

VAR A:matr;

Обращение к элементу матрицы: A[1,2] A[i,j] A[2\*i, 3\*(j+1)]

Расположение в памяти:

1 строка 2 строка 10 строка

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A11 | A12 | A13 | … | A1,15 | A21 | A22 | … | A10,1 | A10,15 |

Задача1 Найти min сумму элементов строк (управляемый курсор)

Uses CRT;

GOTOXY (x,y);

Max по y = 25;

Max по x = 80;

Program P1;

Uses CRT;

Type MATR=array[1..10,1..15] of real;

VAR A:MATR;

i,j:integer; S,min:real;

Begin

Clrscr; {очистка экрана}

{ввод матрицы по строкам}

FOR i:=1 TO 10 DO

FOR j:=1 TO 15 DO

Begin

GOTOXY (5\*i+30, j+5);

Readln(A[i,j]);

End;

Min:=10E10;

FOR i:=1 TO 10 DO

Begin

S:=0;

FOR j:=1 TO 15 DO

S:=S+a[i,j];

IF S<min THEN min:=S;

Writeln(‘Сумма ‘,I,’ =’,S:6:2);

End;

Writeln(‘min Сумма =’,min:6:2);

End.

Задача 2 Найти три суммы элементов матрицы: 1) на главной диагонали; 2) под главной диагональю; 3) над главной диагональю.

1. Главная диагональ: a11, a22, a33, …, ann, т.е. i=j.
2. Под главной диагональю: a21,a32, …, т.е. i>j.
3. Над главной диагональю: a12,a23, …, т.е. i<j.

Program P2;

Uses crt;

Const m=20;

Type matr=array[1..m, 1..m] of real;

VAR A:matr;

I,j,n:integer;

S1,S2,S3:real;

Begin

Clrscr;

Write(‘введите n<=20 =’);

Readln(n);

Writeln(‘введи матрицу размером ‘,n,’\*’,n);

{ввод матрицы по столбцам}

FOR j:=1 TO n DO

FOR i:=1 TO n DO

Begin

GOTOXY(5\*j+30,i+5);

Readln(A[I,j]);

End;

S1:=0; S2:=0; S3:=0;

FOR i:=1 TO n DO

FOR j:=1 TO n DO

IF (i=j) THEN S1:=S1+a[I,j] ELSE IF (i>j) THEN S3:=S3+a[I,j] ELSE S2:=S2+a[I,j];

Writeln(S1, S2, S3);

End.

Лекция №8

Дана матрица A[10,10]. Найти сумму элементов под главной диагональю.

1 способ:

i>j

…

S1:=0;

FOR i:=1 to 10 DO

FOR j:=1 to 10 DO

IF (i>j) THEN S1:=S1+A[i,j];

Writeln(S1);

…

2 способ:

Внешний цикл 2<=i<=10 – перебор строк

Внутренний цикл 1<=j<=i-1 – перебор столбцов

…

S1:=0;

FOR i:=2 to 10 DO

FOR j:=1 to i-1 DO

S1:=S1+A[i,j];

…

3 способ:

…

S1:=0;

FOR j:=1 to 9 DO

FOR i:=j+1 to 10 DO

S1:=S1+A[i,j];

Writeln(S1);

…

4 способ:

…

S1:=0;

FOR i:=2 to 10 DO

FOR j:=i-1 DOWNTO 1 DO

S1:=S1+A[i,j];

Writeln(S1);

…

Найти сумму элементов под побочной диаганалью:

…

S:=0;

FOR j:=10 DOWNTO 2 DO

FOR i:=12-j TO 10 DO

S:=S+A[i,j]:

…

Досрочный выход из цикла

Дан массив {A}100 = {A1, A2, …, Ai, …, A100}

Определить индекс (номер) первого нулевого элемента.

начало

i

Нет 0

i>100

Ai=0

I=1

A100

I=i+1

Нет

Да

Нет

Да

I вариант:

Program P;

Uses CRT;

Label KON;

VAR A:array [1..100] of real;

I:integer;

Begin

Clrscr;

FOR i:=1 to 100 DO

Begin

Write(‘A[‘,I,’]=’);

Readln(A[i]);

End;

FOR i:=1 to 100 DO

IF (A[i]=0) THEN

Begin

Writeln(‘Номер=’,i);

GOTO KON;

End;

Writeln(‘НЕТ 0’);

KON: end.

Оператор EXIT прерывает выполнение программы и передаёт управление на конец процедуры или модуля.

II вариант:

Program P;

Uses CRT;

VAR A:array [1..100] of real;

I:integer;

Begin

Clrscr;

FOR i:=1 to 100 DO

Begin

Write(‘A[‘,I,’]=’);

Readln(A[i]);

End;

FOR i:=1 to 100 DO

IF (A[i]=0) THEN

Begin

Writeln(‘Номер=’,i);

EXIT;

End;

Writeln(‘НЕТ 0’);

End.

Оператор HALT завершает работу всей программы.

Метод совмещения условий

{$R+} – включить контроль границ диапазона.

{$R-} – сокращённое вычисление логических выражений.

…

Begin

<Ввод массива A(100)>

I:=1;

While (A[i]<>0) and (i<=100) DO

I:=i+1;

IF i>100 THEN writeln(‘нет 0’) ELSE

Writeln(‘номер ‘,i);

End.

i

Нет 0

i>100

I=i+1

A(100)

I=1

Начало

(A[i]<>0) and (i<=100)

Да Нет

Конец

Оператор break … continue

<заголовок цикла>

…

Break; {переход на оператор, который следует за

<конец цикла> циклом}

<заголовок цикла>

Continue; {переход на последний оператор в цикле}

…

<конец цикла>

Примеры:

Begin

…

K:=0;

FOR i:=1 TO 100 DO

begin

IF (A[i]=0) THEN begin

Writeln(‘i=’,i);

Break;

End;

IF k<100 THEN k:=k+1;

End;

Writeln(‘нет 0’);

end.

BEGIN

…

FOR i:=1 TO 100 DO

Begin

IF (A[i]<>0) THEN continue;

Writeln(‘i=’,i);

Exit;

End;

Writeln(‘нет 0’);

End.

Метод флажков (универсальный метод)

F=false

Нет 0

i>100

F=false

I=i+1

i

Ai=0

F

I=1

F=true

Нет

Да

Да нет

Нет да

Program P;

Var A:array [1..100] of real;

I:integer; F:Boolean;

Begin

<Ввод A>;

F:=true;

I:=1;

While F do

If (A[i]=0) Then begin

Writeln(‘i=’,i);

F:=false;

End else begin

I:=i+1;

If i>100 then begin

Writeln(‘нет 0’);

F:=false;

End; end;

End.

Сортировка одномерного массива A(100)

* По возрастанию (знак «<»);
* По убыванию (знак «>»).

Метод пузырька

Aj=X

Aj-1=Aj

X=Aj-1

Aj-1>Aj

J=100

I=2

A(100)

Да

A(100)

i>100

I=i+1

j>i

J=j-1

Да

Нет

нет

Да

Program P;

Var A:array [1..100] of real;

I,j:integer; x:real;

Begin

<Ввод A>;

For i:=2 to 100 do

For j:=100 downto I do

If (A[j-1] > A[j]) then begin

X:=A[j-1];

A[j-1]:=A[j];

A[j]:=X;

End;

<Вывод A>;

End.

Лекция №9

Процедуры

Основная программа

(головной модуль)

|  |  |
| --- | --- |
| Program … | |
| Обращение к процедуре | |
|  | |
| Обращение к процедуре | |
|  | |
| Процедура | |
| Описание процедуры |  | |  |

Бывают:

1. Внутренние процедуры;
2. Внешние процедуры.

Внутренняя процедура

Описание процедуры помещается в раздел описаний программы.

Program P;

Uses …

Const …

Label …

TYPE …

VAR …

Procedure <имя> [(<список параметров>)];

…

Begin {описание процедуры}

…

End;

Function <имя> (<список параметров>): <тип данных>;

…

Begin {описание функции}

…

End;

Begin {раздел операторов}

…

<обращение к процедуре>

…

End.

Процедура вызывается указанием имени и фактических параметров, функции вызываются в выражениях.

Внешняя процедура

Program P;

Uses mod1;

…

P1;

…

End.

Unit mod1;

…

Procedure P1;

…

End.

Процедуры различаются по типу:

1. Стандартные процедуры (функции, описание которых подключено непосредственно компилятором);
2. Procedure;
3. Function.

Обращение к процедуре является операндом. В списке фактических параметров можно задавать выражения, в процедуру будет передаваться значение выражения.

Описание процедуры – Procedure

Procedure <имя> [<список формальных параметров>];

{заголовок процедуры}

VAR

… <раздел описаний локальных имен>

Begin {тело процедуры}

… {раздел операторов}

End;

Обращение к процедуре:

<имя> [<список фактических параметров>)];

Обращение к процедуре является самостоятельным оператором. При обращении к процедуре список формальных параметров заменяется на список фактических параметров.

Процедуры без параметров

Дан массив A(20). Найти в нем min и max.

Program P;

Const n=20;

TYPE mas=array [1..n] of real;

VAR A:mas; i:integer; min,max:real;

Procedure minmax;

VAR i:1..n;

Begin

Min:=A[1];

Max:=A[1];

FOR i:=2 TO n DO

Begin

IF (A[i] > max) THEN max:=A[i];

IF (A[i] < min) THEN min:=A[i];

End;

End;

Лекция № 10

Связь формальных и фактических параметров

При описании процедуры мы задаем формальные параметры, а при обращении к процедуре – фактические параметры.

Между формальными и фактическими параметрами должно существовать взаимно-однозначное соответствие (в обе стороны).

Соответствие устанавливается:

1. Количеством параметров;
2. Строгим порядком следования;
3. Тип данных параметров должен совпадать.

Пример:

PROCEDURE NAME (a, b: real);

NAME (2.0, 3.5);

Входные Procedure NAME выходные

Параметры VAR параметры

Program P;

VAR a, b: integer;

Procedure P1 (x: integer; VAR y: integer);

Begin

X:= x+1;

Y:= y+1;

End;

Begin

A:= 0;

B:= 0;

P1 (a, b);

Writeln (‘a=’,a,’ b=’,b);

End.

A=0 B=1

Для входных параметров (параметры-значения) в процедуру передается значение фактического параметра и в процедуре для него выделяется локальная область памяти.

Для выходных параметров (параметры-переменные) в процедуре никакой памяти не выделяется, а передается в процедуру адрес фактического параметра.

P1 (0, 0) – второй 0 – это ошибка!

В качестве фактического параметра можно использовать выражение, если ему соответствует входной параметр.

Процедуры-функции

Описание функции:

Function <имя>[(<список формальных параметров>)]: <тип данных>;

<описание локальных параметров>

Begin

<операторы>

End;

Отличия функции от процедуры:

1. Функция имеет тип данных, в то время как процедура общего вида типа не имеет;
2. В теле функции должен встретиться хотя бы один оператор, в котором имени функции присваивается значение;
3. Вызов функции происходит в выражении (оператор «присвоить значение»), а вызов процедуры - просто указанием имени и параметров.

Пример:

VAR y:real;

PROCEDURE SUM (a, b: real; VAR y: real);

Begin

Y:= a+ b;

End;

Begin

SUM(2.0, 3.0, y);

Writeln (y);

End.

VAR y:real;

Function SUM(a, b :real): real;

Begin

SUM:= a+b;

End;

Begin

Y:= SUM(2.0, 3.0); {можно writeln (SUM(2.0, 3.0)); }

Writeln (y);

End.

Задача 2: Найти сумму одномерного массива B(50) – S=

В процедуре посчитать y=

Program P;

Const m=50;

Type MAS=array [1..m] of real;

VAR B: MAS; j: integer; S: real;

PROCEDURE SUMMA (A:MAS; n: integer; VAR y: real);

VAR i: integer;

Begin

Y:=0;

FOR i:=1 TO n DO

Y:= y + A[i];

End;

Begin {начало программы}

FOR j:=1 TO m DO

Begin

Write (‘B [‘, j, ‘]=’);

Readln (B[j]);

End;

SUMMA (B, m, S);

Writeln (‘Сумма = ‘, S);

End.

Глобальные и локальные имена

Глобальные имена объявляются во внешнем программном блоке и распространяют свое действие на все внутренние блоки, за исключением тех блоков, в которых имеется собственное описание данного имени.

Локальные имена действуют только в том блоке, в котором они описаны. При выходе из блока они теряют свое значение.

Процедура с параметрами

Пример

Даны 2 одномерных массива A и B. Найти в каждом из них min и max.

Program P;

Const n=20;

Type MAS=array [1..n] of real;

VAR A, B: MAS; i: integer;

mina, minb, maxa, maxb: real;

PROCEDURE Minmax (C: MAS; VAR min, max: real);

VAR i: integer;

Begin

Min:= C[1];

Max:= C[1];

FOR i:= 2 TO n DO

Begin

IF (С[i] > max) THEN max:= С[i];

IF (С[i] < min) THEN min:= С[i];

End;

End;

Begin {Начало программы}

FOR i:=1 TO n DO

Begin

Write (‘A[‘, I, ‘]=’);

Readln (A[i]);

End;

FOR i:=1 TO n DO

Begin

Write (‘B[‘, I, ‘]=’);

Readln (B[i]);

End;

Minmax (A, mina, maxa);

Minmax (B, minb, maxb);

Writeln (‘mina=’, mina,’ maxa=’, maxa);

Writeln (‘minb=’, minb,’ maxb=’, maxb);

End.

**Лекция №11**

**Функциональный тип данных. Использование функции в качестве параметра процедур.**

Задача №1

Y= { sin x2, если x<0

Cos ln x, если x>=0

Опция компилятора {$F+} – удалённый вызов процедур.

Program P;

{$F+}

Type fun = Function (x: real): real;

Function F1 (x: real): real;

Begin

F1:=sin (x\*x);

End;

Function F2 (x: real): real;

Begin

F2:= cos (ln (x));

End;

Var

X, y: real;

F: fun;

Begin

Write (‘Введи x’);

Readln (x);

IF x<0 then F:=F1 else F:=F2;

Y:=F(x);

Writeln (x, y);

End.

Задача 2: Найти интеграл функции

I вариант: отрезок от А до В с шагом h

{$F+}

Program P;

Type FUN=Function (x: real): real;

Function F1(x: real): real;

Begin

F1:= sin(x)/x;

End;

Procedure Integral (A, B, H: real; F: Fun; Var S: real);

Var x: real;

Begin

X:= A;

S:= 0;

While x<B do begin

S:= S+ f(x);

X:=x+H;

End;

S:= S\*H;

End;

Var W: real;

Begin {начало программы}

Integral (0.1, 3.0, 0.1, F1, W);

Writeln (W);

End.

II вариант

…

Begin

F1:= sin(x)/x;

End;

Function Integral (A, B, H: real; F:Fun): real;

Var x, S: real;

Begin

X:= A;

S:= 0;

Repeat

S:= S+f(x);

X:= x+ H;

Until x>B;

Integral:=S\*H;

End;

Begin {главная программа}

Writeln (Integral (0.1, 3.0, 0.1, F1):8:4);

End.

III вариант

…

Function Integral (A, B, H: real; F: Fun): real;

Var I, n: integer;

X, S: real;

Begin

N:= TRUNC ((B-A)/H);

For i:= 1 to n do begin

S:= S+ F(x);

X:= x+H;

End;

Integral:= S\*H;

End;

IV вариант ( посчитать интеграл с заданной точностью)

…

Var A, B, S1, S2, h, d: real; n: integer;

Const eps=0.000001;

Begin {начало программы}

S1:= 0;

H:= (B-A)/2;

N:= 0;

Repeat

N:= n+1;

S2:= Integral (0.1, 3.0, h, F1);

H:= h/2;

D:=abs(S2-S1);

S1:=S2;

Until d<=eps;

Writeln (‘Интеграл = ‘, S2:9:4, ‘ число итераций = ‘, n);

End.

Задача 3

Найти корень уравнения методом деления отрезка пополам.

Задана F(x)=0, задан отрезок [a, b], корень внутри этого отрезка.

Правило уточнения приближения: xs=(a+b)/2

Ещё 2 правила:

1. Если f(xs) и f(B) имеют разные знаки, то a=xs
2. Если одинаковые знаки, то b=xs

Правило остановки: |a-b| <= eps

Procedure kor (a, b, eps: real; f:fun; var x:real);

Var xs, d: real;

Begin

Repeat

Xs:= (a+b)/2;

If (f(xs)\*f(B) < 0) then a:= xs;

If (f(xs)\*f(B) > 0) then b:= xs;

D:= abs (a-b);

Until d <= eps;

X:= xs;

Writeln (‘Корень = ‘, x:9:4);

End;

**Лекция №12**

**Файлы – совокупности данных**

Файл – это однотипная совокупность данных, имеющая имя, расположенная последовательно на внешнем носителе. Операция ввода – это копирование данных с внешнего запоминающего устройства (ВЗУ) в оперативную память (ОП). Операция вывода – наоборот: из ОП в ВЗУ. Ввод – это read, а вывод – write.

Read [ln] [([< имя файла >,] < список объектов в/в > [:F[:D]])];

Write

Ln – только для текстовых файлов.

Примеры: read (a, b, c); read (CON, a, b, c);

CON – это консоль.

Если имя файла опущено, то подразумевается, что вывод идёт на экран.

Writeln (PRN, ‘HELLO’);

Uses Printer – для вывода на печать программы.

Каждый элемент ввода-вывода можно дополнять форматом, показывающим количество выделяемого места. Формат F указывает, сколько символов всего, а формат D – сколько символов после десятичной запятой.

Примеры: writeln (2.5:7:3); - выравнивание по правому краю: 2.500

Writeln(2.5:-7:3); - выравнивание по левому краю:2.500 .

Writeln(true:3); - tru

Writeln(false:8); - false .

Read (i);

If i=0 then b:= false else b:= true;

У файла есть метка начала файла и метка конца файла. Функция определения конца файла - EOF (f) принимает два значения: либо TRUE, либо FALSE.

Пример: i:= 1;

While not EOF (f) do

Begin

Read(f, A[i]);

I:= i+1;

End;

В языке Pascal выделяется 3 вида файлов:

1. Текстовый файл – тип TEXT;
2. Типизированный файл – тип FILE OF <тип>;
3. Нетипизированный файл – тип FILE.

Примеры:

Type product=record

Name: string;

Code: integer;

Cost: real

End;

Var f1: text;

F2: file of char;

F3: file of product;

F4: file;

Доступ к файлам:

Assign (<ф.п.>, <имя файла или л.у.>);

Связывание файловой переменной с именем физического файла или логического устройства, например:

Assign (f, ’A:\IVANOV\NAME.DAT’);

Открыть файл в Pascal можно для чтения, для записи, а также для чтения и записи одновременно.

Открытие файла для чтения – RESET (<ф. п.>);

Например: RESET (f);

Открытие файла для записи – REWRITE (<ф. п.>);

Например: REWRITE (f);

Для текстовых файлов, открытых для чтения, нельзя использовать оператор WRITE[LN], а для открытых на запись – нельзя использовать существующие файлы (их содержимое будет уничтожено). Если нужно добавить текст в существующий файл, то используется процедура APPEND (<ф. п.>);

Для закрытия файла используется процедура CLOSE (<ф. п.>); эта процедура обеспечивает сохранение данных, если файл был открыт процедурами REWRITE или APPEND. После закрытия файл можно снова открыть без использования ASSIGN.

Для переименования файла используется процедура RENAME (<ф. п.>, <новое имя>); например:

RENAME (f, ‘A:\PETROV\FIO.DAT’);

Процедура ERASE (<ф. п.>); уничтожает файл. Существуют и другие процедуры и функции.

В типизированных файлах длина любого компонента строго постоянна. При открытии файла указатель стоит на первом компоненте с номером 0 (в случае с APPEND после последнего компонента).

Процедура READ обеспечивает чтение очередных компонент:

READ (<ф. п.>, <список ввода>);

Процедура WRITE записывает очередные компоненты в типизированный файл:

WRITE (<ф. п.>, <список вывода>);

Процедура SEEK смещает указатель к требуемому компоненту:

SEEK (<ф. п.>, <№ компонента>);

Например: SEEK (f, 8);

Вывод в файл:

VAR f: file of integer;

Begin

Assign (f, ‘NAME’);

Rewrite (f);

Write (f, 25);

Write (f, 37);

…

Seek (f, 5);

Write(f, 30);

…

Close (f);

Ввод из файла:

VAR f: file of integer;

I: integer;

Begin

Reset (f);

Read (f, i);

Пример:

Program P;

VAR f: file of integer;

I, n: integer;

Begin

Assign (f, ‘D:\TTT.TXT’);

Rewrite (f);

FOR i:= 1 TO 10 do

Begin

N:=sqr (i);

Write (f, n);

End;

Close (f);

Reset (f);

FOR i:=1 TO FILESIZE (f) do

Begin

Read (f, n);

Writeln (n: 5);

End;

Close (f);

End.