

Министерство образования и науки Украины
Донбасская государственная машиностроительная академия

ПРОГРАММИРОВАНИЕ И АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ

КОНСПЕКТ ЛЕКЦИЙ

(для студентов направления «Системный анализ»)

Утверждено
на заседании кафедры ИСПР
Протокол № 2 от 9 сентября 2014г.

Краматорск 2014

УДК 681.3

Программирование и алгоритмические языки : конспект лекций (для студентов направления «Системный анализ» всех форм обучения). Часть 1 / Сост. А.Ю. Мельников – Краматорск : ДГМА, 2014. – 36 с.

Приведены лекционные материалы к курсу.

Составитель	Мельников А.Ю., канд. техн. наук, доцент
-------------	--

Отв. за выпуск	Мельников А.Ю., канд. техн. наук, доцент
----------------	--

СОДЕРЖАНИЕ

Модуль 1. Основы программирования на языке Турбо-Паскаль.....	4
Лекция №1. Основные данные о языке программирования Паскаль....	4
Лекция №2. Разновидности вычислительных процессов	8
Лекция №3. Нестандартные и ограниченные типы данных	12
Лекция №4. Массивы и подпрограммы.....	13
Лекция №5. Обработка символьных данных	20
Лекция №6. Тип записи, файлы и файловые типы данных	21
Лекция №7. Модули в Турбо-Паскале	27
Лекция №8. Использование стандартных модулей в Турбо-Паскале...	28
Лекция №9. Базовые положения ООП и их использование в Турбо-Паскале.....	29
Модуль 2. Основы программирования на языке Турбо-Си.....	31
Лекция №10. Основные данные о языке программирования Си.....	31
Лекция №11. Разновидности вычислительных процессов	32
Лекция №12. Специфика языка программирования Си.....	33
Лекция №13. Массивы и обработка символьных данных.....	34
Лекция №14. Сложные типы данных.....	34
Лекция №15. Работа препроцессора и исключительные ситуации.....	35
Список рекомендованной литературы.....	36

Модуль 1

Основы программирования на языке Турбо-Паскаль

Лекция 1. Основные данные о языке программирования Паскаль

Алгоритм - конечная последовательность предписаний, однозначно определяющая процесс преобразования исходных данных в результат решения задачи. В процессе разработки алгоритма могут использоваться различные способы его описания. Наиболее распространенные:

- словесная запись;
- графические схемы алгоритмов (блок-схемы);
- псевдокод (формальные алгоритмические языки).

В программе на языке Паскаль можно использовать только символы, входящие в алфавит языка:

- а) 26 латинских букв (прописные и строчные не отличаются!);
- б) цифры от 0 до 9;
- в) знаки препинания: точка (.), точка с запятой (;), двоеточие (:), запятая (,), апостроф (');
- г) знаки арифметических операций: плюс (+), минус (–), умножение (*), деление (/);
- д) знаки операций отношения: больше (>), меньше (<), равно (=);
- е) круглые скобки: (), фигурные скобки { }, квадратные скобки [];
- ж) пробел.

В тексте, заключенном в апострофы, можно использовать любые символы, имеющиеся на клавиатуре компьютера.

Комментарий – это взятая в фигурные скобки любая последовательность символов, не содержащая закрывающей фигурной скобки. Вместо фигурных скобок можно использовать комбинацию (* ... *)

Данные бывают двух видов – константы и переменные.

Константы – это такие данные, значения которых, будучи определенными перед выполнением программы, не изменяются при ее выполнении.

Переменные – это такие данные, значения которых могут изменяться в процессе выполнения программы.

Данные (константы и переменные) относятся к какому-либо типу. Под типом данного понимается множество его допустимых значений. Тип данного определяет также множество допустимых над ним действий.

В языке Pascal существует несколько групп типов, основное деление: простые и структурированные типы данных. Также можно выделить стандартные (предопределенные) типы и типы, определяемые пользователем. Последние должны быть определены либо в разделе описания типов, либо в разделе описания переменных.

Из группы простых типов данных рассмотрим шесть основных:

Таблица 1

Тип	Имя типа	Содержание
Целый	INTEGER	Целые числа в интервале –32768 до 32767
Действительный (вещественный)	REAL	Вещественные числа в интервале от 10^{-38} до 10^{38}
Логический	BOOLEAN	Логические данные. Могут принимать значения TRUE (истина) или FALSE (ложь)
Символьный (литерный)	CHAR	Данные символьного типа. Могут принимать значения одной литеры из набора символов в ЭВМ.
Перечисляемый		Набор значений, которые может принимать данное значение
Диапазон		Подмножество упорядоченных значений, определяемое минимальным и максимальными значениями

Идентификатор – это имя любого объекта программы: переменной, константы, типа, метки и т.д. Идентификатор может включать буквы, цифры и символ подчеркивания (пробелы в идентификаторе недопустимы). Начинаться идентификатор должен с буквы (буквы только латинские). Прописные и строчные буквы в идентификаторе не различаются: NAME и name будут идентичны. Длина идентификатора может быть любой, но существенными являются только первые 63 символа.

Тип **константы** простого типа однозначно определяется ее значением и явно не описывается.

Описание констант имеет вид:

CONST < Имя константы > = Значение;

Имя константы содержит максимум 40 символов и должно начинаться с буквы (используются только латинские буквы).

Пример: **CONST a=1.92; c='S'; k=7;**

Числа с дробной частью записываются в двух формах: основной и полулогарифмической (с порядком). Вместо запятой ставится десятичная точка. Вместо основания степени 10 ставится буква E, что позволяет четко отделить мантиссу от показателя степени и записать все символы числа в одной строке. Незначащий нуль перед десятичной точкой может быть отброшен. В записи числа, имеющего только целую часть, десятичная точка не используется. Знак «+» может быть опущен

По умолчанию определены специальные константы:

Pi=3.14159265... , MAXINT - наибольшее целое число, равное 32767.

Раздел описания **переменных** начинается с ключевого слова **VAR**, после которого перечисляются переменные, массивы и их типы. Общий вид:

VAR < Имя переменной >, ..., < имя переменной > : < тип переменной >;

Имя переменной описывается аналогично именам констант.

Например:

VAR

k: integer;

x, y: real;

ar: array [1..10] of integer;

В процессе решения задач часто возникает необходимость в вычислении элементарных функций. Для обращения к функции необходимо в выражении записать идентификатор функции и в круглых скобках – аргумент. Аргументами функции могут быть константы, переменные, функции или выражения. Для тригонометрических функций угол следует задавать в радианах. Различают стандартные арифметические функции, применяемые в Pascal (табл. 2), и стандартные функции преобразования (табл. 3).

Таблица 2

Функция	Математическая запись	Запись в Pascal
Синус	$\sin x$	SIN (X)
Косинус	$\cos x$	COS (X)
Арктангенс	$\operatorname{Arctg} x$	ARCTAN (X)
Абсолютное значение	$ x $	ABS (X)
Корень квадратный	\sqrt{x}	SQRT (X)
Вычисление экспоненты	e^x	EXP (X)
Натуральный логарифм	$\ln x$	LN (X)
Возведение в квадрат	x^2	SQR (X)
Присвоение знака	–	SGN (X)
Вычисление числа	π	PI

Таблица 3

TRUNC (X)	Вычисляет целую часть аргумента X
ROUND (X)	Определяет округленное значение X
ORD (X)	Определяет порядковый номер аргумента X в упорядоченном множестве значений, определяемом типом X
CHR (X)	Определяет символ, порядковый номер которого равен аргументу X
SUCC (X)	Выдает значение, следующее за аргументом X в списке значений, определяемом для типа X
PRED (X)	Выдает значение, предшествующее аргументу X в списке значений, определяемом для типа X

Для возведения переменной x в некоторую степень a используют известное равенство $X^a = e^{a \ln(x)}$. Тогда на языке Pascal оно выглядит: EXP (A * LN (X)).

Различают выражения арифметические, строковые и выражения типа отношения.

Арифметические выражения определяют последовательность вычисления значения. Выражения могут включать в себя константы, переменные, стандартные функции, которые разделяются скобками и знаками операций.

Знаки арифметических операций: «+» (сложение), «-» (вычитание), «*» (умножение), «/» (деление), DIV (деление на целое), MOD (определение остатка от деления).

Действия выполняются слева направо с соблюдением следующего старшинства:

- а) выражения в скобках;
- б) мультипликативные операции (* , / , DIV , MOD);
- в) аддитивные операции (+ , -).

Тип результата выражения зависит от типов операндов, участвующих в операции. Тип результата + , - , * является INTEGER , если оба операнда имеют тип INTEGER, и REAL – в противном случае.

Результатом операции / всегда является тип REAL, а результат операций MOD. DIV всегда имеет тип INTEGER, так как аргументы могут быть только типа INTEGER.

Выражения типа отношений служат для установления отношений между двумя числовыми или строковыми значениями. Допустимы 6 операций: «=» (равно), «< >» (не равно), «<» (меньше), «>» (больше), «<=» (меньше или равно), «>=» (больше или равно).

Возможно выполнение логических операций: NOT – отрицание, AND – умножение (логическое и), OR – сложение (логическое или).

Пример: A = B, (A > 0) AND (B > 0)

Структура программы

Программа на языке Pascal состоит из заголовка, раздела описаний, раздела операторов и заканчивается точкой. Раздел операторов включает в себя последовательность исполняемых операторов, разделённых точкой с запятой (;) и ограниченных операторными скобками – служебными словами BEGIN END.

```

PROGRAM < имя >;
LABEL
    < метка >, ..., < метка >;
CONST < имя константы > = < константа >;
    < имя константы > =- < константа >;
TYPE < имя типа > = < тип >;
    < имя типа > = < тип >;
VAR
    < имя переменной >, ..., < имя переменной >: < тип >;

```

```

    < имя переменной >, ..., < имя переменной >: < тип >
PROCEDURE < заголовок процедуры >;
    < блок >;
FUNCTION < заголовок функции >;
    < блок >;
BEGIN
    < оператор >;
    < оператор >
    ...
END.

```

Метка – целое число без знака, содержащее не более четырех цифр или обычный идентификатор. Метка от оператора отделяется двоеточием. Метка используется для пометки оператора, на который осуществляется переход. Все используемые в программе метки должны быть определены в разделе описания меток.

Обязательным является только раздел операторов. Все остальные элементы программы могут отсутствовать.

Заголовок программы носит чисто декоративный характер и игнорируется компилятором.

В Pascal порядок размещения разделов описаний произвольный, единственное правило, которое необходимо выдержать: можно использовать лишь те идентификаторы, которые перед этим были определены.

Лекция 2. Разновидности вычислительных процессов

Операторы – это описание каких-либо действий над переменными и константами. Тело программы можно рассматривать как последовательность операторов. Операторы отделяются друг от друга точкой с запятой. В одной строке программы можно записать несколько операторов. И наоборот, один оператор может размещаться на нескольких строках.

Составной оператор – это совокупность последовательно выполняемых операторов, заключенных в операторные скобки BEGIN и END.

Внутри операторных скобок операторы также отделяются друг от друга точкой с запятой. BEGIN – это не оператор. Поэтому после него точка с запятой не ставится. Перед END точка с запятой допускается, но ее наличие необязательно.

Составной оператор может понадобиться в тех случаях, когда в соответствии с правилами построения конструкций языка допустимо использование только одного оператора, а требуется выполнить несколько операторов. Таким единственным оператором может выступать составной оператор, в который входит ряд операторов, выполняющих требуемые действия.

В дальнейшем - везде, где будет сказано, что можно использовать один оператор, им может быть составной оператор.

Оператор присваивания служит для вычисления значения выражения и присваивания его переменной. Формат оператора: имя переменной:= выражение;

Вычисляется значение выражения, записанного справа, а затем переменной присваивается это значение. Имя переменной и результат должны принадлежать одному типу. Исключение может составлять случай, когда выражение имеет значение целого типа, а имя результата - действительного типа.

Ввод информации осуществляется при помощи процедур:

READ (b1, b2,..., bn);

READLN (b1, b2,..., bn);

READLN;

где b1,b2,..., bn – имена переменных, значения которых вводятся.

READ (b1,b2,...,bn) осуществляет ввод данных.

READLN(b1,b2, ... ,bn) осуществляет ввод данных и обеспечивает переход к началу новой строки. Переменные вводятся последовательно в одной строке и отделяются друг от друга пробелами.

READLN обеспечивает пропуск одной строки и переход к началу новой строки.

Для вывода информации используются процедуры:

WRITE (b1, b2,..., bn);

WRITELN (b1, b2,..., bn);

WRITELN;

где b1,b2,...,bn – выражения, значения которых выводятся (это могут быть идентификаторы переменных или констант).

WRITE (b1,...,bn) выполняет вывод значений, соответствующих выражений, размещая выводимые значения в одной строке.

WRITELN (b1,b2, ... ,bn) выполняет вывод значений и после вывода последнего значения осуществляет переход к новой строке.

WRITELN обеспечивает пропуск строки и переход к началу новой строки.

Значения выражений в списке операторов вывода могут принадлежать к целому вещественному, символьному или логическому типам.

Общий вид записи вывода значений целого типа:

WRITE (b: m);

WRITELN (b: m);

а для вывода действительного типа:

WRITE (b: m: n);

WRITELN (b: m: n);

где b – арифметическое выражение;

m – поле, отводимое под значение;

n – часть поля, отводимого под дробную часть числа.

Например: writeln (a:6:2, b:0:2, k:4);

Передача управления

Оператор безусловного перехода имеет вид: GOTO n;
где n – метка оператора.

Разветвления в программе возникают при необходимости выбора одного из нескольких возможных путей в решении задачи. Оператор условного перехода позволяет изменить порядок выполнения операторов в программе в зависимости от определенных условий. Общий вид оператора условного перехода:

```
IF <условие> THEN
    <оператор1>
ELSE
    <оператор 2>;
```

Если условие, заданное в операторе IF, истинно, то выполняется THEN-ветвь, т. е. оператор (простой или составной), стоящий после THEN (<оператор 1>). В противном случае выполняется ELSE-ветвь, т.е. оператор, стоящий после ELSE (<оператор 2>). После выполнения одной из ветвей работа программы продолжается с оператора, следующего за IF. Используется также усеченный формат оператора условного перехода:

```
IF <условие> THEN <оператор>;
```

Если в какой-то ветви требуется выполнить более одного оператора, из них необходимо образовать составной оператор, т.е. заключить эти операторы в операторные скобки BEGIN и END.

Алгоритмы циклической структуры

Циклическим называется вычислительный процесс, содержащий многократные вычисления по одним и тем же математическим зависимостям, но для различных значений входящих в него переменных. В Паскале существует три вида циклов.

1. Цикл с предусловием. Действия повторяются, пока условия, приведенные перед началом цикла, истинны.

```
WHILE <условие продолжения цикла> DO
    <Операторы>;
```

Если в цикле необходимо выполнить более одного оператора, то их следует заключить в операторные скобки begin end, т.е. образовать из них составной оператор. До тех пор пока соблюдается условие, последовательно выполняется тело цикла (<оператор>). Если условие не соблюдается, то выполнение программы продолжается, начиная с оператора, следующего за циклом (т.е. в этом случае тело цикла не выполнится ни разу).

Пример:

```
s:=0; x:=0;
while x <=5 do begin
    s:=s+x*x;
    x:=x+0.5
end;
```

2. Цикл с постусловием. Действия повторяются, пока не выполнятся условия, приведенные после окончания цикла.

```
REPEAT
    <Оператор 1>;
    ...
    <Оператор n>
UNTIL <условие выхода из цикла>;
```

Особенности такого цикла:

- тело цикла выполнится как минимум один раз;
- не требует дополнительных операторных скобок (begin - end).

Пример:

```
s:=0; x:=0;
repeat
    s:=s+x*x;
    x:=x+0.5
until x > 5;
```

3. Цикл с параметром (со счетчиком, итерационный). Действия повторяются, пока переменная-«счётчик» не достигнет последнего значения.

```
FOR <переменная>:=<нач.знач.> TO <кон.знач.> DO <Операторы>;
```

Особенности такого цикла:

- не нужно принудительно изменять значений переменной цикла;
- переменная цикла может быть только целого типа;
- шаг изменения переменной равен 1 или -1.

Пример:

```
s:=0; for i:=0 to 10 do s:=s+sqr(i/2);

s:=0; for i:=10 downto 0 do s:=s+sqr(i/2);
```

Лекция 3. Нестандартные и ограниченные типы данных

Новые типы описываются в специальном разделе типов или определяются непосредственно при описании переменных.

Перечисляемый тип задается перечислением значений, которые может принимать переменная. Общая форма задания перечисляемого типа такова:

```
TYPE T = (A1, A2,..., AN);
```

Здесь T – имя нового типа; A1,A2,...,AN определяют константы нового типа данных. Последовательность значений, составляющая перечисляемый тип, упорядочена.

Например:

```
Type Season = (winter, spring, summer, autumn);
```

```
Var S: Season;
```

```
...
```

```
    If S = Summer then S:=Autumn;
```

```
...
```

Для значений перечисляемых типов определены стандартные функции SUCC и PRED.

Отрезок значений любого порядкового типа может быть определен как ограниченный тип (тип диапазон).

Общий вид

```
TYPE T=MIN..MAX;
```

Здесь T – имя типа; MIN, MAX - левая и правая границы диапазона.

Например:

```
TYPE DEN=1..31; GOD= 1900..2000;
```

```
VAR X: DEN; Y: GOD; C: 1.. 12;
```

Оператор варианта является обобщением условного оператора: он дает возможность выполнить один из нескольких операторов в зависимости от значения некоторого выражения, называемого селектором. В общем случае оператор имеет вид:

```
CASE < селектор > OF
```

```
< список меток 1 >:< Оператор 1 >;
```

```
< список меток 2 >:< Оператор 2 >;
```

```
...
```

```
< список меток n >: < Оператор n >;
```

```
[ ELSE
```

```
< оператор >]
```

```
END;
```

где

< селектор > – выражение любого типа, кроме вещественного;
 < оператор > – любой оператор языка, в том числе и составной;
 < список меток > – список разделенных запятыми значений выражения
 < селектор > или одно его значение.

Оператор варианта выбирает для исполнения тот оператор, одна из меток которого равна текущему значению выражения < селектор >. Если значение выражения < селектор > не совпадает ни с одной из меток, то выполняется оператор, соответствующий ELSE. Ветвь ELSE необязательна. По окончании выполнения выбранного оператора управление передается в конец оператора CASE. Селектор и метки должны быть одного и того же типа.

Пример:

CASE k OF

1: y:=k;

2..5: y:=k*k;

6, 7, 9: y:=ln(k);

ELSE y:=0

END;

Лекция 4. Массивы и подпрограммы

Базируясь на простых типах, можно строить более сложные – структурированные типы данных. К ним относятся массивы, записи и последовательности (файлы).

Массив состоит из фиксированного числа компонент одного типа, характеризуется именем и размерностью.

Общий вид описания типа массив:

TYPE T = ARRAY [T1, T2 ... Tk] OF TC;

где T – имя массива;

T1,..., Tk – индексы;

TC – базовый тип.

Количество индексов k определяет размерность массива. Индексы задаются в виде ограниченного типа данных (диапазона).

Массивы можно описывать непосредственно в разделе описаний переменных, например:

Var m: array [1..10] of integer;

Доступ к любому элементу массива осуществляется указанием идентификатора массива, за которым в квадратных скобках следует индексное выражение. Индексное выражение должно давать значения, лежащие в диапазоне, определяемом типом индекса. Для объявленного в примере массива M в программе доступны следующие индексные переменные: M[1], M[2], ... , M[10].

Количество индексов, необходимое при обращении к элементу массива, определяет k-мерность массива. При $k = 1$ массив называется *одномерным*. При $k = 2$ - *двумерным*. Одномерный массив соответствует понятию вектора (линейной таблицы, строки), двумерный массив – понятию матрицы (набора векторов, прямоугольной таблицы).

Селективная обработка массива – это выделение из массива элементов, удовлетворяющих условию и обработка выделенных фрагментов. Часто из выделенных элементов формируют новый (рабочий) массив и его обрабатывают.

Наиболее часто встречаются такие условия обработки элементов массива:

четные	$A[i] \bmod 2 = 0$
нечетные	$A[i] \bmod 2 < > 0$
кратные k	$A[i] \bmod k = 0$
некратные k	$A[i] \bmod k < > 0$
на четных местах	$i \bmod 2 = 0$
на нечетных местах	$i \bmod 2 < > 0$
положительные	$A[i] > 0$
неотрицательные	$A[i] \geq 0$
в интервале (x1,x2)	$(A[i] > x1) \text{ and } (A[i] < x2)$

Нахождение наибольшего и наименьшего значений массива выполняется в цикле, в котором текущий элемент массива сравнивается с наибольшим (наименьшим) из всех предыдущих элементов массива. Если текущий элемент окажется больше (меньше) наибольшего (наименьшего) из предыдущих, то его надо считать новым наибольшим (наименьшим). В противном случае наибольшее (наименьшее) сохраняет старое значение, т.е.

$$Y_{\max} = \begin{cases} Y_i & \text{если } Y_i > Y_{\max} \\ Y_{\max} & \text{если } Y_i \leq Y_{\max} \end{cases}$$

$$Y_{\min} = \begin{cases} Y_i & \text{если } Y_i < Y_{\min} \\ Y_{\min} & \text{если } Y_i \geq Y_{\min} \end{cases}$$

Если телом цикла является циклическая структура, то такие циклы называют вложенными или сложными. Цикл, содержащий в себе другой цикл, называют внешним. Цикл, содержащийся в теле другого цикла, называют внутренним.

Двухмерный массив характеризуется именем и размерностью, где первый индекс определяет максимальное число строк, а второй – максимальное число столбцов. Элемент массива – переменная, расположенная на пересечении столбца и строки. При обработке таких массивов обычно и применяют вложенные циклы.

Пример:

Var m: array [1..3,1..3] of integer;

Ссылка на элемент матрицы M, лежащий на пересечении i-й строки и j-го столбца, имеет вид M[i,j].

При обработке матриц часто приходится выделять элементы:

- k-й строки	A[k, j]	j=1,...,m
- k-й колонки	A[i, k]	i=1,...,n

для квадратных матриц (m=n) также:

главной диагонали	A[i, i]	i=1,...,n
побочной диагонали	A[i, n+1-i]	i=1,...,n
наддиагональные	A[i, j]	i > j
поддиагональные	A[i, j]	i < j

Процедуры и функции

В языке Pascal предусмотрена возможность объединения любой последовательности операторов (как повторяющегося в разных местах программы фрагмента, так и просто логически самостоятельного фрагмента) в отдельную подпрограмму, к которой можно многократно обращаться (вызывать) из любого места основной программы, освобождая последнюю от вынесенных в подпрограмму фрагментов.

Обращение к подпрограмме осуществляется по ее имени. При этом выполнение основной программы будет временно приостановлено, выполняться все действия подпрограммы, и выполнение основной программы снова будет продолжено, начиная с оператора, следующего за оператором вызова подпрограммы.

Таким образом, за именем подпрограммы скрываются некоторые вынесенные из программы действия, которые иницируются автоматически с появлением в программе обращения к этой подпрограмме.

Любая подпрограмма, в свою очередь, может обращаться к другим подпрограммам, а также к самой себе (явление рекурсии).

Структура подпрограммы аналогична структуре всей программы, т.е. в ней есть заголовок, раздел описаний и тело подпрограммы. Подпрограмма должна быть описана до того, как она будет использована в программе или другой подпрограмме.

Все переменные, используемые в подпрограмме, можно разделить на три категории:

- локальные переменные описываются и используются только внутри подпрограммы;
- глобальные переменные описываются в основной программе, могут использоваться как в программе, так и во всех ее подпрограммах;
- формальные параметры – зарезервированные в подпрограмме переменные, в которые при обращении к программе будут занесены из внешней программы конкретные данные или их адреса.

В Паскале имеются две разновидности подпрограмм – процедуры и функции.

В простейшем случае процедура может представлять собой лишь поименованную группу операторов.

```
Procedure Switch1;
begin
    r:=x; x:=y; y:=r
end;
```

Такая процедура не имеет ни параметров, ни локальных переменных. Все используемые в ней переменные (в нашем примере это x, y и r) являются глобальными, т.е. они должны быть описаны во внешней (основной) программе.

В примере процедура осуществляет обмен значениями переменных x и y. Ясно, что эти переменные должны быть доступны внешней программе, т.е. объявление их глобальными оправдано. А вот переменная r – это рабочая переменная, используемая для временного хранения.

```
Procedure Switch2;
var r: real;
begin
    r:=x; x:=y; y:=r
end;
```

Однако полученная процедура «жестко» привязана к глобальным переменным x и y. Поэтому данные в процедуру (и из процедуры) нужно передавать как параметры:

```
Procedure Switch3 (var x: real; var y: real);
var r: real;
begin
    r:=x; x:=y; y:=r
end;
```


Пример. Процедура ввода N целых чисел.

Пусть в основной программе определен тип:

```
TYPE mas=array [1..100] of integer;
```

```
Procedure input (var m: mas; n: integer); {заголовок процедуры}
var i: integer; {локальная переменная - параметр цикла}
begin
    writeln ('введите', n:3, ' целых чисел');
    for i:=1 to n do read(m[i]);
end;
```

Для вызова процедуры в основной программе или другой подпрограмме следует записать оператор, состоящий из имени процедуры, и заключенного в скобки списка фактических параметров, которые должны совпадать по количеству и типу с формальными параметрами процедуры.

Вызов Input (m, 10) означает, что вызывается процедура для ввода 10 целых чисел в массив M.

Функция предназначена для вычисления какого-либо значения. У этой подпрограммы два основных отличия от процедур:

- заголовок состоит из слова FUNCTION, за которым следует имя функции, список формальных параметров и тип функции (т.е. тип возвращаемого значения);
- в теле функции хотя бы один раз имени функции должно быть присвоено значение (т.е. значение, которое должно быть передано в основную программу в качестве результата работы функции).

Формальные параметры подпрограммы указывают, с какими параметрами следует обращаться к этой подпрограмме. При обращении к подпрограмме формальные параметры заменяются на эквивалентные фактические параметры вызывающей программы или подпрограммы, типы соответствующих фактических и формальных параметров должны быть идентичны.

Все формальные параметры можно разбить на две категории:

- параметры-значения (эти параметры в основной программе подпрограммой не меняются);
- параметры-переменные (эти параметры подпрограмма может изменить в основной программе).

Параметр-значение передается основной программой в подпрограмму в виде копии своего значения и, следовательно, подпрограммой измениться не может, точнее, все изменения параметра-значения, выполненные в подпрограмме, при выходе из нее теряются и в основную программу не возвращаются.

Параметр-значение указывается в заголовке подпрограммы своим именем и через двоеточие – типом. Если параметров-значений одного типа несколько, их можно перечислить через запятую, а затем уже указать общий тип. Отдельные группы параметров отделяются друг от друга точкой с запятой.

В качестве фактического параметра на месте параметра-значения при вызове подпрограммы может выступать любое выражение совместимого для присваивания типа. Это выражение вычисляется, и полученное значение передается подпрограмме.

Параметры-переменные передаются основной программой в подпрограмму своими адресами. Следовательно, подпрограмма имеет доступ к этим параметрам и может их изменять. При выходе из подпрограммы новое значение параметра-переменной возвращается основной программе.

Параметры-переменные указываются в заголовке подпрограммы аналогично параметрам-значениям, но только перед именем параметра записывается зарезервированное слово VAR. Действие слова VAR распространяется до ближайшей точки с запятой. При вызове подпрограммы в качестве фактического параметра на месте параметра-переменной должна использоваться только переменная идентичного типа. Использование выражения здесь недопустимо.

Например, в описанной выше процедуре Input параметр M является параметром-переменной, а параметр N - параметром-значением. Поэтому к этой процедуре допустимы такие обращения:

```
Input (M, 10)
Input (M, k+2)
Input (A, k+SQR(m))
```

Пример. Найти произведение средних арифметических положительных значений массивов A(10), B(20), C(15).

```
PROGRAM ZADAN1;
  uses crt;
  type mas=array [1..20] of real;
  var A, B, C: mas;  i: integer;  sa, sb, sc: real;
  ccc: char;

                                {Процедура ввода}
  PROCEDURE InpMas (var x: mas; k: integer);
  Begin
    writeln ('введите массив из', k:3, ' чисел');
    for i:=1 to k do read (x[i]);
    writeln;
  end;

                                {Процедура вывода}
  PROCEDURE OutMas (x:mas; k:integer);
  Begin
    writeln ('исходный массив:');
    for i:=1 to k do write (x[i]:5:2);
    writeln;
  end;
```

```

                                {Функция расчета среднего арифметического}
FUNCTION Sred(x:mas; k:integer):real;
var s:real; kol:integer;
begin
s:=0; kol:=0;
for i:=1 to k do
if x[i]>0 then
begin
s:=s+x[i]; kol:=kol+1;
end;
if kol=0 then
writeln ('в массиве нет положительных чисел')
else
begin
s:=s/kol; writeln ('среднее арифметич. ', s:6:4);
end;
writeln;
sred:=s;
end;
                                {Основная программа}
Begin
InpMas (A, 10);
InpMas (B, 20);
InpMas (C, 15);
OutMas (A, 10);
SA:=Sred(A, 10);
OutMas (B, 20);
SB:=Sred(B, 20);
OutMas (C, 15);
SC:=Sred(C, 15);
writeln ('Результат: ',SA*SB*SC:8:4);
CCC:=ReadKey;
End.

```

Лекция 5. Обработка символьных данных

В языке PASCAL переменная типа CHAR (символьная переменная) может принимать значения из некоторой упорядоченной совокупности (множества) символов ЭВМ с последовательности ASCII-символов):

```
...<0<1<...<9<...<A<B<...<Z<...<a<b<c<...<z<....  
...<A<Б<В...<Я<...<a<б<...<я<...
```

Каждому символу из этой совокупности соответствует некоторый код (называемый кодом ASCII) - порядковый номер символа в последовательности. Символьные данные описываются ключевым словом CHAR, например:

```
var c, b, simv, S:char;  
var A:ARRAY[1..N] of char;
```

Для символьных данных определены операции отношения =, <, >, < > и присвоения :=, а также функции:

ORD(x) - выдает код символа x

CHR(i) - выдает i-й символ последовательности ASCII, т.е. по коду i выдает символ

SUCC(x) и PRED(x) - выдает последующий или предыдущий (соответственно) символ в ASCII - последовательности.

Значением символьной переменной (типа char) является только один символ.

Тип STRING (строка) – последовательность символов произвольной длины (но не более 255 символов), которую можно рассматривать как одномерный массив символов ARRAY [0...n] of char.

При объявлении типа STRING[n] определяется максимальное количество символов в строке. Если N не указано, принимается максимально возможное значение – 255.

К каждому символу в строке можно обращаться как к элементу одномерного массива, например:

```
var st: string;  
...  
for i:=1 to length(st) do if st[i]='+' then st[i]:='-';
```

Первый байт в этом массиве имеет индекс 0 и содержит текущую длину строки. Первый значимый символ строки занимает второй байт и имеет индекс 1. Над длиной строки можно выполнять требуемые действия (например, изменять). Текущую длину строки можно также получить с помощью функции LENGTH(st).

К строкам можно применять операции сцепления (конкатенации) "+". Строки также можно сравнивать, к тому же строки при сравнении могут иметь разные длины, сравнение выполняется слева направо в соответствии с кодами символов, считается, что отсутствующий символ в более короткой

строке имеет код меньший, чем любой код действительного символа. Например: 'x' меньше чем 'xs'.

При обработке строчных данных используются следующие процедуры и функции.

1. COPY (ST, N, K) – скопировать (т.е. создать новую строку) из строки ST K символов, начиная с символа с номером N, если N больше длины строки ST, возвращается пустая строка.

2. POS (ST1, ST) – найти в строке ST первое вхождение подстроки ST1 и вернуть номер позиции, с которой она начинается, если подстрока не найдена, возвращается нулевое значение.

3. LENGTH (ST) – вернуть текущую длину строки ST.

4. DELETE (ST, N, K) – процедура, которая удаляет из строки ST подстроку в K символов, начиная с символа N.

5. INSERT (ST1, ST, N) – процедура, которая вставляет подстроку ST1 в строку ST, начиная с символа с номером N, если полученная строка содержит более 255 символов, все символы после 255-го отбрасываются.

6. STR(X, ST) – процедура, которая преобразует число x в строковую переменную, результат заносится в строку ST.

7. VAL (ST, X, Kod) – процедура, которая преобразует символьную величину типа STRING из строки в ее численное представление и результат помещает в переменную X. Если строка записана с ошибкой, то индекс (позиция) неверного символа заносится в переменную Kod, иначе Kod равен нулю.

Лекция 6. Тип запись, файлы и файловые типы данных

Запись – это структурированный тип данных, состоящий из фиксированного числа компонент, называемых полями. В одном поле данные имеют один и тот же тип, а в разных полях могут иметь различные типы.

В записи каждое поле имеет свое имя (идентификатор). Количество полей, тип и идентификатор каждого поля фиксируются в определении типа, к которому относится запись. Определение начинается с ключевого слова RECORD, за которым следует перечисление всех полей с указанием через двоеточие их типов.

Поля отделяются друг от друга точкой с запятой. Поля записи могут быть любого типа. Если несколько полей имеют один и тот же тип, то имена полей можно перечислять через запятую и затем указать общий тип. Завершает определение типа записи ключевое слово END.

Общий вид описания типа записи:

Type T = RECORD

I1: T1;

I2: T2;

.....

IN: TN

End;

Можно задать переменные этого типа:

```
var A,..., B: T;
```

Здесь:

T – идентификатор типа;

Ik – имена полей или список разделенных запятыми имен полей;

Tk – типы полей;

A,..., B – идентификаторы (имена) переменных типа записи.

Пример. Ввести информацию о студентах. Запись о каждом человеке содержит значение фамилии, имени, отчества, даты рождения, группы и среднего балла.

```
Type Date = Record
    Day, Month, Year: Integer
End;
Student = Record
    Fam, Name, Parent: String [20];
    DR: Date;
    Group: String [10];
    SR: Real
End;
Var S: Student;
    SM: array [1..25] of Student;
```

Доступ к полям переменной типа записи осуществляется указанием имени переменной и имени поля, записываемого через точку:

```
S.Fam:='Иванов';
S.DR.Day:=1;
```

Для того чтобы не записывать каждый раз имя переменной при обращении к полям записи, можно использовать оператор объединения WITH:

```
For i:=1 to 25 do
    With SM[i] do Begin
        Readln(Fam, Name, Parent, Group);
        Readln(SR);
    End;
```

Понятие файла тесно связано с размещением информации на внешних устройствах (магнитных дисках, флэш-накопителях и т.п.).

Файлом называется последовательность компонентов одного типа. Компоненты файла могут быть любого типа, за исключением типа файла. Число компонентов в файле теоретически не ограничено.

Над файловыми переменными нельзя выполнять никаких операций (присваивать значения, сравнивать и т.д.). Файловые переменные можно лишь использовать для чтения информации из файла и записи информации в файл.

Общий вид описания файлового типа:

Type T = FILE of TK;

Здесь: T – идентификатор типа файла;

TK – тип компонента.

Var F : T ;

Тип файл может быть описан и непосредственно при описании переменной типа файл:

Var F : FILE of TK;

В языке Pascal ввод-вывод информации на внешний файл осуществляется только через файловые переменные.

Перед тем, как осуществлять ввод-вывод, файловая переменная должна быть связана с конкретным внешним файлом с помощью стандартной процедуры ASSIGN (см. ниже). Затем файл должен быть открыт для чтения или записи. После этого можно осуществлять организацию ввода-вывода.

В каждый конкретный момент для обработки доступен только один компонент файла (текущий), т.е. на этот компонент установлен «текущий указатель» файла.

Обычно все файлы считаются файлами последовательного доступа. Это означает, что начать писать в файл нужно только с самого его начала, дописывая новые компоненты последовательно, один за другим. Для чтения также надо начать просмотр файла с самого начала, однако с помощью стандартной процедуры SEEK можно установить режим произвольного доступа, устанавливая указатель текущей позиции файла на требуемый компонент.

Для того, чтобы определить готовность файла к записи либо к чтению информации, существует стандартная функция EOF(F). Если указатель текущей позиции файла продвинулся за конец файла (готовность к записи), то эта функция принимает значение TRUE (истина), в остальных случаях – FALSE (ложь). Функцию EOF можно использовать в условном операторе IF или в операторе цикла WHILE. Если нужно использовать не условие конца файла, а условие того, что файл не закончен (готовность к чтению), то можно применять, например, следующую конструкцию:

```
while not EOF(F) do
begin
.....
READ (F,A);
.....
end;
```

В этой конструкции осуществляется последовательное чтение (и обработка) компонентов файла до тех пор; пока указатель текущей позиции указывает на очередной компонент (т.е. в файле есть еще не обработанные компоненты). Цикл завершится, как только указатель текущей позиции продвинется за конец файла (т.е. будут обработаны все компоненты файла).

После работы с файлом его необходимо закрыть с помощью стандартной процедуры CLOSE.

Для работы с файлами используются следующие стандартные процедуры (в дальнейшем подразумевается, что F - идентификатор типа файл):

1. ASSIGN (F, ST) – связывает файловую переменную F с внешним файлом, имя которого задается в строковом выражении (типа STRING) ST. Имя файла должно отвечать требованиям MS-DOS и может содержать маршрут (имя устройства и имена каталогов). Все последующие операции с F будут производиться над внешним файлом с именем, заданным в ST.

2. REWRITE (F) – создает и открывает для записи новый файл с именем, связанным с переменной F процедурой ASSIGN. Если файл с таким именем существует, он стирается и вместо него создается новый пустой файл. Если файл с таким именем открыт, он закрывается, стирается и создается заново. Указатель текущей позиции файла устанавливается в начало пустого файла (на признак конца файла), функции EOF(F) присваивается значение TRUE.

3. WRITE (F, A) – записывает значения переменной A в файл F и перемещает указатель текущей позиции файла на следующий компонент.

4. RESET (F) – открывает существующий файл с именем, связанным с переменной F. Файл с таким именем должен существовать. Если файл уже открыт, он вначале закрывается, а затем открывается вновь. Указатель текущей позиции файла устанавливается в начальное положение.

5. Read (F, A) – считывает из файла текущий компонент, присваивает его значение переменной A и перемещает указатель текущей позиции файла на следующий компонент. Если EOF (F) = True, то при попытке чтения возникает ошибка.

6. Seek (F, n) – перемещает указатель текущей позиции файла на компонент с номером n. Первый компонент файла имеет номер 0. Для перемещения на конец файла можно использовать процедуру Seek(F,FileSize(F)).

7. Close (F) – закрывает открытый файл, связан с переменной F.

Также при работе с файлами можно использовать следующие функции:

1. EOF (F) – возвращает статус конца файла. Результат функции TRUE (истина), если указатель текшей позиции файла находится на признаке конца файла т.е., за последним компонентом файла, и FALSE (ложь) – в противном случае.

2. FileSize (F) – возвращает текущий размер файла, т.е. количество компонентов в файле. Если файл пустой, FileSize (F)=0.

3. FilePos (F) – возвращает текущее значение указателя файла. Если текущее значение указателя соответствует началу файла, FilePos возвращает 0, если концу файла (т.е. EOF(F) = True), то FilePos (F) = FileSize (F).

При формировании файла (выводе в файл) обычно используется следующая последовательность стандартных процедур и операторов:

ASSIGN – связать файловую переменную с внешним файлом.

REWRITE – открыть файл для записи.

WRITE – вывести (в цикле) компоненты в файл.

CLOSE – закрыть файл.

Например:

```
assign (F,...);
```

```
Rewrite (F);
```

```
{оператор цикла, например While или for}
```

```
begin
```

```
.....  
write (F,...);
```

```
end;
```

```
close (F);
```

Для чтения из файла обычно применяется такая последовательность:

ASSIGN – связать файловую переменную с внешним файлом.

RESET – открыть файл для чтения.

READ – считать (в цикле) компоненты из файла.

CLOSE – закрыть файл.

Например:

```
assign (F,...);
```

```
reset (F);
```

```
while not EOF(F) do
```

```
begin
```

```
....  
read (F,...);
```

```
End;
```

```
close (F);
```

Если в одной и той же программе осуществляется и ввод в файл, и чтение из него, то оператор ASSIGN достаточно записать один раз.

На практике очень часто компонентами файла являются записи. В этом случае файловую переменную можно описать, например, следующим образом:

```
Var F: File of Student;
```

Кроме типизированных файлов, в Паскале используется еще два вида файлов. Для вывода текстовой информации используются текстовые файлы, вывод в которые аналогичен выводу на экран. Текстовый файл не позволяет произвольно установить текущий указатель или вычислить размер файла, но позволяет использовать процедуру APPEND (открыть файл для добавления).

```

Var G: Text;
...
    Assign (G, 'Out.txt');
    Reset (G); Append (G);
    ...
    Writeln(G, 'x=', x:6:2, ' y=', y:6:2);
    ...
    Close (G);

```

Бестиповые (нетипизированные) файлы позволяют записывать на диск произвольные участки памяти ЭВМ и считывать их с диска в память. Операции обмена с бестиповыми файлами осуществляется с помощью процедур BlockRead и BlockWrite. Кроме того, вводится расширенная форма процедур Reset и Rewrite. В остальном принципы работы остаются такими же, как и с компонентными файлами.

Перед использованием логический файл

```
var f: File;
```

должен быть связан с физическим с помощью процедуры Assign. Далее файл должен быть открыт для чтения или для записи процедурой Reset или Rewrite, а после окончания работы закрыт процедурой Close.

При открытии файла длина буфера устанавливается по умолчанию в 128 байт. TURBO PASCAL позволяет изменить размер буфера ввода - вывода, для чего следует открывать файл расширенной записью процедур

```
Reset(var f: File; BufSize: Word)
```

или

```
Rewrite(var f: File; BufSize: Word)
```

Параметр BufSize задает число байтов, считываемых из файла или записываемых в него за одно обращение. Минимальное значение BufSize - 1 байт, максимальное - 64 К байт.

Чтение данных из бестипового файла осуществляется процедурой

```
BlockRead(var f: File; var X; Count: Word; var QuantBlock: Word);
```

Эта процедура осуществляет за одно обращение чтение в переменную X количество блоков, заданное параметром Count, при этом длина блока равна длине буфера. Значение Count не может быть меньше 1. За одно обращение нельзя прочесть больше, чем 64 К байтов.

Необязательный параметр QuantBlock возвращает число блоков (буфров), прочитанных текущей операцией BlockRead. В случае успешного завершения операции чтения QuantBlock = Count, в случае аварийной ситуации параметр QuantBlock будет содержать число удачно прочитанных блоков. Отсюда следует, что с помощью параметра QuantBlock можно контролировать правильность выполнения операции чтения.

Запись данных в бестиповой файл выполняется процедурой

BlockWrite(var f: File; var X; Count: Word; var QuantBlock: Word);
которая осуществляет за одно обращение запись из переменной X количества блоков, заданное параметром Count, при этом длина блока равна длине буфера. Необязательный параметр QuantBlock возвращает число блоков (буферов), записанных успешно текущей операцией BlockWrite.

Лекция 7. Модули в Турбо-Паскале

Чтобы сделать основную программу меньше и понятней для прочтения, обычно рекомендуется все используемые в ней подпрограммы выносить в отдельный файл, называемый модулем. Модуль имеет следующую структуру:

```
UNIT <имя>;
USES <список используемых модулей>;
INTERFACE
    <первый раздел описаний>
IMPLEMENTATION
    <второй раздел описаний>
[ BEGIN <секция инициализации> ]
END.
```

Здесь: UNIT – заголовок модуля, который обязательно должен совпадать с именем файла модуля;

INTERFACE – описываются константы, переменные и заголовки подпрограмм, которые доступны как непосредственно модулю, так и обращающимся к нему программам;

IMPLEMENTATION – описываются константы, переменные и заголовки подпрограмм, которые доступны только в пределах модуля, а также тела подпрограмм, описанных в секции «Interface».

Секция инициализации, которая может отсутствовать, содержит операторы, выполняющиеся перед началом работы вызывающей модуль программы.

Подключение модуля осуществляется командой USES <имя>:

```
Uses Unit1, Unit2, Unit3;
```

Если в интерфейсных секциях двух модулей обнаружатся одноименные идентификаторы, то по умолчанию будет использован идентификатор из модуля, приведенного в списке позднее (правее). Допустимо явное обращение через точку: «имя модуля . имя идентификатора».

Лекция 8. Использование стандартных модулей в Турбо-Паскале

Библиотеки подпрограмм Паскаля содержат несколько встроенных модулей для работы с экраном как в текстовом, так и в графическом режимах. Например, при использовании модуля CRT (после заголовка программы необходимо написать «uses crt») можно вызывать процедуру очистки экрана (clrscr), задержки выполнения программы до нажатия любой клавиши (readkey) или заданное число миллисекунд (delay).

Модуль для работы с графикой GRAPH содержит множество процедур и функций, позволяющих рисовать на экране графические объекты. Часть из них приведена в таблице. Необходимо помнить, что любая работа с графикой в Паскале начинается с инициализации соответствующего режима процедурой InitGraph(Gd,Gm,Path), где Gd,Gm:integer, (номера режимов), Path:string (путь к графическим драйверам), а заканчивается процедурой CloseGraph. Левый верхний угол экрана имеет координаты (0,0), правый нижний – (639,479) или (799,599).

Некоторые процедуры и функции модуля Graph
(var fills:array[1..4] of PointType;)

Название	Описание
InitGraph(Gd,Gm,"")	Инициализация графического режима
Rectangle(x1,y1,x2,y2)	Прямоугольник (заданы координаты двух вершин)
Bar(x1,y1,x2,y2)	Закрашенный прямоугольник
Circle(x,y,r)	Окружность с центром в (x,y) и радиусом r
Arc(x,y,a1,a2,r)	Дуга окружности от угла a1 до a2
PieSlice(x,y,a1,a2,r)	Закрашенная дуга окружности
SetFillStyle(t,c)	Установить заливку номером t и цветом c
Заливка участка	FillPoly(SizeOf(fills) div SizeOf(PointType), fills);
Line(x1,y1,x2,y2)	Прямая линия из точки (x1,y1) в (x2,y2)
LineTo(x2,y2)	Прямая линия из текущего места в точку (x2,y2)
MoveTo(x2,y2)	Перенести текущий указатель в точку (x2,y2)
PutPixel(x,y,color)	В точке (x,y) нарисовать точку цветом color (0..7)
C:=GetPixel(x,y)	Извлечь номер цвета в заданной точке
SetColor(c)	Установить цвет линий или заливки
C:=GetColor	Извлечь цвет линий или заливки
SetBkColor(c)	Установить цвет фона
C:=GetBkColor	Извлечь цвет фона
C:=GetMaxColor	Извлечь номер белого цвета
CloseGraph	Заккрытие графического режима

Полный перечень подпрограмм модуля Graph можно получить, нажав Ctrl+F1 на любом из набранных имен.

Лекция 9. Базовые положения ООП и их использование в Турбо-Паскале

Задание данных вместе с процедурами и функциями их обработки превращает их в новый тип данных – объект. Структурно объект напоминает тип запись, полями которой могут быть подпрограммы. При этом поля объекта называются свойствами, а процедуры и функции для их обработки – методами.

В качестве примера опишем объект, позволяющий открыть и закрыть графический режим.

```

program test1;
type gr=object
  gd,gm:integer;
  procedure graphinit;
  procedure graphclose;
  end;
procedure gr.graphinit;
begin
  gd:=detect;initgraph(gd,gm,"");
  if graphresult <> 0 then begin
    writeln('Невозможно открыть графический режим!');
    halt(1) end
end;
procedure gr.graphclose;
begin
  closegraph
end;
var p:gr; {p – это переменная типа object, или экземпляр объекта}
{программа просто рисует точку в центре экрана}
begin
  with p do begin
    graphinit;
    putpixel(round(getmaxx/2), round(getmaxy/2),7);
    readln; {ожидание нажатия клавиши Enter}
    graphclose
  end;
end.

```

Принципы объектно-ориентированного программирования позволяют использовать любой объект для порождения иерархии «объектов-потомков» с наследованием доступа каждого из них к коду и данным предка. То есть, например, для рисования точки в центре экрана мы можем преобразовать вышеописанную программу путем внесения нового объекта как потомка объекта gr:

```

program test2;
type gr=object
  gd,gm:integer;
  procedure graphinit;
  procedure graphclose;
  end;
procedure gr.graphinit;
begin
  gd:=detect;initgraph(gd,gm,"");
  if graphresult <> 0 then begin
    writeln('Невозможно открыть графический режим!');
    halt(1) end
end;
procedure gr.graphclose;
begin
  closegraph
end;
point=object(gr)
  x,y:integer; {координаты точки}
  procedure setd(px,py:integer); {задание координат}
  procedure show; {нарисовать точку}
  procedure hide; {скрыть точку – нарисовать ее цветом фона}
  end;
procedure point.setd(px,py:integer);
begin
  x:=px;y:=py;
end;
procedure point.show;
begin
  putpixel(x,y,getmaxcolor);
end;
procedure point.hide;
begin
  putpixel(x,y,getbkcolor);
end;
var p:point; {p – это экземпляр объекта point}
{программа рисует точку в центре экрана, а потом стирает ее}
begin
  with p do begin

```

```

graphinit; {обратите внимание: этот метод используется как свой собствен-
ный}
setd(round(getmaxx/2), round(getmaxy/2)); {задаем координаты точки}
show; {рисую точку}
readln; {ожидание нажатия клавиши Enter}
hide; {скрываем точку}
readln; {ожидание нажатия клавиши Enter}
graphclose
end;
end.

```

Модуль 2

Основы программирования на языке Турбо-Си

Лекция №10. Основные данные о языке программирования Си

Особенности языка Си:

- ключевые определения языка Си повторяют свои аналоги в Паскале;
- все идентификаторы регистрозависимы;
- точка с запятой (;) представляет собой не разграничение операторов и описаний, а окончание оператора или описания;
- вместо операторных скобок begin – end используется блок {}
- присвоение значения имеет вид = (знак «равно»);
- сравнение данных: равно (==), не равно (!=);
- логические операции: ИЛИ (||), И (&&) и НЕТ (!), например: ((x>0)|| (y>0))&&(z>0);
- комментарии: /* комментарий */ или // до конца строки.

Си предоставляет возможность сокращения записи везде, где это возможно (табл.1):

Таблица 1

Паскаль	Си
k:=k+1	k++
k:=k-1	k--
k:=k+2	k+=2
k:=k-2	k-=2
k:=k*2	k*=2
k:=k/2	k/=2
k:=k div n	k%=n
m:=k; k:=k+1	m=k++

k:=k+1;m:=k	m=++k
-------------	-------

Остальные операции будут рассмотрены в других разделах.
Основные типы данных приведены в таблице 2.

Таблица 2

Тип данных	Размер, бит	Диапазон значений
void	0	Нет данных
char	8	-128...127
unsigned char	8	0...255
int	16	-32768...32767
unsigned int	16	0...65535
long	32	±2 млрд.
unsigned long	32	0..4 млрд.
float	32	±3.4E±38
double	64	±1.7E±308
long double	80	±3.4E±4932

Описание переменных происходит по шаблону: *тип имя*;

Например: *int k*; или *int k=5*;

Перед типом можно указать класс памяти, к которому относится переменная: auto, register, static, extern.

Область действия переменных: блок, функция, программа.

Преобразование типов: (*имя_типа*) *операнд* или *имя_типа (операнд)*.

Например: *int (k)* или (*unsigned int*) *k*.

Лекция 11. Разновидности вычислительных процессов

Ввод данных: scanf(формат, список_адресов_переменных);

Например: scanf("%i %f",&k,&a);

Вывод данных: printf(формат, список_переменных);

Например: printf("k=%3i, a=%5.2f",k,a);

Пример типовой программы для вычисления значений функции:

```
// вычисление значения функции
#include <stdio.h>
#include <math.h>
main()
{
```



```

float x,y,z;
printf("Введите X:\n");
scanf("%f",&x);
y=sqrt(sin(x)+1);
z=pow(2,abs(x-y))*(tan(y)+1);
printf("Результат: x=%5.2f y=%5.2f z=%5.2f\n",x,y,z);
}

```

Операция выбора: **Выражение1 ? выражение2 : выражение3**

Оператор выбора: **if (выражение) оператор1 [else оператор2]**

Переключатель:

```

switch (выражение)
{ case выражение_1: операторы_1;
  case выражение_2: операторы_2;
  ...
  case выражение_n: операторы_n;
  default: операторы;
}

```

Перечисления выражений, переходы внутри переключателя (использование **break**).

Цикл с предусловием: **while (выражение_условие) {тело_цикла}**

Цикл с постусловием: **do {тело_цикла} while (выражение_условие);**

Итерационный цикл:

for (инициализация; выражение_условие; список) {тело_цикла}

Особенности итерационного цикла, основные отличия от Паскаль-аналога.

Операторы передачи управления: **goto, return, break, continue**.

Лекция 12. Специфика языка программирования Си

Описание указателя: **тип *имя**. **NULL** – указатель, который никуда не указывает. Сдвиг указателя (отличие **r++** от ***r++**).

Структура функции: **тип имя (формальные_параметры) {тело_функции}**

Оператор возврата в точку вызова. Тип функции, которая ничего не возвращает. Особенности обращения к функции без параметров. Особенности соответствия числа формальных и фактических параметров, а также их типов.

Прототип функции, его расположение и смысл использования.

Функции с переменным количеством параметров: описание и работа (два способа решения проблемы).

Перегрузка функций. Шаблоны функций.

Лекция 13. Массивы и обработка символьных данных

Описание массива: **тип имя[константа]**, нумерация.

Варианты описания и задания начальных значений: **int a[5]; int b[5]={1,2,3,4,5}; int c[5]={1,2,3}; int d[]={1,2,3,4,5}; extern int e[];**

Имя массива как указатель на его первый элемент. Варианты обращения к элементам массива: **a[i] ~ *(a+i); a[i] ~ i[a];**

Описание двумерного массива: **тип имя[константа1] [константа2]**, обращение к его элементам: **a[i][j]; *(* (a+i)+j);**

Варианты описания и задания начальных значений: **int a[2][2]; int b[2][2]={1,2,3,4}; int c[2][2]={1,2,3}; int d[2][2]={{1},{2,3}};**

Массивы указателей (**int *a[5]**) и указатель на массив (**int (*a)[5]**)

Варианты хранения строк в памяти: массив символов (**char s[10]**) и последовательность символов (**char *s**). Обязательный последний символ строки. Необходимость инициализации значения строки. Двойные и одинарные кавычки.

Использование библиотечных функций для работы с символьными данными.

Лекция 14. Сложные типы данных

Структура как тип и совокупность данных. Обращение к элементам структуры. Указатель на структуру, особенности его записи (**имя->поле** или **(*имя).поле**).

Объединения разнотипных данных. Расположение полей в памяти, объем объединения.

Битовые поля структур и объединений.

Описание файла: **FILE *f;**

Открытие файла: **имя=fopen(имя_на_диске, режим)**, где режим - "r" (чтение) или "w" (запись). Например: **f=fopen("lab8.dat","w");**

Значение файловой переменной в случае ошибки открытия файла.

Режим работы с файлами (переменная **_fmode**: **O_TEXT** или **O_BINARY**).

Чтение и запись данных в двоичный файл:

fread (адрес_переменной, кол-во_байт, сколько_раз, имя_файла);

Например: **fread(&k, sizeof(k),1,f);**

fwrite (адрес_переменной, кол-во_байт, сколько_раз, имя_файла);

Например: **fwrite(&k, sizeof(k),1,f);**

Чтение и запись данных в текстовый файл:

fscanf (имя_файла, формат, список_адресов_переменных);

fprintf (имя_файла, формат, список_переменных);

Закрытие любого файла: **fclose(имя);**

Потоковый ввод и вывод данных (iostream.h): **cin >> имя; cout << выражение**

Лекция 15. Работа препроцессора и исключительные ситуации

Стадии и команды препроцессорной обработки.

Замены в тексте.

```
#define M 20
#define AAA for (int i=0; i < 10; i++)
```

Включение текстов из файлов.

```
#include <stdio.h>
#include "myprog.c"
```

Условная компиляция.

```
#ifdef M
#ifdef
#elif
#else
#endif
```

Макроподстановки средствами препроцессора.

```
#define max(a,b) a>b?a:b
```

Встроенные (заранее определенные) макроимена: `__LINE__`, `__FILE__`, `__DATE__`, `__TIME__` и другие.

Обработка исключительных ситуаций. Ключевые слова `try`, `catch` и `throw`.

```
try
{
оператор1;
оператор2;
...
throw выражение_генерации_исключения;
}
catch (тип_исключения имя) {операторы}
```

Заголовочный файл `graphics.h`. Особенности работы с графикой в Турбо-Си:

```
int *gd=0,*gm; initgraph(gd,gm,"");
if (graphresult() !=0 ) {printf("Error!");}
```

Особенности работы с динамическими структурами в Турбо-Си:

```
struct record
{stud s; // ранее описанная структура с данными о конкретном сту-
денте
  record *next; // указатель на следующий элемент списка
};
record *last=NULL;
last = new(record);
```

СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

1. Turbo Pascal / С.А. Немнюгин. – СПб.: Питер, 2002. – 496 с.
2. Абрамов С.А., Зима Е.В. Начала информатики. - М.: Наука, 1989.-256 с.
3. Климова Л.М. Практическое программирование. Решение типовых задач. - М.: КУДИЦ-ОБРАЗ, 2000. - 528с.
4. Грызлов В.И., Грызлова Т.П. Турбо-Паскаль 7.0. - 3-е изд., испр. и доп. - М.: ДМК, 2000. - 416с.
5. Подбельский В.В. Язык С++: Учебное пособие. – 5-е изд. – М.: Финансы и статистика, 2001. – 560 с.
6. Навчальний посібник. Практикум для виконання лабораторних робіт з дисципліни «Мови об'єктно-орієнтованого програмування». / Укл.: О.Ф.Тарасов, О.В.Алтухов. – Краматорськ: ДДМА, 2001. – 152 с.
7. Касаткин А.И., Вальвачев А.Н. Профессиональное программирование на языке Си: От Turbo C к Borland C++: Справочное пособие; Под общ. Ред. А.И. Касаткина. – Мн.: Выш.шк., 1992. – 240 с.
8. Мельников А.Ю. Алгоритмизация и программирование: учебное пособие для студентов специальности «Интеллектуальные системы принятия решений» / А.Ю. Мельников. – Издание 2-е, с изменениями. – Краматорск: ДГМА, 2010. – 96 с. – ISBN 978-966-379-437-2.
9. Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на С++, 2-е изд. / Пер. с англ. – М.: «Издательство Бинном», СПб.: «Невский диалект», 2001. – 560 с.
10. Подбельский В.В. Язык С++: Учебное пособие. – 5-е изд. – М.: Финансы и статистика, 2001. – 560 с.
11. Алексеев В.Е, Ваулин А.С., Петрова Г.Б. Вычислительная техника и программирование: Практикум по программированию, - М.: Высшая школа, 1991,- 324 с.
12. Вальвачев А.И., Крисевич В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС. - Минск: Выш. шк.,1989. - 223 с.
13. Васюкова Н.Д., Тюляева В.В. Практикум по основам программирования. Язык Паскаль. - М.: Выш. школа, 1991.- 160 с.
14. Мизрохи С.В. TURBO PASCAL и объективно - ориентированное программирование. - М.: Финансы и статистика, 1992.- 192 с.
15. Керниган Б., Ритчи Д. Язык программирования Си. – 2-изд. – М.: Финансы и статистика, 1992. – 272 с.
16. Страуструп Б. Язык программирования С++. – М.: Радио и связь, 1991. – 352 с.
17. Шилдт Г. Язык Си для профессионалов. – М.: ИВК-СОФТ, 1992. – 319 с.
18. Касаткин А.И. Профессиональное программирование на языке Си. Управление ресурсами: Справочное пособие. – Мн.: Выш.шк., 1992. – 432 с.
19. Касаткин А.И., Вальвачев А.Н. Профессиональное программирование на языке Си: От Turbo C к Borland C++: Справочное пособие; Под общ. Ред. А.И. Касаткина. – Мн.: Выш.шк., 1992. – 240 с.