# Краткий конспект к курсу «Введение в программирование» (Основы Паскаль)

*Оглавление*[\_Toc502783495](#_Toc502783495)

[1. Алгоритм и алгоритмизация. Блок-схемы. Неформальная и формальная постановки задачи. 3](#_Toc502783496)

[2. Системы счисления. Представление целых чисел в памяти компьютера. Арифметические и логические операции. Целочисленные типы данных в ЯП Паскаль. 10](#_Toc502783497)

[3. Представление вещественных чисел в памяти компьютера. Выполнение операций с нормализованными числами. 12](#_Toc502783498)

[4. Типы данных в ЯП Паскаль. Структура программы на ЯП Паскаль. Линейный алгоритм. Алгоритмы с ветвлением. 17](#_Toc502783499)

[5. Вычисления, стандартные функции, управляющие операторы, Работа с файлами в ЯП Паскаль. Условный оператор. Таблица истинности. Сложные условия. Множественный выбор. 18](#_Toc502783500)

[6. Цикл с переменной. Цикл с предусловием. Цикл с постусловием. Вложенные циклы. 23](#_Toc502783501)

[7. Процедуры и функции. Передача параметров по ссылке и по значению. 24](#_Toc502783502)

[8. Алгоритмическая сложность. Асимптотическая сложность. Понятия О-большое. 25](#_Toc502783503)

[9. Массивы. Перебор элементов массива. 26](#_Toc502783504)

[10. Кодирование символов. Символьные строки. Функции для работы со строками. Преобразование «строка-число». Операции над строковым типом данных. 27](#_Toc502783505)

[11. Двумерные массивы. Матрицы. 28](#_Toc502783506)

[12. Перечисляемый тип данных. Интервальный тип данных. 29](#_Toc502783507)

[13. Структуры. Записи с вариантами. 30](#_Toc502783508)

[14. Понятие указателя. Работа с указателями в ЯП Паскаль: типизированные и нетипизированные указатели, разыменование, взятие адреса, арифметика указателей. Динамические массивы. 31](#_Toc502783509)

[15. Динамические структуры данных. Линейный список. Очередь. Стек. Дек. 38](#_Toc502783510)

[16. Рекурсия. 39](#_Toc502783511)

[17. Модули. Раздельная компиляция. 44](#_Toc502783512)

[18. Уточнение понятия алгоритм. Машина Тьюринга. Машина Поста. 51](#_Toc502783513)

[19. Алгоритмически неразрешимые задачи. 56](#_Toc502783514)

# 1. Алгоритм и алгоритмизация. Блок-схемы. Неформальная и формальная постановки задачи.

**Алгоритм решения задачи** - это последовательность допустимых команд исполнителя, определяющих его действия по переходу от исходных данных к искомому результату.

Главная особенность любого алгоритма - формальное исполнение, позволяющее выполнять заданные действия (команды) не только человеку, но и техническим устройствам (исполнителям). Таким образом, исполнителями алгоритмов могут быть, например, человек, компьютер, принтер, робот-манипулятор, станок с числовым программным управлением. Каждый исполнитель умеет выполнять определенные команды, т.е. обладает системой команд исполнителя (СКИ).

**Свойства алгоритма:**

1. *дискретность* - алгоритм делится на отдельные элементарные шаги;
2. *определенность* - каждая команда однозначно определяет действие исполнителя;
3. *конечность*(результативность) - алгоритм должен завершаться за конечное число шагов.

Кроме этого, алгоритм может обладать еще одним полезным (но не обязательным) свойством - *массовостью*. Это значит, что он будет годиться не для одной конкретной задачи, а для целого класса похожих задач.

Различают три основных типа алгоритмов:

* Линейные (следования);
* Разветвляющиеся (развилка);
* Циклические.

Во многих задачах искомые результаты из исходных данных можно получить без проверки выполнения каких бы то ни было условий. Алгоритмы решения таких задач получили название линейных алгоритмов.

**Линейный** – это такой алгоритм, в котором все команды выполняются строго последовательно друг за другом.

*Способы записи алгоритма:*

1. Запись на естественном языке.
2. Запись алгоритма на специальном алгоритмическом языке;
3. Графический способ записи (блок-схема). Позволяет представить каждый шаг алгоритма в виде геометрических фигур, соединенных между собой линиями, указывающие последовательность выполнения шагов.

начало/конец алгоритма

действие

ввод/вывод данных

Например, алгоритм деления обыкновенных дробей может быть записан следующими способами:

а) запись на естественном языке:

1. Числитель первой дроби умножить на знаменатель второй;
2. Знаменатель второй дроби умножить на числитель второй;
3. Записать дробь, числитель которой есть результат выполнения пункта 1, а знаменатель - результат выполнения пункта 2.

б) запись на алгоритмическом языке: в) блок- схема

Начало

Ввод a, b, c, d

m := a \* d

n := b \* c

Конец

Вывод m/n

Исходные данные:  - делимое;  - делитель

Результат: 

алг Деление дробей

нач

ввод a,b,c,d

m:=a\*d

n:=b\*c

    вывод m/n

кон

Если величина в ходе выполнения алгоритма может принимать различные значения, то она называется *переменной*. В данном алгоритме использовались переменные a, b, c, d, m, n. Переменная - это область в памяти компьютера, где хранится некоторое значение. Каждая переменная имеет имя, значение и тип. Имя переменной обозначается английской буквой, после которой может быть цифра, причем первой стоит буква, например: x, y, z, A1, B2.

**Пример 1.**

Алгоритм вычисления периметра и площади прямоугольника по заданным длинам сторон (a=6, b=4).

b=4

a=6

S=a\*b

P=(a+b)\*2

вывод S,P

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Шаг | Команда | Значения переменных | | | |
| a | b | S | P |
| 1 | a=6 | 6 | - | - | - |
| 2 | b=4 | 6 | 4 | - | - |
| 3 | S=a\*b | 6 | 4 | 24 | - |
| 4 | P=(a+b)\*2 | 6 | 4 | 24 | 20 |

**Пример 2.**

Алгоритм вычисления дискриминанта квадратного уравнения

Трассировочная таблица составлена для a=1, b=4, c=2.

Ввод a, b, c

D=b2–4\*а\*с

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Шаг** | **Команда** | **Значения переменных** | | | |
| **a** | **b** | **c** | **D** |
| 1 | Ввод a, b ,c | 1 | 4 | 2 | - |
| 2 | D=b2–4\*а\*с | 1 | 4 | 2 | 8 |

Вывод D

**Пример 3.**

Алгоритм обмена значений двух переменных А и В.

В данном алгоритме используется вспомогательная переменная Т.

Трассировочная таблица составлена для А=7, В=10.

Т=А

А=В

В=Т

вывод А,В

ввод А,В

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Шаг** | **Команда** | **Значения переменных** | | |
| **А** | **В** | **Т** |
| 1 | Ввод А, В | 7 | 10 | - |
| 2 | Т=А | 7 | 10 | 7 |
| 3 | А=В | 10 | 10 | 7 |
| 4 | В=Т | 10 | 7 | 7 |

**Пример 4.**

Алгоритм обмена значений переменных А и В без использования вспомогательной переменной (трассировочная таблица составлена для А=3, В=8)

|  |  |  |  |
| --- | --- | --- | --- |
| **Шаг** | **Команда** | **Значения переменных** | |
| **А** | **В** |
| 1 | Ввод А, В | 3 | 8 |
| 2 | А=А+В | 11 | 8 |
| 3 | В=А-В | 11 | 3 |
| 4 | А=А-В | 8 | 3 |

А=А+В

В=А-В

А=А-В

вывод А,В

ввод А,В

**Пример 5.**

Алгоритм вычисления а8 , с использованием только операций умножения.

(трассировочная таблица составлена для а=2)

|  |  |  |
| --- | --- | --- |
| **Шаг** | **Команда** | **Значение а** |
| 1 | Ввод а | 2 |
| 2 | а=а\*а | 4 (а2 ) |
| 3 | а=а\*а | 16 (а4 ) |
| 4 | а=а\*а | 256 (а8 ) |

а=а\*а

а=а\*а

а=а\*а

вывод а

ввод а

**Разветвляющиеся алгоритмы**

Алгоритмы линейной структуры на практике встречается крайне редко. Чаще необходимо организовать процесс, который в зависимости от каких-либо условий проходит по той либо иной ветви алгоритма. Такой алгоритм называется ***разветвляющимся***.

В блок-схемах ветвление обозначается следующим блоком:

условие

команда1

команда2

да

нет

Если условие выполняется, то исполняется **команда1**, иначе – **команда2**. Такое ветвление называется полным. Неполное ветвление обозначается следующим образом:

условие

команда1

да

нет

Пример.

Вычислить y=1/x для введенного х.

ввод Х

х=0

Неверный ввод данных

Y:=1/x

вывод Y

+

-

**Контрольные вопросы**

1. Что такое алгоритм? Какими свойствами он обладает?
2. Какими способами может изображаться алгоритм?
3. Какой алгоритм называется линейным?
4. Изобразите основные блоки, составляющие блок-схему линейного алгоритма, и поясните их назначение.
5. Какой алгоритм называется разветвляющимся?
6. Изобразите основные блоки, составляющие блок-схему алгоритма с полным ветвлением, и поясните их назначение.
7. Изобразите основные блоки, составляющие блок-схему алгоритма с неполным ветвлением, и поясните их назначение.

Литература:

1. В Blackboard: Курс «Практикум на ЭВМ» -> Раздел «Конспекты лекций» -> Методичка «Практикум на ЭВМ», стр. 1-3

<https://bb.dvfu.ru/webapps/blackboard/execute/content/file?cmd=view&mode=designer&content_id=_92799_1&course_id=_2759_1>

1. Презентация «Паскаль\_Основы» 2-27 слайды

# 2. Системы счисления. Представление целых чисел в памяти компьютера. Арифметические и логические операции. Целочисленные типы данных в ЯП Паскаль.

Выражения представляют собой конструкцию языка, которая содержит в себе данные, операции и имеется строгий порядок выполнения этих операций. Выражения всегда возвращают какое-либо значение.

**Операнды** — это данные, над которыми производятся действия. Операндами могут быть переменные, константы, переменные массивов и другие элементы.

**1. Арифметические операции**

Арифметические операции выполняются над целыми и действительными числами. Результатом выполнения арифметического выражения является, соответственно, целое или действительное число.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Знак Операция | Операции | Типы операндов | Тип результата | Пример |
| + | Сложение | Integer, Real | Integer, Real | x+y |
| — | Вычитание | Integer, Real | Integer, Real | x-y |
| \* | Умножение | Integer, Real | Integer, Real | x\*y |
| / | Деление | Integer, Real | Real | x/y |
| div | Целочисленное деление | Integer | Integer | X div Y |
| mod | Целочисленный остаток от деления | Integer | Integer | X mod Y |

**Операций div и mod:**  
• результатом целочисленного деления x div Y будет результат деления x на Y, округленный в сторону нуля до ближайшего целого числа. Например, результатом выражения 4 div 2 будет 2, а результатом выражения 17 div 4 будет 4;  
• результатом операции mod будет остаток от целочисленного деления операндов. Таким образом, выражение x mod у эквивалентно выражению x-(x div Y) \*Y. Например, результатом выражения 4 mod 2 будет о, а результатом выражения 9 mod 4 будет 1.

Пример

Var a, b, c, d: integer;  
Begin a: = 7; b: = 5; c: = -10;  
d: = a+b; d: = -d; end;

**2. Логические операции**

**Логические выражения** — это выражения, в которых содержатся:

1. логические константы True И (ИЛИ) False;

2. логические переменные и (или) константы типа Boolean;

3. операции отношения;

4. логические операции;

5. скобки.

В результате выполнения логической операции получается логическое значение True или False.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Знак Операция | Операции | Типы операндов | Тип результата | Пример |
| not | Отрицание | Boolean | Boolean | not(C in MySet) |
| and | Конъюнкция (логическое И) | Boolean | Boolean | A and В |
| or | Дизъюнкция (логическое ИЛИ) | Boolean | Boolean | A or В |
| xor | Исключающая дизъюнкция (исключающие ИЛИ) | Boolean | Boolean | A xor В |

Для того чтобы был четкий порядок выполнения операции, в языке Delphi имеется понятие приоритет.

Приоритет определяет старшинство операций, служит для установления порядка выполнения операций. Операции с более высоким уровнем приоритета выполняются раньше остальных.

По приоритету все операции разделяются на четыре уровня:

По первый (наивысший) уровень приоритета. К нему относятся операции and и not;  
• второй уровень приоритета — операции \*, /, div, mod, and, shl, shr и as;  
• третий уровень приоритета — операции +, -, or и хоr;  
• четвертый (самый низкий) уровень приоритета — операции =, о, <, >,<=, >=, in и is.

Если подряд идут несколько операций с одинаковым приоритетом, то операции выполняются слева направо.

Литература:

1. <http://computer-lectures.ru/osnovnye-ponyatiya-informatiki/1-3-sistemy-schisleniya/>
2. Презентация «Представление информации в компьютере» 14-39 слайды.

# 3. Представление вещественных чисел в памяти компьютера. Выполнение операций с нормализованными числами.

Представление вещественных чисел в памяти компьютера

Вещественные числа имеют две формы представления: с фиксированной запятой и плавающей запятой.

**Числа в формате с фиксированной запятой** имеют вид:

123456.789

целая часть дробная часть

Для хранения целой и дробной части отведено фиксированное число разрядов.

Такой формат обозначается **(M.N)**, где **М** – число разрядов целой части, включая знак числа, **N** – число разрядов дробной части.

Представление чисел в таком формате используется в программах, работающих с данными заранее известного ограниченного диапазона (например, формат широко распространён в литературе по цифровой обработке сигналов и соответствующих приложениях).

Достоинства формата – простота представления и реализации арифметических операций, недостатки – ограниченный диапазон.

**Числа в формате с плавающей запятой представляются в виде:**

Xp = ± M\*p ± k

p – основание системы счисления, M – мантисса, ± k - порядок числа

k – целое число, которое показывает, на сколько разрядов сдвигается запятая, а по знаку определяется направление сдвига.

Примеры:

-1234,8910 = -12, 3489\*102 = -12348,9\*10-1

1010, 0112 = 1,010011\*23 = 101001,1\*10-2

Для однозначности представления для мантиссы задаётся допустимый диапазон:

**1 ≤ Мp < 10p**

Например, физические константы в системе СИ записываются с использованием этого диапазона:

скорость света c = 2,99793\*108 м/с

масса покоя электрона me = 9,10956\*10-31 кг

Процедура приведения мантиссы к допустимому диапазону называется **нормализацией**. После нормализации числа будут иметь вид:

123,4510 = 1,2345\*102

1110011, 110012 = 1,11001111001\*26

-0,001100112 = -1,10011\*2-3

Нормализация мантиссы позволяет сэкономить один разряд в её двоичном представлении: после нормализации любое двоичное число будет иметь вид «1,…..», то есть заранее известно, что первый значащий разряд равен 1, поэтому его можно не хранить (это называется использованием **неявного старшего бита**).

**Вещественные числа в Паскале:**

|  |  |  |
| --- | --- | --- |
| Тип | Память | Диапазон |
| SINGLE  одинарная точность | 4 байта = 32 бита | ± 10-38 …. ± 10+38  7 значащих цифр мантиссы |
| REAL | 6 байт = 48 бит | ± 10-45 …. ± 10+48  11 значащих цифр мантиссы |
| DOUBLE  двойная точность | 8 байт = 64 бита | ± 10-308 …. ± 10+308  15 значащих цифр мантиссы |
| EXTENDED  повышенная точность | 10 байт = 80 бит | ± 10-4932 …. ± 10+4932  19 значащих цифр мантиссы |

Диапазон задаётся с точностью до множителя и показывает возможные порядки записываемых в этом типе вещественных чисел.

**(**

**(**

**)**

**)**

**-10-38**

**+10-38**

**-10+38**

**+10+38**

**0**

Представление вещественного числа в заданном типе зависит от количества разрядов, которые отводятся на хранение мантиссы и порядка.

В формате **с одинарной точностью** (тип SINGLE в языке Pascal, FLOAT в языке C) из 32 бит отводится 1 бит для записи знака числа (знаковый бит), 8 бит для смещённого порядка (смещение равно +127), 23 бита для мантиссы без первой 1.

В формате **с двойной точностью** (тип DOUBLE в языках Pascal и C, основной формат хранения данных в MatLab) из 64 бит отводится 1 бит для знака, 11 бит для смещённого порядка (смещение равно +1024), 52 бита для мантиссы без первой 1

В формате **с расширенной точностью** (тип EXTENDED в языке Pascal, LONG DOUBLE в языке C) из 80 бит отводится 1 бит для знака, 15 бит для хранения порядка, 64 бита для мантиссы

**Представление в формате с одинарной точностью** (тип SINGLE):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  | ………………….. |  |  |
| знак | 8 бит  смещённый порядок | | | | | | | | | 23 бита  мантисса без первой единицы | | | | |

Чтобы найти представление вещественного числа в типе SINGLE , нужно:

1. перевести число в двоичную систему счисления;
2. нормализовать мантиссу (привести к виду « ± 1,… \*2 k»)
3. вычислить смещённый порядок kсмещ = k +127
4. записать знаковый бит (0 - для неотрицательных чисел, 1 - для отрицательных чисел);
5. записать двоичное восьмибитное представление смещённого порядка;
6. записать значение мантиссы без первой единицы (т.е. все разряды нормализованного числа после запятой); если число значащих цифр меньше, чем 23 разряда, дополнить нулями, если больше, чем 23 разряда, то выполнить округление по правилам двоичной системы.

**Примеры:**

1) Найти внутреннее двоичное и 16-ричное представление вещественного числа 16,510 в типе SINGLE.

16,510 = 10000,12= 1,000012\*24

k = 4, kсмещ = 4 +127 = 100000112

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0…………..0 |

Двоичное представление переведём в 16-ричное, разделим 32 бита на блоки по 4 двоичных символа (тетрады), получим: **41 84 00 00**

2) Найти внутреннее двоичное и 16-ричное представление вещественного числа -20,610 в типе SINGLE.

-20,610 = -10100,1001(1001)2= - 1,0100(1001)2\*24

k = 4, kсмещ = 4 +127 = 100000112

При переводе получили бесконечную периодическую дробь, повторяем период, пока не заполнятся все 23 разряда мантиссы, и проверяем значение первого отбрасываемого разряда:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1001 1001 …1001 10**0** | **1** |

Первый отбрасываемый разряд равен 1, следовательно, при выполнении округления увеличиваем младщий разряд на 1 (если первый отбрасываемый разряд равен 0, младший разряд ооставляем без изменений).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1001 1001 …1001 10**1** |  |

Делим на блоки по 4 двоичных символа, получаем 16-ричное представление: **C1 A4 CC CD**

3) Обратная задача. Найти вещественное число по его внутреннему 16-ричному представлению в типе SINGLE: **BF 30 00 00**

Двоичное представление: 1011 1111 0011 0000 ….0000

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0…………..0 |

Число будет отрицательным, следующие 8 разрядов – это смещенный порядок kсмещ = 011111102 =126, тогда k = kсмещ -127= -1

Записываем 1 и после запятой – цифры мантиссы, переносим запятую в соответствии с вычисленным порядком на 1 позицию влево, полученную двоичную дробь переводим в десятичную систему счисления:

- 1, 01100…02\*2-1 = - 0, 10112 = - 0,6875

**Диапазон представимых вещественных чисел** вычислить труднее, чем в целых типах, нужно учитывать минимальные и максимальные значения, как для мантиссы, так и для порядка.

Сделаем приблизительную оценку для типа SINGLE. **Наибольшее по модулю число** будет иметь вид:

1,1……….1\*2127 = 0,1…..….1\*2128 ≈ 1\*2128 ≈ (210)12 28 ≈ **1038**

23 разряда 24 разряда

**Количество вещественных чисел**, представленных в памяти компьютера в типе SINGLE:

N= 2t \* (U – L+1) \* 2 = 223 \*256\*2 = 232 ≈ (210)3 \*4 ≈ **4\*109**

t – количество разрядов мантиссы

U – максимальное значение порядка

L – минимальное значение порядка

2 – учитываем, что число может быть положительным и отрицательным

Литература:

1. Презентация «Представление информации в компьютере» 43-53 слайды.
2. <https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2%D0%B5%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB>

# 4. **Типы данных в ЯП Паскаль. Структура программы на ЯП Паскаль. Линейный алгоритм. Алгоритмы с ветвлением**.

Литература:

1. <http://fkn.ktu10.com/?q=node/8504>
2. Презентация «Паскаль\_Основы» 2-48 слайды.

# 5. Вычисления, стандартные функции, управляющие операторы, Работа с файлами в ЯП Паскаль. Условный оператор. Таблица истинности. Сложные условия. Множественный выбор.

Язык программирования Паскаль (назван в честь выдающегося французского математика и философа Блеза Паскаля (1623 — 1662)), разработан в 1968 — 1971 гг. Н.Виртом. Язык Паскаль, созданный первоначально для обучения программированию как систематической дисциплине, скоро стал широко использоваться для разработки программных средств в профессиональном программировании.

Язык программирования Паскаль отражает фундаментальные и наиболее важные концепции (идеи) алгоритмов в очевидной и легко воспринимаемой форме, что предоставляет программисту средства, помогающие проектировать программы. Язык Паскаль позволяет четко реализоватьидеи *структурного программирования* и *структурной организации данных*. Язык Паскаль сыграл большую роль в развитии методов аналитического доказательства правильности программ и позволил реально перейти от методов отладки программ к системам автоматической проверки правильности программ.

Разработка программ на Паскале включает в себя следующие действия (этапы разработки программы): ввод и редактирование текста программы на языке программирования Паскаль, ее трансляцию, отладку.

Для выполнения каждого этапа применяются специальные средства: для ввода и редактирования текста используется *редактор текстов*, для трансляции программы - *компилятор*, для построения исполняемого компьютером программного модуля с объединением разрозненных откомпилированных модулей и библиотекой стандартных процедур Паскаля - *компоновщик (linker),* для отладки программ с анализом ее поведения, поиском ошибок, просмотром и изменением содержимого ячеек памяти компьютера - *отладчик (debugger).*

Систему программирования Турбо Паскаль называют ***интегрированной*** (integration — объединение отдельных элементов в единое целое) средой програм­мирования, так как она объединяет в себе возможности ранее разрозненных средств, используемых при разработке программ: редактора текстов, компилятора, компоновщика, отладчика, и при этом обеспечивает программисту великолепные сервисные возможности. Часто ее кратко называют *IDE (Integrated Development Environment - интегрированная среда разработки).*

**Операторы присваивания, ввода и вывода.**

Команда *присвоения*: **переменная:= выражение;**

Например, а:=35

x:=x+1

Команда *вывода данных:* **WRITELN (<список>);**

Список составляют переменные, константы, выражения, разделенные запятой.

Например, WRITELN (a, b);

WRITELN (‘S=’ , S);

WRITELN (‘Сумма чисел равна ‘, a+b);

*Команда ввода данных:* READLN (переменная1, переменная2,…);

Например, READLN (a);

READLN (a, b,c);

**Пример 1.**

Вычисление суммы чисел A и B.

PROGRAM PR1;

VAR A, B, C:INTEGER;

BEGIN

READLN (A, B);

C: =A+B;

WRITELN (C);

END.

**Пример 2**

Вычисление дискриминанта квадратного уравнения.

PROGRAM PR2;

VAR

A, B, C:INTEGER;

BEGIN

READLN (A, B, C);

D: =B\*B-4\*A\*C;

WRITELN (‘D=’,D);

END.

**Пример 3.**

Вычисление гиперболического косинуса gc= 

Program pr2;

Var x: integer;

gc: real;

begin

readln(x);

gc:=(exp(x)+exp(-x))/2;

writeln('Гиперболический косинус=',gc);

end.

**Пример 4.**

Вычисление суммы цифр введенного натурального двузначного числа.

Program pr4;

Var n, a, b: integer;

begin

write('Введите число '); readln(n);

a:=n div 10;

b:=n mod 10;

writeln('Сумма цифр = ', a+b);

end.

**Пример 5.**

Вводится число а. Не пользуясь никакими арифметическими операциями, кроме умножения, получить a6 за три операции.

Program pr5;

Var a, b: integer;

begin

write('Введите число '); readln(a);

a:=a\*a;

b:=a;

a:=a\*a;

a:=a\*b;

writeln( a);

end.

**Пример 6.**

Обмен значениями переменных x и y.

Program pr6;

Var x, y, t: integer;

begin

write('Введите x, y '); readln(x,y);

t:=x;

x:=y;

y:=t;

writeln(‘x=’, x, ‘y=’,y);

end.

**Контрольные вопросы**

1. Запишите общий вид оператора вывода. Что можно указывать в качестве элементов списка вывода? Какой символ используется для разделения элементов списка вывода?
2. Что будет выведено на экран, если в списке вывода записано:

а) число;

б) имя величины;

в) текст в кавычках;

г) арифметическое выражение?

1. Запишите общий вид оператора ввода. Что можно указывать в качестве элементов списка ввода? Как работает оператор ввода (что происходит при его выполнении)?
2. Запишите общий вид оператора присваивания.
3. Укажите значение величины S после выполнения следующих операторов присваивания:

s := 6   k:= -25    s := 0    s := s + k

Таблица истинности - это такая таблица, в которой показываются все выходные состояния элемента для любых комбинации входных сигналов. Иными словами, с помощью таблиц истинности можно определять истинностное значение любого высказывания для всех возможных случаев значений истинности составляющих его высказываний. Общее количество всех возможных комбинаций в таблице можно определить по формуле N=2n; где N - общее число возможных комбинаций, n - количество входных переменных. В основном таблицы истинности применяются в булевой алгебре и в цифровой электронной технике для описания работы логических схем.

Построить таблицу истинности, описывающую работу логических элементов, несложно при небольшом количестве входных переменных. Если же число переменных больше трех, то таблица получается слишком большой. Так при наличии 4 переменных, количество наборов в таблице будет равно 16, а уже при 6 переменных - 64! А еще нужно учитывать скобки, приоритет и количество операций! Очень легко совершить ошибку! И вот для того, чтобы исключить такие ошибки, на данном сайте представлена возможность построить таблицу истинности в автоматическом режисвцме. От Вас потребуется только ввести входящие данные (логическую формулу) и программа сама рассчитает и построит таблицу со всеми промежуточными вычислениями. Максимальное число входных переменных таблицы истинности составляет 6. Работа программы интуитивно понятна и не вызывает каких либо сложностей.

Кроме построения таблиц истинности, данные онлайн инструменты позволяют узнать СКНФ и СДНФ функции, найти полином Жегалкина, минимизировать булеву функцию с помощью карт Карно, построить логическую схему.

Литература:

1. Презентация «Паскаль\_Основы» 29-48, 88-96 слайды.
2. Презентация «Паскаль\_Массивы» 132-164 слайды.
3. <http://physics.herzen.spb.ru/library/03/01/pp/TPHelp/statements.htm>

# 6. Цикл с переменной. Цикл с предусловием. Цикл с постусловием. Вложенные циклы.

Литература:

1. В Blackboard: Курс «Практикум на ЭВМ» -> Раздел «Конспекты лекций» -> Методичка «Практикум на ЭВМ», стр. 13-14

<https://bb.dvfu.ru/webapps/blackboard/execute/content/file?cmd=view&mode=designer&content_id=_92799_1&course_id=_2759_1>

1. Презентация «Паскаль\_Основы» 49-87 слайды.
2. <http://learnpascal.ru/vvedenie-v-paskal/for-while-repeat.html>

# 7. Процедуры и функции. Передача параметров по ссылке и по значению.

Литература:

1. <http://mif.vspu.ru/books/pascal/procedure.html>
2. <http://learnpascal.ru/vvedenie-v-paskal/procedure-and-function-pascal.html>
3. Презентация «Паскаль\_Основы» 153-163 слайды.

# 8. Алгоритмическая сложность. Асимптотическая сложность. Понятия О-большое.

Литература:

1. <https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C>
2. <https://tproger.ru/articles/computational-complexity-explained/>
3. <https://www.intuit.ru/studies/courses/683/539/lecture/12149?page=3>
4. <https://intellect.ml/analizu-slozhnosti-algoritmov-70>

# 9. Массивы. Перебор элементов массива.

Контрольные вопросы

1. Что такое массив?
2. Какой массив называется линейным (одномерным)?
3. Как называется номер элемента одномерного массива?
4. Как описываются (объявляются) массивы?
5. Что происходит в памяти компьютера при объявлении одномерного массива?
6. Как можно заполнить одномерный массив?
7. Задан массив: А[1]=4 А[2]=-5 А[3]=0 А[4]=-1 А[5]=24 А[6]=14

а) чему равен элемент с индексом 2?

б) какой индекс у элемента 24?

в) выпишите элементы с четными индексами;

г) выпишите индексы положительных элементов.

Литература:

1. Презентация «Паскаль\_Массивы» 2-13, 22-38 слайды.
2. <http://www.pascal.helpov.net/index/one-dimensional_arrays_pascal_programming>

# 10. Кодирование символов. Символьные строки. Функции для работы со строками. Преобразование «строка-число». Операции над строковым типом данных.

Литература:

1. Презентация «Кодирование информации» 21-29 слайды.
2. Презентация «Паскаль\_Массивы» 87-113 слайды.
3. <https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%BE%D0%B2,_%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8B_%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%BE%D0%BA>
4. <http://mif.vspu.ru/books/pascal/string.html>

# 11. Двумерные массивы. Матрицы.

Контрольные вопросы

1. Как описываются двухмерные массивы?
2. Сколько индексов характеризуют конкретный элемент двухмерного массива? Что означает каждый индекс?
3. Как можно заполнить двухмерный массив?
4. Задан массив А(3,3): 23 45 -8

7 80 3

44 16 -5

Определите:

а) значение элементов: А(1,2), А(2,1), А(3,1), А(1,3);

б) индексы отрицательных элементов;

в) индексы элементов, стоящих на главной диагонали (диагональ, соединяющая левый верхний и правый нижний углы матрицы);

г) индексы элементов, стоящих на побочной диагонали (диагональ, соединяющая правый верхний и левый нижний углы матрицы); чему равна сумма индексов каждого элемента побочной диагонали?

Литература:

1. Презентация «Паскаль\_Массивы» 121-131 слайды.
2. <http://www.borlpasc.narod.ru/docym/prac/dvymerni.htm>

# 12. Перечисляемый тип данных. Интервальный тип данных.

Литература:

1. <http://gospodaretsva.com/urok-17-perechislyaemyj-i-intervalnyj-tipy-dannyx.html>

# 13. Структуры. Записи с вариантами.

Контрольные вопросы

1. Понятие записи. Структура объявления типа записи.
2. Обращение к значению поля. Составные имена.
3. Присвоение полям записи значений. Массивы записей.
4. Какие типы файлов существуют?
5. Поясните назначение файловой переменной.
6. Перечислите основные процедуры для работы с файлами данных

Литература:

1. Презентация «Динамические структуры данных» 17-32 слайды
2. <http://informatics.mccme.ru/mod/book/view.php?id=534&chapterid=259>

# 14. Понятие указателя. Работа с указателями в ЯП Паскаль: типизированные и нетипизированные указатели, разыменование, взятие адреса, арифметика указателей. Динамические массивы.

В любой вычислительной системе память относится к таким ресурсам, которых всегда не хватает. Управление памятью — одна из главных забот программиста, так как для него очень важно создавать программы, эффективно использующие память, ведь во время выполнения программы память необходима для следующих элементов программ и данных:

• сама программа пользователя;

• системные программы времени выполнения, которые осуществляют вспомогательные действия при работе программы пользователя;

• определяемые пользователем структуры данных и константы;

• точки возврата для подпрограмм;

• временная память для хранения промежуточных результатов при вычислении выражений;

• временная память при передаче параметров и т.п.

Решение проблемы экономного расходования памяти состоит в том, чтобы не резервировать заранее максимальный объем памяти для размещения данных, а, предварительно определив тип данных, создавать новый экземпляр данных всякий раз, когда в нем возникает необходимость. Такие переменные, которые создаются и уничтожаются в процессе выполнения программы, называются *динамическими.* Динамическая память широко используется для временного запоминания данных при работе с графическими и звуковыми средствами компьютера.

Указатели

Все рассмотренные ранее типы данных содержат непосредственно данные, размещенные в памяти компьютера. Для организации динамической памяти применяются особые переменные, называемые *указателями.* Назначение их состоит в том, чтобы указывать местоположение какой-то другой переменной заранее определенного типа.

Таким образом, *указатель* - это переменная, которая в качестве своего значения содержит адрес первого байта памяти, по которому записаны данные. Сам по себе указатель занимает в памяти всего *четыре байта*, а данные, на которые он указывает, могут простираться в памяти на десятки килобайт. Переменной, на которую указывает указатель, не обязательно присваивать какое-либо имя. К ней можно обращаться через имя указателя, потому она называется *ссылочной переменной.*

Типизированные указатели

Указатели, содержащие адрес, по которому записана переменная заранее определенного типа, называются *типизированными*. Для объявления типизированного указателя используется знак ^, который помещается перед соответствующим типом.

Для объявления типа указателя Р на целочисленный тип данных следует записать:

*Type Р = ^Integer;*

Тип *Integer* в данном примере является *базовым типом*. Имея в программе определение типа указателей (ссылочного типа), можно по общим правилам описать переменные этого типа. При этом ссылочные типы в описаниях переменных можно задавать как посредством идентификаторов, так и явно, например:

*Var*

*Р1, Р2 : Р; {Тип Р введен выше}*

*R : ^Byte;*

Описание типов указателей - единственное исключение из общего правила, согласно которому все идентификаторы должны быть описаны перед использованием. Однако если базовый тип является еще не объявленным идентификатором, то он должен быть объявлен в той же самой части объявления, что и тип указатель.

*Например:*

*RecPtr = ^RecordType;*

*RecordType=record*

*Name : string[20];*

*Number : integer;*

*end;*

В данном примере *RecPtr* описывается как указатель на переменную *RecordType*. Базовый тип *RecordType* описывается в той же самой последовательности определений типов, что и тип *RecPtr.*

Реально *значения* ссылочных типов *(указателей)* содержат *адреса расположения в памяти конкретных значений базового типа*. В персональном компьютере адреса задаются совокупностью двух шестнадцатиразрядных слов, которые называются сегментом и смещением. *Сегмент* - это участок памяти, имеющий длину *65536 байт (64 Кбайт)* и начинающийся с физического адреса, кратного 16 (0, 16, 32, 48 и т.д.). *Смещение* указывает, сколько байт от начала сегмента необходимо пропустить, чтобы обратиться к нужному адресу. Таким образом, по своей внутренней структуре любой указатель представляет собой *совокупность двух слов* (данных типа *Word*), трактуемых как *сегмент и смещение.*

*Абсолютный адрес* образуется следующим образом: *сегмент\* 16+смещение.*

Для того чтобы присвоить переменной ссылочного типа некоторое значение, можно воспользоваться унарной операцией *взятия указателя*, которая строится из знака этой операции-символа *@* (амперсант) и одного операнда-переменной. Например, если имеется описание переменной *I* целого типа:

*Vаr I : integer;*

то применение этой операции к переменной I: @I дает в качестве результата значение типа *указатель на целое*. Аналогичный результат получится и в результате операции *Р = ^Integer;*

*Операция взятия указателя* допустима для любых переменных, в том числе для элементов массивов, полей записи и т. д.

Например, если есть описание:

*Var А : array[1..10] of integer;*

то конструкция *@А[I]* имеет смысл *указателя на I-е целое в массиве А* и также может участвовать в присваивании: *Р := @А[I].*

Ссылочные типы можно образовывать от любых типов, поэтому допустимо определение вида *указатель на указатель*.

Среди всех возможных указателей в Турбо Паскале выделяется один специальный указатель, который *"никуда не указывает"*. Можно представить такую ситуацию: в адресном пространстве оперативной памяти компьютера выделяется один адрес, в котором заведомо не может быть размещена никакая переменная. На это место в памяти и ссылается такой нулевой (или пустой) указатель, который обозначается словом *Nil*. Указатель *Nil* считается константой, совместимой с любым ссылочным типом, поэтому его значение можно присваивать любому указателю. Обычно значение *Nil* присваивают указателю, когда его указание надо отменить или в начале инициализации программы. Это позволяет проверять значение указателя, прежде чем присваивать ему какое-либо значение.

Нетипизированный указатель *(pointer)*

В Турбо Паскале можно объявлять указатель и не связывать его при этом с каким-либо конкретным типом данных. Для этого служит стандартный тип *Pointer*. Он обозначает *нетипизированный указатель*, т. е. указатель, который не указывает ни на какой определенный тип. С помощью нетипизированных указателей удобно динамически размещать данные, структура и тип которых меняются в ходе программы.

Переменные типа *Pointer* не могут быть разименованы: указание символа ^ после такой переменной вызывает появление ошибки. Как и значение, обозначаемое словом *Nil*, значения типа *Pointer* совместимы со всеми другими типами указателей.

Над значениями ссылочных типов допускаются две операции сравнения на равенство и неравенство, например *@Х <> @Y* или *Р1 = Р2.*

*Два указателя равны* только в том случае, *если они ссылаются на один и тот же объект.*

Доступ к переменной по указателю

Для доступа к переменной имеются две возможности: первая - использовать идентификатор переменной, вторая - воспользоваться адресом этой переменной, который содержится в указателе на эту переменную.

Например, чтобы увеличить значение переменной *I,* на которую указывает указатель *Р:* по первому способу можно записать: *I:=I+2.*Для реализации второго, *косвенного доступа к переменной* по указателю, используется *разыменование указателя*. Правило разыменования таково: для того чтобы по указателю на переменную получить доступ к самой переменной, нужно после переменной-указателя поставить знак ^. Так, запись: *Writeln(Int2^)* означает напечатать значение переменной, на которую ссылается указатель *Int2*. Поэтому чтобы увеличить значение переменной I, на которую указывает указатель Р, используя косвенный доступ к переменной по указателю, можно записать: *Р ^:= P^ + 2.*

Разыменование допускается для любых ссылочных типов. В случае использования *указателя на указатель* возможно многократное разыменование. Разыменование считается некорректным, если ссылочная переменная имеет значение *Nil*, т. е. в этом случае не существует переменной, на которую ссылается указатель.

Управление динамической памятью

Вся динамическая память в Турбо Паскале рассматривается как сплошной массив байтов, который называется *кучей*. Физически куча располагается в старших адресах сразу за областью памяти, которую занимает тело программы.

Начало кучи хранится в стандартной переменной *HeapOrg*, конец - в переменной *HeapEnd*. Текущую границу незанятой динамической памяти указывает указатель *HeapPtr*.

Для управления кучей используются следующие стандартные процедуры и функции.

Процедуры динамического распределения

***Dispose***  Уничтожает динамическую переменную

***FreeMem*** Уничтожает динамическую переменную данного размера

***GetMem*** Создает новую динамическую переменную заданного размера и устанавливает переменную-указатель для нее

***Mark*** Записывает в переменной-указателе состояние кучи

***New*** Создает новую динамическую переменную и устанавливает на нее переменную-указатель

***Release*** Возвращает кучу в заданное состояние

**Функции динамического распределения**

***MaxAvail*** Возвращает размер наибольшего непрерывного свободного блока кучи, соответствующей размеру наибольшей динамической переменной, которая может быть распределена в момент вызова *MaxAvail*

***MemAvail*** Возвращает количество имеющихся в куче свободных байтов

**Функции для работы с указателями и адресами**

***Addr*** Возвращает адрес заданного объекта

***Cseg*** Возвращает текущее значение регистра *CS*

***Dseg***  Возвращает текущее значение регистра *DS*

***Ofs***  Возвращает смещение заданного объекта

***Ptr*** Преобразует базовый адрес сегмента и смещение в значение типа указатель

***Seg*** Возвращает сегмент для заданного объекта

***SPtr*** Возвращает текущее значение регистра *SP*

***Sseg*** Возвращает текущее значение регистра *SS*

Основные действия над динамическими переменными - создание и уничтожение - реализуются в Турбо Паскале стандартными процедурами *New* и *Dispose*.

Процедура *New* предназначена для создания динамических переменных определенного типа. Она отводит новую область памяти в куче для данной динамической переменной и сохраняет адрес этой области в переменной-указателя. Можно присвоить значение переменной-указателю и с помощью оператора @ или функции *Ptr*. Оператор *@* устанавливает переменную-указатель на область памяти, содержащую существующую переменную, включая и те переменные, которые имеют идентификаторы. Функция *Ptr* устанавливает переменную-указатель на определенный адрес в памяти. Например:

*New(Int1);*

*P1 := @X;*

*Ptr($40, $49);*

Для освобождения памяти в куче предназначена процедура *Dispose* с параметром указатель на динамическую переменную, причем эта переменная должна быть ранее размещена в куче посредством *New*, а тип размещенной переменной должен совпадать с базовым типом параметра процедуры *Dispose.* Например: *Dispose(Int1)* освобождает выделенный в предыдущем примере в куче фрагмент памяти так, что его можно снова использовать, если потребуется.

Если в программе не предполагается использование кучи, то при работе с программой Турбо Паскаля рекомендуется снизить потребность в памяти для создаваемой программы с помощью меню *Options/Heap*.

При освобождении динамической памяти нужно соблюдать осторожность, т.е. пользоваться либо *New/Dispose*, либо *New/Mark/Release*, либо *GetMem/FreeMem*, но ни в коем случае не путать сочетание этих процедур.

Пример программы с использованием динамической памяти

*{Программа указателя на массив, доступа к элементам массива}*

Program Point1;

Uses Crt;

Type

massiv=array[1..10] of integer; *{описание типа массива из 10 целых чисел}*

Var

i:integer;

a:massiv; *{массив из 10 целых чисел}*

PtrMas:^massiv;*{указатель на массив}*

Begin

*{ввод элементов массива}*

Writeln('Введите элементы массива : ');

for i:=1 to 10 do

begin

Writeln(i,'-й элемент массива ');

Readln(a[i]);

end;

ClrScr;

Writeln('Введенный массив: ');

*{вывод элементов введенного массива }*

for i:=1 to 10 do

Write(' ',a[i]);

Writeln;

*{Присвоить указателю PtrMas адрес начала размещения массива в памяти}*

PtrMas:=@a;

Writeln('Значение первого элемента массива, на который указывает PtrMas: ');

*{Напечатать значение первого элемента массива, на который указывает указатель PtrMas}*

Writeln(PtrMas^[1]);

*{Присвоить указателю PtrMas  адрес начала размещения второго элемента массива}*

PtrMas:=@a[2];

Writeln(' Значение второго элемента массива, на который указывает указатель PtrMas: ');

*{Напечатать значение первого элемента массива, на который указывает указатель PtrMas}*

Writeln(PtrMas^[1]);

Writeln(' Значение 4-того элемента массива, на который указывает указатель PtrMas: ');

*{Присвоить указателю PtrMas адрес 4-того элемента массива}*

PtrMas:=Ptr(Seg(a[3]),Ofs(a[3])+SizeOf(integer));

*{Ptr- преобразует базовый адрес сегмента и смещения в значение типа указатель}*

*{Напечатать значение 1 элемента массива, на который указывает указатель PtrMas}*

Writeln(PtrMas^[1]);

Readkey;

end.

Контрольные вопросы

1. Статические и динамические переменные.
2. Указатели: типизированные, нетипизированные, Nil. Доступ к переменной по указателю.
3. Процедуры и функции управления динамической памятью

Литературы:

1. Презентация «Паскаль динамические структуры данных» 2-16 слайды

# 15. Динамические структуры данных. Линейный список. Очередь. Стек. Дек.

* 1. Для Стека уметь реализовывать методы:
     + push (var head:pNode, x:type)
     + pop (var head:pNode):type
     + top (head:pNode):type
     + clearAll (var head: pNode)
     + size (head:pNode):integer
     + isEmpty (head:pNode): boolean
  2. Для Очереди:
     + push (var head, tail:pNode, x:type)
     + pop (var head, tail:pNode):type
     + front (head, tail:pNode):type
     + back (head, tail:pNode):type
     + clearAll (var head, tail: pNode)
     + size (head, tail:pNode):integer
     + isEmpty (head, tail:pNode): boolean
  3. Для Дэка предусмотреть 2 реализации: обычную и кольцевую:
     + push\_head, push\_tail (var head, tail:pNode, x:type)
     + pop\_head, pop\_tail (var head, tail:pNode):type
     + front (head, tail:pNode):type
     + back (head, tail:pNode):type
     + clearAll (var head, tail: pNode)
     + size (head, tail:pNode):integer
     + isEmpty (head, tail:pNode): boolean

Литература:

1. Презентация «Паскаль динамические структуры данных» 33-80 слайды
2. <http://labs.org.ru/pascal-15_2/>

# 16. Рекурсия.

**Рекурсией** называется ситуация, когда процедура или функция вызывает сама себя. Рекурсивный алгоритм обращается к самому себе, пока не выполнится определенное условие, поэтому в любой рекурсивной подпрограмме должна быть нерекурсивная ветвь (в примере 1 это: if N=1 then Fact:=1).

**Пример 1.**

Вычисление факториала натурального числа **N**.

N!=1\*2\*3\*…\*N

Вычисление факториала можно определить следующим образом:

N! =

Program Pr1;

Var

N: integer;

Function Fact(N:integer):real;

begin

if N=1 then Fact:=1 else Fact:=N\*Fact(N-1);

end;

BEGIN *{основная программа}*

Write('N='); Readln(N);

Writeln(N, '!=', Fact(N):4:0);

Readln

END.

**Пример 2.**

Расчет суммы первых N натуральных чисел

Вычисление суммы 1+2+3+…+N можно определить следующим образом:

Sum(N) =

Program Pr2;

Var

N: integer;

Function Sum(N: integer): integer;

begin

if N=0 then Sum:=0 else Sum:= N+Sum(N-1);

end;

BEGIN *{основная программа}*

Write('N='); Readln(N);

Writeln('S=', Sum(N));

Readln

END.

**Пример 3.**

Вычисление **n**-го члена последовательности Фибоначчи.

Последовательность Фибоначчи определяется следующим образом: первые два числа равны 1, а каждое следующее равно сумме двух предыдущих (1, 1, 2, 3, 5,…).

Вычисление **n**-го члена последовательности Фибоначчи можно определить следующим образом:

Fib(n)= 

Program Pr3;

Var

n:integer;

Function Fib(n:integer):integer;

begin

if n<=2 then Fib:=1 else Fib:=Fib(n-1)+Fib(n-2);евк

end;

BEGIN *{основная программа}*

Write('n='); Readln(n);

Write(Fib(n):5);

Readln

END.

**Пример 4.**

Расчет n-члена арифметической прогрессии, заданной значением первого члена a1 и разностью d.

Вычисление n-члена арифметической прогрессии можно определить следующим образом:

an =

Program Pr4;

Var

n:integer; d,a1:real;

Function Arifm(a1,d:real; n:integer):real;

begin

if n=1 then Arifm:=a1 else Arifm:=Arifm(a1,d,n-1) + d;

end;

BEGIN *{основная программа}*

Write('a1='); Readln(a1);

Write('d='); Readln(d);

Write('n='); Readln(n);

Write(Arifm(a1,d, n):5:2);

Readln

END.

**Пример 5.**

Нахождения суммы **n** членов арифметической прогрессии, заданной значениями первого члена прогрессии **а** и разности **d.**

Program Pr5;

Var

s,a,n,d: integer;

Function sa(n,a,d:integer):integer;

begin

if n>0 then sa:=a+ sa(n-1,a+d,d) else sa:=0;

end;

BEGIN *{основная программа}*

Write('n='); Readln(n);

Write('a='); Readln(a);

Write('d='); Readln(d);

Write(sa(n,a,d));

Readln

END.

**Пример 6.**

Возведение числа **a** в целую степень **n**.

Program Pr6;

Var a:real;

n:integer;

Function Stepen(a: real;n:integer):real;

begin

if n=0 then Stepen:=1

else if n>0 then Stepen:=a\*Stepen(a,n-1)

else Stepen:=(1/a)\*Stepen(a,n+1);

end;

BEGIN *{основная программа}*

Write('a='); Readln(a);

Write('n='); Readln(n);

Write(Stepen(a,n):6:4);

Readln

END.

**Пример 7.**

Представление натурального числа в двоичной системе счисления.

Program Pr7;

Var

n:integer;

Procedure Bin\_n(n:integer);

begin

if n>1 then Bin\_n(n div 2);

Write( n mod 2)

end;

BEGIN *{основная программа}*

Write('n='); Readln(n);

Bin\_n(n);

Readln

END.

**Пример 8.**

Нахождения наибольшего общего делителя двух натуральных чисел

Program Pr8;

Var

a,b:integer;

Function Nod(a,b:integer):integer;

begin

if a=b then Nod:=a

else if a>b then Nod:=Nod(a-b, b)

else Nod:=Nod(a, b-a) ;

end;

BEGIN *{основная программа}*

Write('a='); Readln(a);

Write('b='); Readln(b);

Writeln('NOD(', a, ',', b, ')=', Nod(a,b));

Readln

END.

Привести примеры рекурсивных алгоритмов. Привести алгоритмы с заменой рекурсии на цикл. привести алгоритмы с неоптимальной заменой.

Литература

1. http://kvodo.ru/recursion-in-pascal.html

# 17. Модули. Раздельная компиляция.

Богатство алгоритмических возможностей Паскаля в значительной степени достигается благодаря использованию модулей.

*Модуль* представляет собой набор констант, типов данных, переменных, процедур и функций. Каждый модуль по своей структуре аналогичен *отдельной программе*.

*Модуль - программная единица, текст которой компилируется независимо (автономно).*

Вместе с тем, структура модуля позволяет использовать его как своеобразную *библиотеку описаний*. Модули являются достаточно гибким и удобным инструментальным средством при разработке больших программах комплексов рамках совместной технологии разработки программного обеспечения (структурное программирование и др.).

Кроме того, использование модулей позволяет практически обойти известное для 16-разрядной ПЭВМ *ограничение на размер кодового сегмента* (как известно, размер *кодового сегмента* отдельной программы не должен превышать *64 Кбайт*). Это достигается благодаря тому, что каждому модулю при выполнении при выполнении программы отводится свой отдельный сегмент оперативной памяти.

Паскаль располагает 8-мью стандартными (встроенными) модулями. Это *System, Dos, Overlay, Graph, CRT, Printer, Turbo3, Graph3*. Два последних модуля предназначены для поддержки совместимости программ, написанных на Турбо-Паскале версии 3.0.

Все перечисленные стандартные модули (кроме Graph, Graph3, Turbo3) объединены и сохранены в файле *TURBO.TPL.*

Модуль *SYSTEM* поддерживает все стандартные процедуры и функции, обеспечивает ввод-вывод данных, обработку строк, динамическое распределение оперативной памяти и ряд других возможностей Турбо-Паскаля. Модуль *SYSTEM* подключается к любой программе автоматически.

Модель *DOS* содержит многочисленные процедуры и функции, многие из которых по своему действию эквивалентны командам MS-DOS (*GetTime*, *DiskSize* и др.).

Модуль *OVERLAY* обеспечивает поддержку систем оверлеев.

Модуль *CRT* поддерживает ряд стандартных процедур и функций, которые обеспечивают работу с экраном дисплея в текстовом режиме, управление звуком и работу с клавиатурой.

Модуль *PRINTER* определяет драйвер печатающего устройства и позволяет организовывать вывод информации на принтер.

Модуль *GRAPH* обеспечивает работу с экраном дисплея в графическом режиме.

Наряду с использованием стандартных модулей каждый программист имеет возможность организации собственных модулей.

*Структура любого следующего модуля имеет вид:*

*ЗАГОЛОВОК МОДУЛЯ*

**Unit** < имя модуля >;

*ИНТЕРФЕЙСНАЯ ЧАСТЬ*

**Interfaсe**

**Uses** < список используемых модулей >

{ открытые объявления}

**Type**

**Var**

**Procedure**

**Function**

*РЕАЛИЗАЦИОННАЯ ЧАСТЬ*

**Implementation**

**Uses** < список используемых модулей >

{ собственные объявления }

**Type**

**Var**

{ процедуры и функции }

**Procedure**

**Function**

*ИНИЦИАЛИЗАЦИОННАЯ ЧАСТЬ*

**Begin**

**…** {Основной блок модуля}

**End**.

Имя модуля записывается за ключевым словом *UNIT*. При выборе имени модуля необходимо учитывать одну особенность: *имя модуля должно совпадать с именем файла, в котором он хранится.*

Далее записывается раздел интерфейсной части (за ключевым словом *Interfaсe*). Эта часть модуля является доступной (“видимой”) для любой программы, использующей этот модуль. То есть объявленные в этом разделе константы, типы данных, переменные, процедуры и функции, могут использоваться в любой другой программе. В свою очередь, в разделе интерфейса могут указываться другие используемые модули (их список следует за ключевым словом *Uses*). При этом все объекты, объявленные в интерфейсах этих модулей могут быть использованы в любом объявлении в интерфейсе данного модуля.

*Примечание.* Для “видимых” процедур и функций в интерфейсном разделе приводится только их заголовки. Полностью эти процедуры и функции записываются в разделе реализации.

Следующим структуре модуля описывается раздел реализации (за ключевым словом *Implementation*). В этом разделе могут использоваться все объекты, описанные в разделе интерфейса. Вместе с тем, здесь могут объявляться свои константы, типы данных, переменные процедуры и функции. Они могут быть использованы *только в данном разделе реализации* и в этом смысле являются “не видимыми”. Это же ограничение относится и к интерфейсам других модулей, список которых следует за ключевым словом *Uses* (в отличие от аналогичного списка в разделе интерфейса данного модуля). Таким образом, различие всех описаний содержащихся в разделе интерфейса и реализации заключается в сфере их использования (первые -доступны извне, вторые -только внутренние).

Раздел реализации модуля начинается ключевым словом *Implementation* и заканчивается *end*. Но если между ними появляется ключевое слово *begin*, то получившийся составной оператор *begin…end* становится разделом инициализации модуля. Раздел инициализации обычно используется для открытия файлов и для формирования значений структур данных и переменных.

При выполнении программы, использующей модули, их разделы инициализации вызываются раньше загрузки самой программы. Для того чтобы использовать модули в программах их имена следует указать в разделе подключения модулей *Uses*.

Например, вы разработали программу, которая наряду со стандартным модулей *Crt* использует ваш разработанный собственный модуль с именем *Modul.* Тогда, в программе следует указать список используемых модулей в следующем виде:

Program Pr;

Uses Crt, Modul;

. . .

Модули транслируются отдельно. В отличие от основных программ результатом трансляции которых будут файлы с расширением *EXE* модули получают расширение *TPU*. Полученные в результате трансляции *TPU - файлы* можно подсоединить к стандартному файлу *TURBO.TPL* с помощью утилиты *TPUMOVER.EXE*. Если этого не делать, то при трансляции самой программы все используемые модули (*TPU - файлы*) присоединяться к ней автоматически. Если в каком-либо из используемых модулей были внесены изменения, то при трансляции программы, все модифицируемые модули также будут предварительно перетранслированы (эту функцию реализует интегрированная среда).

Пример разработки собственного модуля

Разработку собственного модуля рассмотрим на следующем примере:

Пусть дано задание: разработать личную библиотеку, включив в нее процедуры:

* ввода элементов числовой матрицы размером N\*N;
* транспонирования матрицы;
* вывода результирующей матрицы.

В основной программе ввести размер матрицы N.

Начнем разработку модуля, который будет носить название Matrix. Программно это будет выглядеть так:

**Unit** Matrix;

{Зарезервированное слово *Unit* служит для указания имени библиотеки. Это имя

должно совпадать с именем PAS-файла библиотеки (т.е библиотека Matrix должна

находиться с файле *Matrix.Pas*), а иначе компилятор даст ошибку при попытке

использования такой библиотеки}

**Interface**

{Секция *Interface* содержит описания общедоступных типов данных, констант,

процедур и функций. Т.е. все, что будет здесь находиться можно будет

использовать при подключении данной библиотеки.}

**Type**

TMatrix = array [1..10,1..10] of Integer; { Квадратная матрица }

procedure MatrInput (Var m : TMatrix; n : Integer); { ввод матрицы }

procedure MatrOutput (Var m : TMatrix; n : Integer); { вывод матрицы }

procedure MatrTransp (Var m : TMatrix; n : Integer); { транспонирование }

**Implementation**

{Секция *Implementation* содержит реализацию тел процедур и функций, описанных

в *Interface*. Также здесь могут содержаться типы данных, константы, процедуры

и функции, необходимые для работы, но которые не будут видны программе при

подключении библиотеки.}

{Процедура обмена местами двух элементов матрицы (x1,y1) и (x2,y2).

Эта процедура используется при транспонировании матрицы, но ее

нельзя вызвать при подключении библиотеки, т.к. она не объявлена

в секции *Interface*.}

procedure Swap (Var m : TMatrix; x1,y1,x2,y2 : Integer);

var

temp : Integer;

begin

temp := m[x1,y1];

m[x1,y1] := m[x2,y2];

m[x2,y2] := temp;

end;

{Ввод матрицы с клавиатуры. Параметры процедуры здесь не указаны, т.к. они есть в секции Interface }

procedure MatrInput;

var

i,j : Integer;

begin

for i:=1 to n do

begin

Write(i:3,'-я строка : ');

for j:=1 to n do Read(m[i,j]);

ReadLn;

end;

end;

{Транспонирование матрицы.}

procedure MatrTransp;

var

i,j : Integer;

begin

for i:=1 to n-1 do

for j:=i+1 to n do

Swap (m,i,j,j,i);

end;

{Вывод матрицы на экран.}

procedure MatrOutput;

var

i,j : Integer;

begin

for i:=1 to n do

begin

Write(i:3,'-я строка : ');

for j:=1 to n do Write (m[i,j]:4);

WriteLn;

end;

end;

{Эта секция может использоваться для инициализации работы библиотеки.}

Begin

End.

Создание модуля закончено. Теперь необходимо создать файл, который будет содержать текст основной программы, в которой будет подключаться разработанный выше модуль.

{Это отдельный файл, содержащий основную программу}

Uses

Crt, { Библиотека стандартных процедур управления экраном и клавиатурой }

Matrix; {Наш разработанный модуль-библиотека работы с квадратными матрицами (личная)}

Var

m : TMatrix; { Объявляем матрицу - максимальный размер 10\*10 }

n : Integer; { Размер матрицы }

Begin

{ Повторяем ввод размера, пока не будет введено корректное значение}

repeat

ClrScr;

Write('Введите размер матрицы (1..10) : ');

ReadLn(n);

until (n >= 1) and (n <= 10);

WriteLn;

WriteLn('Введите матрицу размера',n,'\*',n,'по строкам:');

MatrInput (m,n); {вызов процедуры ввода матрицы, определенной в модуле Matrix}

{Транспонируем ее }

MatrTransp (m,n); {вызов процедуры транспонирования матрицы, определенной в модуле Matrix}

{ Выведем результат на экран }

WriteLn;

WriteLn('Транспонированная матрица :');

MatrOutput (m,n); {вызов процедуры вывода матрицы, определенной в модуле Matrix}

End.

Контрольные вопросы

1. Стандартные модули в Паскале.
2. Структура модуля.
3. Ключевые слова Unit, Interface, Implementation. Описание каждого раздела.
4. Концепция разработки собственного модуля. Пример программы.
5. Раздельная компиляция

Литература:

1. <http://www.pascal.helpov.net/index/pascal_modules_programming>

# 18. Уточнение понятия алгоритм. Машина Тьюринга. Машина Поста.

В истории математики накопилось много случаев длительных и часто безрезультатных поисков тех или иных алгоритмов. При этом естественно возникало сомнение в существовании алгоритма.

Одним из ярких примеров такого случая является история решения десятой проблемы Д.Гильберта.

В 1900 году на втором международном математическом конгрессе в Париже немецкий математик Давид Гильберт огласил список 23 трудных проблем, на важность решения которых он обращал внимание математической общественности. Среди них была и следующая 10-я проблема Гильберта: требуется выработать алгоритм, позволяющий для любого диофантова уравнения выяснить, имеет ли оно целочисленное решение.

Рассмотрим всевозможные диофантовы уравнения, т.е. уравнения вида *P* = 0, где *P* является многочленом с целочисленными коэффициентами. Такими будут, например, уравнения

*x*2 + *y*2 – 2*xz* = 0,

10*x*5 + 7*x*2 + 5 = 0,

из которых первое с тремя неизвестными, а второе с одним неизвестным. В общем случае рассматривают уравнения с любым числом неизвестных. Такие уравнения могут иметь целочисленные решения, а могут и не иметь.

Так, уравнения *x*2 + *y*2 – 2*xz* = 0 имеет бесконечное множество целочисленных решений, а уравнение 10*x*5 + 7*x*2 + 5 = 0 таких решений не имеет.

Для частного случая диофантова уравнения с одним неизвестным давно известен алгоритм, позволяющий найти все целочисленные решения. Установлено, что если уравнение

*Рn*(*x*) = *a0xn* + *a1xn-1* +…+ *an-1x* + *an* = 0

с целочисленными коэффициентами имеет целый корень, то он обязательно является делителем *а*n. В связи с этим можно предложить такой алгоритм:

1. Найти все делители числа *аn*: *d1*,*d2*, …, *dn*.
2. Вычислить *Pn*(*x*) для каждого из делителей числа *аn*.
3. Если при некотором *i* из совокупности 1, 2, …, *k* *Рn*(*di*) = 0, то *di*– корень уравнения. Если при всех *i* = 1, 2, …, *k* *Рn*(*di*) ≠ 0, то уравнение не имеет целочисленных решений.

Поиски решения десятой проблемы Гильберта привлекли внимание многих математиков и длились около 70 лет. И только в 1968 году молодым математиком Ю.Матиясевичем было доказано, что нет алгоритма, дающего решение поставленной задачи.

В подходах к определению понятия алгоритма можно выделить *три основных направления*.

*Первое направление* связано с уточнением понятия *эффективно вычислимой функции*. Этим занимались А.Черч, К.Гедель, С.Клини, определившие алгоритм как последовательность построения сложных функций из более простых. В результате был выделен класс так называемых *частично-рекурсивных функций*, имеющих строгое математическое определение. Анализ идей, приведших к этому классу функций, дал им возможность высказать гипотезу о том, что класс эффективно вычислимых функций совпадает с классом частично рекурсивных функций.

*Второе направление* связано с *машинной математикой*. Здесь сущность понятия алгоритма раскрывается путём рассмотрения процессов, осуществляемых в машине. Впервые это было сделано Тьюрингом, который предложил самую общую и вместе с тем самую простую *концепцию вычислительной машины*. Её описание было дано Тьюрингом в 1937 году. При этом Тьюринг исходил лишь из общей идеи работы машины как работы вычислителя, оперирующего в соответствии с некоторым строгим предписанием.

*Третье направление* связано с понятием нормальных алгоритмов, введённым и разработанным российским математиком А.А.Марковым. В рамках этого направления алгоритм рассматривается как конечный набор правил подстановок цепочек символов.

Удивительным научным фактом является доказательство эквивалентности приведенных выше определений алгоритма. Эквивалентность двух абстрактных моделей алгоритма состоит в том, что любой класс проблем, которые можно решить с помощью моделей одного типа, можно решить и на моделях другого типа. Оказалось, что все известные алгоритмы могут быть представлены аналогичными алгоритмами в рамках предложенных направлений (различных определений алгоритма).

*Машины Тьюринга.*

При выполнении алгоритма в интуитивном смысле мы можем пользоваться потенциально неограниченной памятью, запоминая в процессе выполнения алгоритма по мере необходимости нужную информацию, например, на листочке бумаги. И если для решения проблемы известен алгоритм, то для его реализации необходимо лишь четкое выполнение шагов алгоритма. Таким образом, по своей сути, алгоритм есть механический процесс обработки информации.

Впервые английский математик Алан Тьюринг определил понятие алгоритма исходя из понятия автоматически работающей машины; более того, он предложил формальную модель такого устройства, которое интуитивно моделирует действия человека, решающего задачу, руководствуясь некоторым алгоритмом. Это устройство было названо *машиной Тью*ринга.

Рассмотрим один из вариантов указанной машины.

Устройство машины Тьюринга включает в себя:

1. *Внешний алфавит*, т.е. конечное множество символов *А* = {*а*0, *а*1, *а*2,…, *а*n}. В этом алфавите в виде слова кодируется та информация, которая подаётся в машину. Машина перерабатывает информацию, поданную в виде слова, в новое слово.
2. *Внутренний алфавит* машины, состоящий из символов *q*0, *q*1, *q*2, …, *q*m, *п*, *л*, *н*. Символы *q*0, *q*1, *q*2, …, *q*m выражают конечное число состояний машины. Для любой машины число состояний фиксировано. Два состояния имеют особое назначение: *q*1 – начальное состояние машины, *q*0 – заключительное состояние (стоп-состояние). Символы *п*, *л*, *н* – это символы сдвига (вправо, влево, на месте).
3. *Бесконечная в обе стороны лента* (внешняя память машины). Она разбита на клетки. В каждую клетку может быть записана только одна буква. Пустую клетку будем обозначать символом *а*0.
4. *Управляющая головка*. Она передвигается вдоль ленты и может останавливаться напротив какой-либо клетки, т.е. воспринимать символ или печатать символ. В одном такте работы машины управляющая головка может сдвигаться только на одну клетку (вправо, влево) или оставаться на месте.

Каждое сведение, хранящееся на ленте, изображается конечным набором символов (букв) внешнего алфавита, отличного от *а*0. К началу работы машины на ленту подаётся начальное сведение (начальная информация). В этом случае управляющая головка, как правило, находится у крайнего левого знака с указанием начального состояния *q*1 (начальная конфигурация).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *А*0 | *а*1 | *а*2 | *а*3 | … | … | … | *а*n | *а*0 | *А*0 |  |  |  |  |  |  |  |  |

*q*0

Работа машины складывается из тактов, по ходу которых происходит преобразование начальной информации в промежуточные информации.

В качестве начальной информации на ленту можно подать любую конечную систему знаков внешнего алфавита (любое слово в этом алфавите), расставленную произвольным образом по ячейкам. Но в зависимости от того, какая была подана начальная информация, возможны два случая:

1. После конечного числа тактов машина останавливается (переходит в стоп-состояние *q*0), и при этом на ленте оказывается изображённой информация *В*. В таком случае говорят, что машина применима к начальной информации *А* и перерабатывает её в результирующую информацию *В*.
2. Машина никогда не останавливается (не переходит в стоп-состояние). В таком случае говорят, что машина не применима к начальной информации *А*.

В каждом такте работы машины она действует по функциональной схеме, которая имеет вид:

*a*i*q*j ⇒ *a*v{п, л, н}*q*s

Здесь *a*i,*a*v – буквы внешнего алфавита; *q*j,*q*s – состояния машины; п, л, н – символы сдвига.

В зависимости от того, какая буква на ленте обозревается управляющей головкой (в нашей записи *a*i) и в каком состоянии (в нашей записи *q*j) находится машина, в данном такте вырабатывается команда, состоящая из трёх элементов:

1. Буква внешнего алфавита, на которую заменятся обозреваемая буква (*a*v).
2. Адрес внешней памяти для следующего такта {п, л, н}.
3. Следующее состояние машины (*q*s).

Совокупность всех команд образует программу машины Тьюринга. Программа представляется в виде двумерной таблицы и называется Тьюринговой функциональной схемой.

|  |  |  |  |
| --- | --- | --- | --- |
|  | *a*0 | *a*1 | *a*2 |
| *q*1 | *a*2 л *q*3 | *a*1 п *q*2 | *a*2 л *q*1 |
| *q*2 | *a*0 н *q*2 | *a*2 н *q*1 | *a*1 н *q*2 |
| *q*3 | *a*0 п *q*1 | *a*1 п *q*4 | *a*2 н *q*1 |
| *q*4 | *a*1 н *q*3 | *A*0 п *q*4 | *a*2 п *q*4 |

Ясно, что работа машины Тьюринга полностью определяется её программой. Иными словами, две машины Тьюринга с общей функциональной схемой неразличимы, и различные машины Тьюринга имеют различные программы.

1. <http://inf.1september.ru/articlef.php?ID=200700104>
2. <https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0>

# 19. Алгоритмически неразрешимые задачи.

Литература:

1. <http://edulib.pgta.ru/els/_/disk/09.03.03%20-%20%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B0%D1%8F%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F%20%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%20%D0%BD%D0%B5%D1%80%D0%B0%D0%B7%D1%80%D0%B5%D1%88%D0%B8%D0%BC%D1%8B%D0%B5%20%D0%BF%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D1%8B.pdf>
2. <https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8_%D0%BD%D0%B5%D1%80%D0%B0%D0%B7%D1%80%D0%B5%D1%88%D0%B8%D0%BC%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0>