Содержание

Занятие № 1. Алгоритм. Язык и среда программирования.

Рассмотрим различные способы управления любыми устройствами. Всего их существует два: ***ручной*** и ***автоматический***.

Ручной способ подразумевает постоянное выполнение следующих действий:

1. Сбор информации.
2. Анализ.
3. Принятие решения.
4. Выполнение действия над устройством (согласно принятому решению).

Этот способ получил широкое, однако, у этого способа есть серьёзный недостаток. Предположим, что требуется управлять некоторым устройством посредством систем телеметрии дистанционно. Например, запустим космический корабль с вездеходом на Марс и будем управлять им с Земли. Возникает нештатная ситуация – на движущийся по поверхности Марса вездеход катится камень. Оператор на Земле принимает решение свернуть, однако, на момент принятия решения вездехода уже попросту не существует, так как сигнал с Земли достигает Марса со значительной временной задержкой. В сложившейся ситуации ручной способ управления устройством не подходит. На помощь приходит другой способ управления – автоматический. Этот способ управления можно применить во многих отраслях деятельности человека, например, если требуется управление быстротекущими процессами или ситуации когда присутствие человека, управляющего устройством, вовсе невозможно. В таких случаях человек за ранее продумывает действия устройства в различных ситуациях. Некоторым образом записывает эти команды в само устройство. И далее устройство действует без вмешательства человека. Последовательность действий, которой подчиняется устройство, в данном случае может быть названо алгоритмом.

Понятие алгоритма была сформулировано еще арабским математиком Абу Аддула Мухаммед бен Муса аль Маджуса аль Хорезми, и получило широкое распространение не только в управлении устройствами но и при выполнении различных действий человеком (например, в школе некоторые задачи решаются по алгоритму).

Алгоритм – это точная и краткая последовательность действий, направленных на достижение поставленной цели, либо решение сформулированной задачи.

Однако следует понимать, что не любая последовательность действий является алгоритмом. Построение алгоритма должно подчиняться правилам, а сам алгоритм удовлетворять некоторым свойствам.

Свойства алгоритма:

1. Однозначность – подразумевает единственность толкования правил выполнения действия и порядка их выполнения.
2. Конечность – обязательность завершения каждого действия алгоритма и самого алгоритма в целом (в противном случае мы получаем бесконечный алгоритм).
3. Результативность – в завершение работы алгоритма должны быть получены результаты.
4. Массовость – возможность использования алгоритма для решения целого класса аналогичных задач.
5. Правильность – алгоритм не должен содержать ошибок (правильный алгоритм даёт правильные результаты).

К сожалению (а может быть и к счастью), компьютер не обладает интеллектом, поэтому алгоритм записанный естественным способом, т.е. с помощью естественного

языка, не может быть выполнен и однозначно трактован. Таким образом, требуется переписать алгоритм посредством жёстких правил для компьютера, т.е. формализовать его. ***Формальная запись алгоритма получила название программы***. А ***процесс созданя алгоритма и его формализация – программирование. Совокупность правил, по которым осуществляется формализация алгоритма – это язык программирования***. Можно разработать множество различных правил формализации, что привело к возникновению множеств различных языков программирования. На данный момент их существует огромное количество: Assembler, BASIC, C, Pascal, Perl, Prolog, Java и т.д.

Для того чтобы выполнить записанный на одном из языков программирования алгоритм, в свою очередь, требуется специальная программа, выполняющая формализованный алгоритм. ***Совокупность языка программирования и программы, выполняющей алгоритм, записанный на языке программирования, получили название среды программирования***. Их тоже достаточное количество: Visual BASIC, Turbo BASIC, C++, Visual C, Delphi, Turbo Pascal и т.д.

Выделяют ***три уровня языков программирования: низкий*** (например, Assembler)***, высокий*** (например, Pascal) и ***сверхвысокий*** (например, Visual C, Delphi). Чем ниже уровень языка, тем ближе система команд языка к системе команд центрального процессора и, соответственно, объёмнее программа.

Существует ***два способа преобразования программы в машинный код понятный центральному процессору - трансляция: интерпретация*** и ***компиляция***.

Интерпретация подразумевает поэтапное преобразование в код, понятный центральному процессору (двоичный код) и выполнение команд алгоритма. Причём такие действия повторяются при каждом выполнении алгоритма.

Компиляция подразумевает перевод всей программы в код процессора и сохранение полученного кода в памяти. В дальнейшем полученный код можно использовать многократно.

В нашем случае мы поведем разговор о построение алгоритмов для формальных исполнителей, в качестве которого выберем компьютер. В любом случае алгоритм строится с целью решения какой-либо задачи. Приведём ***этапы решения задачи с использованием ЭВМ:***

1. Постановка задачи.
2. Математическое или информационное моделирование.
3. Алгоритмизация задачи.
4. Программирование.
5. Ввод программы и исходных данных.
6. Тестирование и отладка программы.
7. Исполнение отлаженной программы и анализ результатов, полученных с помощью нее.

Контрольные вопросы:

1. Какие существуют способы управления устройствами?
2. Что такое алгоритм?
3. Каковы свойства алгоритма?
4. Что такое язык программирования?
5. В чём отличие языка программирования от среды программирования?
6. Что такое интерпретация и компиляция? В чём отличие?
7. Какие существуют уровни языков программирования?

Задание:

Составьте алгоритмы:

1. Перехода улицы.
2. Приготовления кофе.

Проверьте, соответствуют ли ваши алгоритмы свойствам алгоритма.

Занятие № 2. Язык программирования Pascal.

Мы приступаем к рассмотрению языка программирования Pascal и среды программирования Turbo Pascal (TP).

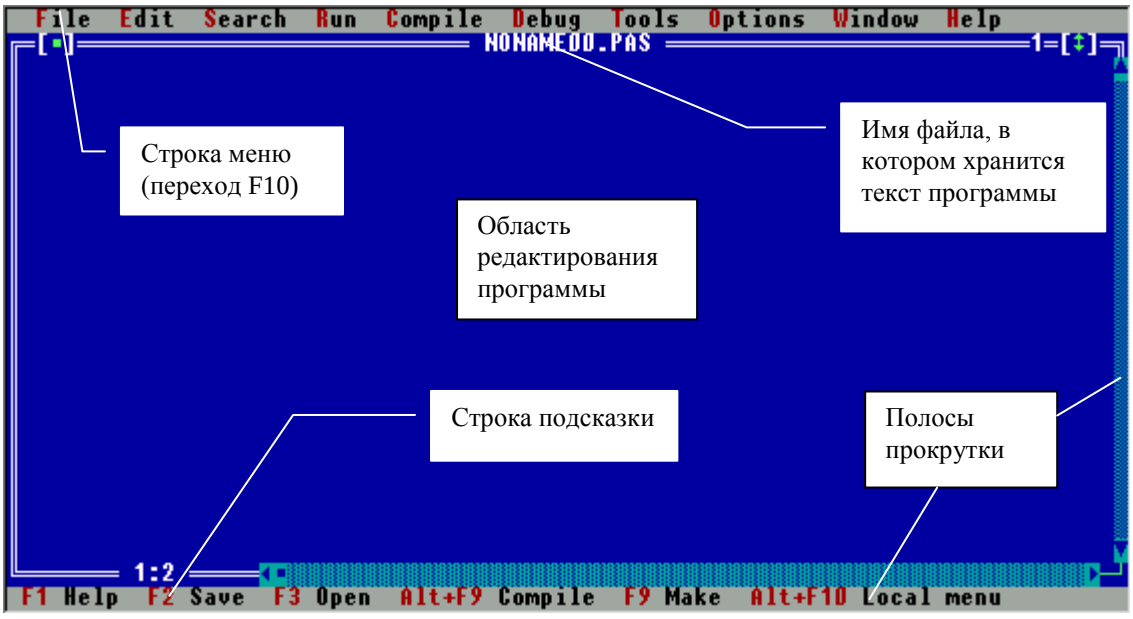
Язык назван в честь французского философа и математика Блеза Паскаля (1623 – 1662), разработан в 1968 – 1971 под руководством Никлауса Вирта – директор института информатики Швейцарской высшей политехнической школы, профессор – на основе языка Algol – 60.

Основными достоинствами Pascal являются компактность, отображение фундаментальных концепций алгоритма, четкое структурное программирование и представление данных, системное программирование, простые, гибкие, компактные структуры.

Файлы среды программирования TP.

turbo.exe	основной файл среды.
turbo.hlp	файл встроенной помощи.
turbo.tp	файл конфигурации среды.
turbo.tpl	библиотека стандартных модулей.
*.bgi	файлы, содержащие драйверы графических адаптеров.
*.tpu	библиотеки алгоритмов в скомпилированном виде.
*.chr	файлы графических шрифтов.
*.pas	файлы текстов программ.

Интерфейс среды программирования TP 7.xx представлен следующими элементами:



Среда программирования многооконная. Поддерживается работа до 9 окон одновременно.

При работе со средой программирования следует уделить особое внимание некоторым «горячим» клавишам:

Клавиша	Меню	Назначение
F1	Help	Экран помощи
Ctrl + F1		Помощь по активному служебному слову
F2	File/Save	Сохранить файл в активном окне
F3	File/Open	Диалоговое окно открытия файла
F4	Run/Go to Cursor	Запускает программу до строки, на которой стоит курсор
F5	Window/Zoom	Масштабирует активное окно

F6	Window/Next •	Переходит к следующему открытому окну
F9	Compile/Make	Запускает Make текущего окна
F10	(none)	Возвращает вас в полосу меню
Редактирование		
Shift+стрелки	(none)	Помечает фрагмент текста в активном окне редактирования
Ctrl+Del	Edit/Clear	Удаляет выбранный текст из окна и не помещает его в карман
Ctrl+Ins	Edit/Copy	Копирует выбранный текст в карман
Shift+Del	Edit/Cut	Помещает выбранный текст в карман и удаляет его
Shift+Ins	Edit/Paste	Помещает текст из кармана в активное окно
Alt+1…9	Window/…	Переключение между окнами по номеру
Alt+0	Window/List	Выводит список всех открытых окон
Компиляция и отладка		
Alt+F9	Compile/Compile	Компилирует последний файл в окне редактора
Ctrl+F2	Run/Program Reset	Переустанавливает выполняемую программу
Ctrl+F4	Debug/Evaluate/Modify	Вычисляет выражение
Ctrl+F7	Debug/Add Watch	Добавляет выражение для просмотра
Ctrl+F9	Run/Run	Компилирует и выполняет программу
Alt+F5		Просмотре результатов работы программы

Любой язык обладает алфавитом, синтаксисом и семантикой. В алфавит входит набор символов, использующихся в языке с помощью которых, в дальнейшем, формируются слова и предложения. Синтаксис представляет собой способы соединения слов в словосочетания и предложения, а также соединения предложения в сложные предложения. Семантика позволяет осуществлять трактовку (понимание) слов и фраз, записанных на языке.

В алфавит языка входят следующие символы:

Латинские буквы	от A до Z и от a до z
Цифры	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Шестнадцатеричные цифры	от 0 до 9, A, B, C, D, E, F
Специальные символы	+ - * / = < > { } [ ] ( ) . , ; : $ @ # ^ := присваивание >= больше, либо равно <= меньше, либо равно <> неравно
Пробел	
Служебные слова (расцениваются средой как один символ)	absolute, and, array, begin, case, const, div, do, downto, else, end, external, file, for, forward, function, goto, if, implementation, in, inline, interface, interrupt, label, mod, nil, not, of, or, packed, procedure, program, record, repeat, set, shi, shr, string, type, then, to, var, while, with, unit, until, uses, xor

Все что пишется в программе между символами {...} является комментарием и полностью исключается из программы при её компиляции.

Программа, написанная на языке Pascal, имеет строгую структуру и состоит из двух частей: описательной (предварительное описание всего, что будет в дальнейшем использоваться) и собственно самой программы.

Чтобы в дальнейшем было легче описывать структуру операторов, введем умолчание: всё, что пишется между символами **<…>** в реальной записи подменяется по правилам указанным внутри. Например, если написано **<столица государства>**, то в реальных условиях следует писать, например, **Москва** (уже без скобок).

Структура программы имеет следующий вид:

PROGRAM <имя программы>; {заголовок программы}

```
 uses <список модулей>; {подключаемые библиотеки}
 label <список меток для переходов>; {объявление меток}
 const <список констант с указанием значений>; {описание констант}
 type <список типов с описанием>; {описание типов пользователя}
 var <список переменных с указанием типа>; {описание переменных}
 procedure …; {описание процедур}
 function … ; {описание функций}
BEGIN
 <текст программы>
END.
```

Вся программа записывается фразами. В конце фразы обязательно ставится знак ; (точка с запятой), который символизирует конец фразы и переход к следующей. Вся программа заканчивается знаком . (точка) или терминатор. Все что написано после терминатора никак не рассматривается.

Контрольные вопросы:

1. Какова история создания языка и среды программирования Pascal?
2. Из каких фалов состоит система программирования TP? Каково назначение каждого из них?
3. Какими элементами представлен интерфейс среды TP?
4. Какова структура программы языка Pascal?
5. Из каких элементов состоит программа на Pascal?
6. Каким знаком отделяются друг от друга фразы в программе?
7. Что такое терминатор и для чего он предназначен?

Задание:

Поработать со средой программирования Turbo Pascal:

1. Запустить среду программирования Turbo Pascal.
2. Ввести следующую программу

```
Program Summ;
 uses crt;
 var a,b,c:integer;
BEGIN
 clrscr;
 write('Введите первое целое число: '); readln(a);
 write('Введите второе целое число: '); readln(b);
 c:=a+b;
 writeln('Результат=',c);
 readkey;
END.
```

3. Выполнить введённую программу.
4. Установить, что делает программа.
5. Выделите фразы в программе. Выделите описательную часть и собственно саму программу. Попытайтесь догадаться о назначении некоторых (а может всех) операторов?
6. Попытайтесь самостоятельно расширить возможности программы.

Занятие № 3. Переменные. Простейшие линейные алгоритмы.

Переменная – это именованная область оперативной памяти ЭВМ, хранящая какое-либо значение. Переменная как контейнер, в который можно положить информацию, а затем изъять оттуда.

Переменная обладает тремя атрибутами: именем, типом, значением.

Имя переменной предназначено для того, чтобы отличать одну переменную от другой и строится по следующим правилам:

1. Имя переменной начинается всегда с буквы латинского алфавита.
2. Имя переменной может содержать также цифры 0…9 и некоторые символы, например **_**.
3. Имя переменной не должно быть очень длинным.
4. Имя переменной должно (желательно) отображать содержимое переменной (например, если в переменной вы храните значение ускорения свободного падения, то логично её назвать **g**).

Тип переменной показывает какого вида данные могут находиться в переменной и обозначается специальными зарезервированными словами. Выделяют также группы типов:

Служебное обозначение	Длинна, byte	Диапазон значений (точность, знаков)	Наличие знака
Целочисленные			
ShortInt	1	-128..127	+
Integer	2	-32768..32767	+
LongInt	4	-2147483648..2147483647	+
Byte	1	0..255	-
Word	2	0..65535	-
Вещественные			
Real	6	$2.9^{-39}..1.7^{38}$ (11-12)	+
Single	4	$1.5^{-45}..3.4^{38}$ (7-8)	+
Double	8	$5.0^{-324}..1.7^{308}$ (15-16)	+
Extended	10	$3.4^{-4932}..1.1^{4932}$ (19-20)	+
Comp	8	$-9.2^{18}..9.2^{18}$ (19-20)	+
Логический			
Boolean	1	True, False	-
Символьный			
Char	1	Символы кода ASCII	-

Для описания совокупностей символов (строк) мы будем использовать тип string – этот тип на самом деле является составным (набор char), однако более подробно об этом мы расскажем позже (см. Занятие № 19).

Существуют также и более сложные структурные типы данных о которых мы расскажем позже.

Значение переменной передаётся с помощью специального знака присваивания в теле программы. Он имеет вид **:=** и пишется без пробелов внутри. Операция присваивания принимает следующий вид и представляет одну фразу:

```
<имя переменной> := <выражение>
```

Причём если происходит присвоение символьной величины, то она заключается в апострофы **`<символьная величина>`**.

Например:

```
g := 9.81;
xmax := 29;
s := 'Слава труду!';
```

```
ch := 'ы';
```

Прежде, чем использовать переменную в программе следует её описать. ***Описать переменную значит указать её имя и определить тип***. Описание переменной производится в разделе ***var*** следующим образом:

```
var <список имён переменных>: <тип>; …;
```

Например:

```
var g: real;
    a, b, c: integer;
    xmax, ymax: double;
    ch: char;
    s: string;
```

Здесь указано, что в дальнейшем в программе можно использовать переменную **g** в которую заносится значение вещественного типа; переменные **a**, **b**, **c**, в которые можно заносить только целочисленные значения и т.д.

В качестве выражения, которое присваивается переменной, может выступать постоянная или арифметическое выражение, в которое также могут входить переменные (при вычислении вместо переменных подставляются их значения).

Арифметические выражения могут состоять из следующих элементов:

1. Постоянные (например, 24, -45, 34.23).
2. Знаки арифметических операций (*, /, +, -, div, mod).
3. Круглые скобки ().
4. Переменные.
5. Функции (о них поговорим попозже).

Приоритеты операций (от высшего к низшему):

1. Вычисление функции.
2. Смена знака (-).
3. Умножение (*), деление (/), div, mod.
4. Сложение (+), вычитание (-).

div – целочисленное деление (например 7 div 3=2)

mod – деление по модулю (целочисленный остаток от целочисленного деления) (например 7 mod 3=1)

Например:

```
X := 23;
Y := 15;
Z := (X-Y)/2;
```

В итоге в переменной **X** будет находиться значение 23, в **Y** будет 15, а в **Z** будет 4. В связи с этим примером следует понимать одну тонкость: в то время как переменные **X** и **Y** могут быть целочисленного типа, переменная **Z** может быть только вещественного, т.к. операция деления всегда порождает вещественный тип.

Мало уметь производить расчеты. Требуется также осуществлять вывод результатов на экран. Это можно сделать с помощью стандартной процедуры вывода WriteLn, которая имеет следующий формат.

```
WriteLn(<список параметров>);
```

В качестве параметров могут выступать текстовые строки (напомним, они заключаются в апострофы) и выражения (при выводе вычисляются), причем если параметров несколько, то они разделяются запятыми. Например:

```
WriteLn('Прыгающий кактус');
```

выведет на экран с позиции курсора словосочетание **`Прыгающий кактус.`** Или

```
WriteLn('2+2=', 2+2);
```

Выведет на экран строчку **`2+2=4.`** Аналогичный результат будет получен при рассмотрении следующего фрагмента программы:

```
A:=2; B:=2; C:=A+B;
WriteLn(A,'+',B,'=',C);
```

Существует возможность также описания констант. Константа, в отличие от переменной, не может изменять своё значение в процессе работы программы. Тип константы определяется автоматически. При описании константы в разделе **const** описательной части программы сразу же указывается её значение. В разделе описания констант можно также описывать переменные с предустановленным значением, в этом случае обязательно указывается тип переменной:

Описание переменной с предустановленным значением	**Описание константы**
const <имя пер-ой.>: <тип> = <значение>;	const <имя константы> = <значение>;

Например,

```
const g = 9.81; {объявление константы}
```

Или

```
const e: real = 1.6E-19; {объявление переменной с предустановленным значением}
```

Контрольные вопросы:

1. Что такое переменная?
2. Какими атрибутами обладает переменная?
3. Какие типы переменных вам известны?
4. По каким правилам следует давать имя переменной?
5. Как производится присвоение значения переменной?
6. Из каких элементов состоят выражения?
7. Каким образом описать переменную с предустановленным значением?
8. Как вывести на экран текст, значение переменной, результат вычисления выражения?

Задание:

Составьте программу вывода на экран результата расчёта следующего выражения:

$$y = (2-a)\cdot\frac{\frac{b}{c}-3a+\frac{c}{2b}}{4}$$, где a=4, b=-2, c=7.25

Занятие № 4. Стандартные операции ввода/вывода.

Важнейшими операциями являются операции ввода и вывода. В частности, мы поговорим о вводе данных с клавиатуры и выводе их на экран.

I. Под выводом подразумевается передача информации из оперативной памяти ЭВМ к внешним устройствам. Для вывода данных на экран используются две стандартные процедуры:

```
Write(<список параметров>);
WriteLn(<список параметров>);
```

Разница между ними заключается в том, что при выводе с помощью процедуры Write курсор остается в той же строке, а при выводе с помощью WriteLn после вывода курсор будет автоматически перемещён в начало следующей строки. Большие возможности предоставляются способностью осуществлять форматирование вывода на экран следующим образом:

```
<выражение>:C:D
```

C – общее количество знаков в числе (строке),

D – количество знаков в десятичной части (не применяется для целочисленных результатов выражения и для строк). Выравнивание при форматирующем выводе всегда ведётся по правой стороне. Например:

```
WriteLn(12.234:7:2);
```

Выведет на экран

```
__12.23
```

В случае выравнивания целочисленных результатов и текстовых значений параметр D упускается.

При выводе на экран вещественных результатов без применения форматирующего вывода используется экспоненциальный способ представления числа (с плавающей точкой). Например, можно увидеть следующий результат:

```
2.53E-02
```

Число представлено тремя частями. Обозначим их:

```
R   E K,
```

где R – мантисса, E – основание, K – степенной показатель. Всё становится ясным, если переписать это число в естественной форме:

$2.53 \cdot 10^{-2}$

II. Под вводом подразумевается передача информации от внешних устройств в оперативную память ЭВМ. Для ввода данных с клавиатуры используются стандартные процедуры:

```
Read(<список переменных>);
ReadLn(<список переменных>);
```

При использовании для ввода процедуры Read курсор остаётся в той же строке, ReadlLn – курсор автоматически переводится на следующую строку.

С помощью одной процедуры возможен ввод значений нескольких переменных. В этом случае значения переменных при вводе разделяются либо знаком пробел, либо нажатием клавиши Enter. Например:

```
ReadLn(a,b);
```

Вводит с клавиатуры значение двух переменных **a** и **b**, первое значение помещается в переменную **a**, а второе в **b**.

Для оформления ввода и вывода может потребоваться ещё две возможности. Это очистка экрана и позиционирование курсора в любом месте экрана. Эти возможности предоставляются процедурами, содержащимися в стандартном модуле CRT. Прежде, чем использовать процедуры и функции этого модуля в описательной части программы в разделе uses требуется его подключить. Делается это следующим образом:

```
uses CRT;
```

После этого можно использовать возможности этого модуля.

Очистка экрана осуществляется процедурой

```
ClrScr;
```

Ещё одна процедура, которая может понадобиться

```
ReadKey;
```

Эта процедура приостанавливает работу программы до нажатия любой клавиши. Эту процедуру очень удобно использовать в конце вашей программы для приостановки её работы (чтобы не приходилось лишний раз нажимать комбинацию клавиш Alt + F5 для просмотра результата работы).

Приведём пример программы нахождения частного двух чисел введённых с клавиатуры:

```
Program Division;
 uses CRT;
 var a, b: integer;
     res: real;
BEGIN
 ClrScr;
 write('Введите первое число: '); readln(a);
 write('Введите второе число: '); readln(b);
 res := a / b;
 writeln(a,' поделить на ',b,' равно ',res);
 readkey;
END.
```

Позиционирование курсора на экране в текстовом режиме производится процедурой

```
GotoXY(x,y);
```

где x ($1 \le x \le 80$) – номер столбца, а y ($1 \le y \le 25$) – номер строки, куда следует переместить курсор.

Приведем ещё один пример программы, использующей и эту процедуру:

```
Program DemoCRT;
 uses crt;
BEGIN
 ClrScr;
 GotoXY(36,12);
 Write('Привет!');
 ReadKey;
END.
```

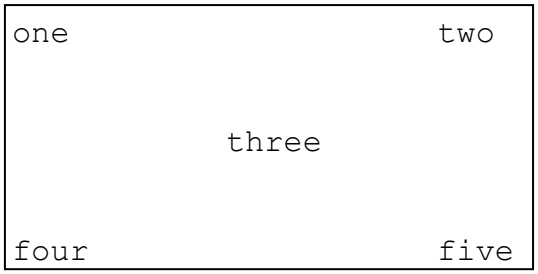
В результате работы программы по центру экрана будет выведено слово **Привет!** и будет осуществлено ожидание нажатия какой-либо клавиши, после чего выполнение программы прекратиться.

Контрольные вопросы:

1. Что подразумевается под вводом и выводом данных?
2. Какие процедуры используются для вывода данных? В чём отличие между ними?
3. Какие процедуры используются для ввода данных? В чём отличие между ними?
4. Как подсоединить к программе внешний модуль?
5. С помощью какой процедуры можно произвести очистку экрана?
6. Для чего используется процедура ReadKey?

Задания:

1. Составить программу, выводящую на экран следующее изображение:



2. Составить программу, запрашивающую ввод с клавиатуры двух чисел и выводящую на экран сумму, произведение, разность и частное этих двух чисел.

3. Напишите программу перевода значения температуры в градусах Цельсия в градусы по Фаренгейту ($^0F=^0C*1.8+32$).

Программа в начале своей работы должна выводить заставку, аналогичную следующей:

```
*************************
*        Программа      *
* вычисления температуры *
*    Автор: Петров В.И.  *
*************************
```

а затем запрашивать ввод температуры в градусах Цельсия.

4. Составьте программу, вычисляющую результат следующего выражения:

$$D = A * \frac{B}{\pi} + C * 3 + \sqrt{123},$$

где π константа, равная 3,1415.

5. Напишите программу вычисления веса идеального мужчины по формуле:

Ид. вес = Рост в см – 100

Значение роста вводится с клавиатуры. Результат вывести в следующем виде (например):

```
Для человека ростом 165 см идеальный вес равен 65 кг.
```

6. По аналогии с задание № 4, составить программу, определяющую идеальный вес женщины по следующей формуле:

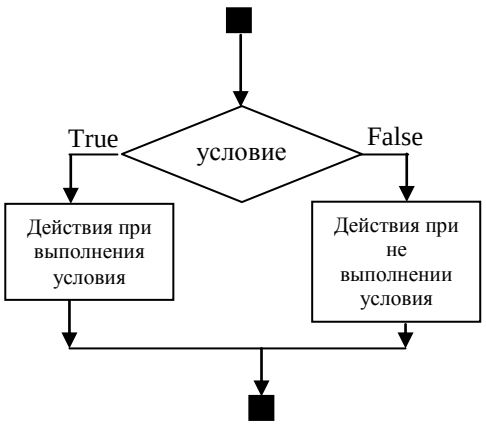
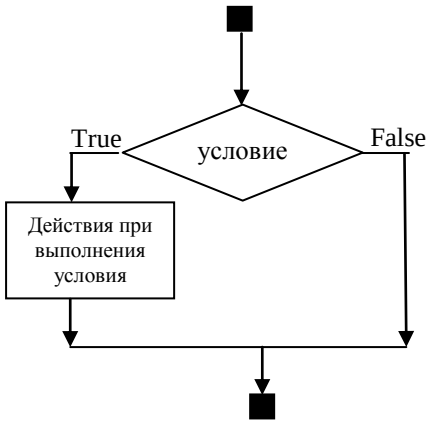
Ид. вес = Рост в см – 100 – 10% (от Рост в см – 100)

Занятие № 5. Ветвление с простым условием.

До сих пор мы разбирали только линейные алгоритмы (действия выполняются друг за другом), однако, очень часто возникают ситуации, когда требуется выполнить одно, либо другое действие в зависимости от выполнения (либо не выполнения) условия.

Например, фраза «Если на улице хорошая погода, то я куплю мороженное, иначе я куплю пирожок» представляет собой ни что иное, как условный выбор. Полный условный выбор состоит из трёх частей: условия (в нашей фразе «хорошая погода), которое может либо выполняться, либо нет; действия, выполняющегося в случае выполнения условия («куплю мороженое»); действия, выполняющегося в случае не выполнения условия («куплю пирожок»).

С условным выбором удобно познакомиться с помощью блок-схемы – графического представления структуры:

Полный условный выбор	Сокращённый условный выбор
	
В языке Pascal существует специальный составной оператор (состоит из нескольких служебных слов) реализующий оба вида условного выбора:	

`if <условие> then`
`  <оператор 1>`
`else`
`  <оператор 2>;`
	`if <условие> then <оператор>;`

Условие представляет собой логическое выражение. Существует два вида условий: простое и составное (разберём позже).

Простое условие – это выражение, устанавливающее однозначное соответствие между двумя его частями:

```
<выражение 1><знак соотношения><выражение 2>
```

Знаки соотношения:

> - больше
< - меньше
= - равно
>= - больше, либо равно
<= - меньше, либо равно
<> - неравно.

В случае если требуется выполнить несколько операторов в условном выборе, то они обрамляются операторными скобками:

```
begin
 <совокупность операторов>
```

```
end;
```

Если операторная скобка закрывается (end) перед ветвью иначе (else) точка с запятой не ставится, т.к. фраза ещё не закончена.

Так же ничто не мешает создавать конструкции с большим количеством ветвлений помещая оператор условного выбора в условный выбор.

Пример. Составить программу, определения количества корней квадратного уравнения:

```
Program SquareRoot;
 uses crt;
 var a,b,c,d:real;
BEGIN
 ClrScr;
 Write('Введите коэффициент a:'); ReadLn(a);
 Write('Введите коэффициент b:'); ReadLn(b);
 Write('Введите коэффициент c:'); ReadLn(c);
 D:=b*b-4*a*c;
 If d<0 then WriteLn('Корней нет.');
 If d=0 then WriteLn('Один корень.');
 If d>0 then WriteLn('Два корня.');
 ReadKey;
END.
```

Контрольные вопросы:

1. Что такое условный выбор?
2. Из каких частей состоит условный выбор?
3. Какие бывают условия?
4. Что такое простое условие?
5. Какие знаки отношений вы знаете?
6. Что делать, если в одной из ветвей условного выбора более одного оператора?
7. Что такое операторные скобки?

Задания:

1. Составить программу, определяющую по введенным с клавиатуры координатам, попадет точка в указанную окружность или нет (радиус окружности равен 50).

2. Услуги телефонной компании оплачиваются по следующему правилу: за разговоры до A минут в месяц оплачиваются B руб., а разговоры сверх установленной нормы оплачиваются из расчета C руб. за минуту. Написать программу, вычисляющую плату за пользование телефоном для введенного времени разговора за месяц (тарифы B и C также вводятся).

3. Даны две точки, определить, которая из них находится ближе к началу системы координат.

4. По известным коэффициентам A, B и C квадратного уравнения определить его корни.

Занятие № 6. Составные условия.

Возникают ситуации, когда полное условие состоит из нескольких меньших условий. Такая ситуация приводит к появлению составного условия. ***Составное условие – это совокупность простых условий объединённых логическими функциями***. Логические функции бывают двух типов: одной и двух переменных. Всего существует 4 логические функции одной переменной и 16 функции двух переменных. Мы ограничимся рассмотрением всего четырёх.

В случае использования составных условий простые условия заключаются в круглые скобки.

Для знакомства с логическим функциями введём условные обозначения и понятия:

T – True (истина, условие выполняется)

F – False (ложь, условие не выполняется)

Таблица истинности – это перебор всевозможных исходов аргументов логической функции (выражения) с анализам результата всей логической функции.

A	B	**Инверсия (НЕ)**	**Конъюнкция (И)**	**Дизъюнкция (ИЛИ)**	**Исключающее ИЛИ**
		Обозначение в Pascal			
		NOT (A)	**(A) AND (B)**	**(A) OR (B)**	**(A) XOR (B)**
F	F	T	F	F	F
F	T		F	T	T
T	F	F	F	T	T
T	T		T	T	F

Как видно из таблицы инверсия – функция одной переменной, все остальные функции – функции двух переменных.

Теперь можно составлять более сложные условия. Продемонстрируем это на примере.

Пример. Даны три положительных числа a, b, c. Проверить, могут ли они быть длинами сторон треугольника. Если да, то вычислить площадь этого треугольника, в противном случае вывести сообщения о невозможности построения такого треугольника.

Условие существования треугольника – выполнение следующей системы неравенств:

$$\begin{cases} a+b \geq c \\ b+c \geq a \\ a+c \geq b \end{cases}$$

Приблизительную площадь треугольника можно рассчитать по формуле Герона:

$S = \sqrt{p(p-a)(p-b)(p-c)}$, где p – полупериметр.

Программа решения:

```
program triangle;
 uses crt;
 var a,b,c,p,s:real;
BEGIN
 ClrScr;
 write('Введите сторону A: '); readln(a);
 write('Введите сторону B: '); readln(b);
 write('Введите сторону C: '); readln(c);
 if (a+b>=c) and (a+c>=b) and (b+c>=a) then
   begin
```

```
    p:=0.5*(a+b+c);
    s:=sqrt(p*(p-a)*(p-b)*(p-c));
    writeln('Площадь треугольника равна ',s:3:2);
   end
  else
   writeln('Треугольник не существует.');
 readkey;
END.
```

Единственную сложность может представлять строка вычисления площади:

```
s:=sqrt(p*(p-a)*(p-b)*(p-c));
```

sqrt(x) – это функция возвращающая корень квадратный своего аргумента заключённого в скобки.

Контрольные вопросы:

1. Что такое составное условие?
2. Какие функции одной переменной вы знаете? Как они работают?
3. Какие функции двух переменной вы знаете? Как они работают?

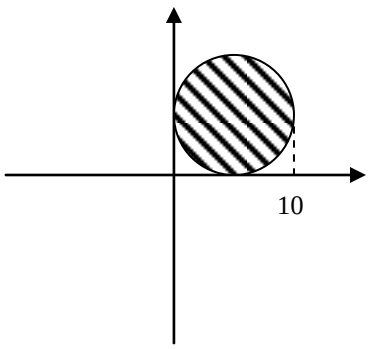
Задания:

1. Составить программу, определяющую количество дней во введенном с клавиатуры месяце года.

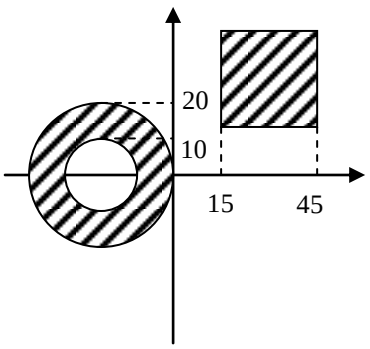
2. Составить программу, решения квадратного уравнения по введенным с клавиатуры коэффициентам **a**, **b** и **c**.

3. Определить принадлежит ли точка с координатами (*x, y)* указанной области:

а)



б)



Занятие № 7. Множественное ветвление. Безусловный переход.

Существенным ограничением условного выбора заключается в том, что максимально возможно выполнить только два различных действия (одно при выполнении условия, другое при не выполнении). Однако возникают ситуации, когда требуется выполнить несколько различных ***исключающих друг друга*** действий. В этом случае можно воспользоваться структурой вложенного условного выбора, например, следующего вида:

```
if <условие> then
 if <условие> then
  if <условие> then
   ...
```

или

```
if <условие> then
  <оператор 1>
 else
  if <условие> then
    <оператор 2>
   ...
```

В случае присутствия в условиях вещественного типа ничего не остаётся, как пользоваться подобными структурами. Если же в условии используется только перечислимые типы (целочисленные), то появляется возможность использования конструкции множественного ветвления:

```
case <выражение> of
  <значение 1>: <оператор 1>;
  <значение 2>: <оператор 2>;
  ...
end
```

или

```
case <выражение> of
  <значение 1>: <оператор 1>;
  <значение 2>: <оператор 2>;
 else
  <оператор>
end
```

В данной структуре выполняется тот оператор, значение перед которым совпадает с выражением, находящимся после служебного слова case. Во втором случае существует ветвь выполнения действия, в случае если `выражение` не совпало не с одним из `значений`.

При указании значения, которому должно соответствовать выражение возможно использование списков, в этом случае значения перечисляются через запятую.

Пример: Составить программу определения количества дней в указанном месяце указанного года (Високосным является год номер которого делится нацело на 4 – это правило не относится к годам, являющимся окончанием столетия (например 1800). Последний год любого столетия считается високосным только в том случае, если его номер делится на 400).

```
Program DayInMonth;
 uses crt;
 var y,m,d:integer;
BEGIN
```

```
 ClrScr;
 write('Введите номер месяца и года (через пробел): ');
 readln(m,y);
 case m of
  1,3,5,7,8,10,12: d:=31;
  4,6,9,11: d:=30;
  else
   if y mod 4<>0 then d:=28
    else
     if (y mod 100=0) and (y mod 400<>0) then d:=28
      else d:=29;
 end;
 writeln('Количество дней ',d);
 readkey;
END.
```

Безусловный переход применяется в том случае, если требуется осуществить скачкообразный переход в любое место программы и продолжить выполнение программы именно с этого места. Следует отметить, что *безусловный переход является анахронизмом в программировании, поэтому от его использования следует воздержаться, либо свести к минимуму*. Данную структуры мы приводим только для общности рассмотрения языка.

Для осуществления безусловного перехода следует завести текстовую метку, которую требуется описать в разделе label описательной части программы. Имена меткам даются по правилам именования переменных. Метка, на которую осуществляется переход, пишется в начале строки и заканчивается двоеточием:

```
<метка>:
```

Для перехода на метку используется следующий оператор:

```
goto <метка>
```

Пример: Составить программу, запрашивающую возраст пользователя. Считать, что возраст пользователя находится в интервале от 4 до 130 лет. В случае неправильного ввода повторить запрос заново.

```
Program Age;
 uses crt;
 label l1;
 var a:byte;
BEGIN
l1: ClrScr;
 write('Введите свой возраст: ');
 readln(a);
 if (a<4) or (a>130) then
  begin
   writeln('Ошибка ввода.');
   writeln('Нажмите любую клавишу и повторите ввод.');
   readkey;
   goto l1;
  end;
 writeln('Ввод принят. Ваш возраст ',a,' лет (год(а))');
 readkey;
END.
```

? Контрольные вопросы:

1. Что такое множественное ветвление?
2. Какие виды множественного ветвления существуют?
3. Что такое безусловный переход?
4. Как описывается метка для безусловного перехода?

Задания:

1. Составить программу, требующую ответ на вопрос «Умеете ли вы плавать» - «да» или «нет». Вопрос повторять до тех пор, пока не будет введён требуемый ответ.

Занятие № 8. Самостоятельная работа.

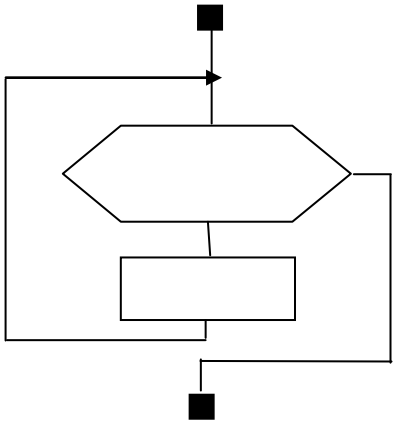
Занятие № 9. Цикл со встроенным счётчиком (с параметром).

Ещё одна часто возникающая ситуация в программировании – это повторение однотипных действий несколько раз. Для реализации этой операции присутствует специальная структура – цикл. Существует два вида циклов: цикл со встроенным счётчиком (с параметром, по параметру) и циклы с условием (см. Занятие № 11). Рассмотрим на данном занятии более подробно цикл со встроенным счётчиком для этого введем рад понятий.

Циклическая операция (тело цикла) – действия, которые выполняются несколько раз в цикле с точностью до параметров.

Критерием применимости цикла со встроенным счётчиком является заранее известное или вычислимое количество циклических операций. Для того, чтобы производить учёт циклических операций в цикле содержится счётчик – это специальная переменная, изменяющая свое значение от начального до конечного с шагом 1 или -1.

Обозначение цикла со встроенным счетчиком на блок-схеме



Структура в Pascal

```
for <переменная-счётчик>:=<нач. значение> to <кон. значение>
 do
  <оператор>
```

или

```
for <переменная-счётчик>:=<нач. значение> downto <кон. значение>
 do
  <оператор>
```

В качестве счётчика может быть использована только переменная перечислимого типа.

В первом случае при каждом совершении циклической операции (<оператор>) значение счётчика увеличивается на 1, возрастая от начального значения. Как только счётчик достигнет конечного значения выполнение циклической операции прекратиться и будет осуществлён выход из структуры цикла.

Во втором случае при каждом выполнении циклической операции значение счётчика убывает на 1 от начального значения до конечного.

В случае нормального выхода из цикла значение переменной-счётчика соответствует конечному.

Существует возможность принудительного выхода из тела цикла без достижения счётчиком конечного значения. Для этого используется процедура

```
break
```

Очень часто значение переменной-счётчика используется в теле цикла.

Пример 1. Составить программу, печатающую на экране 13 раз слово «Пульверизатор»

```
Program Age;
 uses crt;
 var i:byte;
BEGIN
 ClrScr;
 for i:=1 to 13 do writeln('Пульверизатор');
 readkey;
END.
```

Пример 2. Составить программу, выводящую на экран таблицу умножения на 7. (сейчас и далее мы будем приводить только фрагменты программ).

```
for i:=1 to 10 do
 writeln(i,'*7=',i*7);
```

Если внутри тела цикла находится другой цикл, то второй цикл (внутренний) называется вложенным.

Пример 3. Вывести на экран полную таблицу умножения (таблицу Пифагора).

```
for i:=1 to 10 do
 for j:=1 to 10 do
  begin
   gotoxy(i,(j-1)*3+1); {установка курсора в нужное место}
   writeln(i*j:3); {вывод на экран форматированного произведения}
  end;
```

Пример 4. Вывести на экран все простые числа в интервале от 1 до 999

```
Program PrimeNum;
 uses crt;
 var i,j:integer;
     p:boolean;
BEGIN
 ClrScr;
 for i:=1 to 999 do
  begin
   p:=true;
   for j:=2 to i-1 do
    if i mod j=0 then begin p:=false; break; end;
   if p then write(i:4);
  end;
 readkey;
END.
```

Замечание: строка

```
if p then write(i:4);
```

эквивалентна строке

```
if p=true then write(i:4);
```

так как переменная p уже переменная логического типа, коим обладает результат анализа условия.

Контрольные вопросы:

1. Что такое тело цикла или циклическая операция?
2. Для чего используется счётчик?
3. Что используется в цикле в качестве счётчика?
4. Какими способами можно изменять значение счётчика?
5. Что такое вложенный цикл?
6. Существует ли возможность принудительного выхода из цикла?
7. Какова структура цикла со встроенным счётчиком?

Задания:

1. Составить программу, осуществляющую вывод таблицы перевода температур от 0^0C до 100^0C с шагом 10^0C в градусы по Фаренгейту. Таблица должна иметь примерно следующий вид:

```
градусы Цельсия      градусы Фаренгейта
----------------------------------------
       0                    32
      10                    50
      20                    68
      30                    86
```

и т.д.

Формула для расчета: $F(^0\text{по Фаренгейту}) = G(^0\text{по Цельсию}) \cdot 1.8 + 32$

2. Составить программу вывода на экран следующего изображения:

```
*
**
***
****
*****
******
*******
********
*********
```

3. Составить программу расчета среднего арифметического **n** чисел. Число **n**, и сами **n**-чисел вводятся с клавиатуры в процессе работы программы.

Занятие № 10. Стандартные функции. Преобразование типов. Операции над строковым типом данных.

Приведём ряд стандартных функций Pascal, которые нам могут пригодиться:

Функция	Тип аргумента	Тип результата	Описание
pi	-	R	Число π
abs(x)	I, R	I, R	Модуль
int(x)	I, R	R	Целая часть аргумента
frac(x)	I, R	R	Дробная часть аргумента
round(x)	I, R	I	Округление до ближайшего целого
trunc(x)	R	I	Цела часть аргумента
random	-	R	Псевдослучайное число в интервале (0, 1)
random(x)	I	I	Псевдослучайное число в интервале (0, x)
sqr(x)	I, R	I, R	Квадрат x
sqrt(x)	I, R	R	Корень квадратный из x
exp(x)	I, R	R	Экспонента (e^x)
ln(x)	I, R	R	Натуральный логарифм
sin(x)	I, R	R	Синус x (x в радианах)
cos(x)	I, R	R	Косинус x (x в радианах)
arctan(x)	I, R	R	Арктангенс x (результат в радианах)

Возведение в степень в Pascal производится следующим образом:

$x^k \equiv$ `exp(k*ln(x))`

Функции по работе с целочисленным и перечислимым типами:

Функция	Тип аргумента	Тип результата	Описание
ord(x)	порядковый	I	Порядковый номер значения x в его типе
pred(x)	порядковый	то же, что x	Предыдущее x значение в его типе
succ(x)	порядковый	то же, что x	Последующее x значение в его типе
chr(x)	I	char	Символ с порядковым номером x из таблицы ASCII
odd(x)	I	boolean	x – чётное – false x – нечётное – true

Процедуры по работе с целочисленным типом

Процедура	Описание
inc(x)	Увеличивает значение x на 1
inc(x,k)	Увеличивает значение x на k
dec(x)	Уменьшает значение x на 1
dec(x,k)	Уменьшает значение x на k

Функции и процедуры по работе со строковым типом:

Название	Вид	Описание
copy(s,poz,n)	функция	Выделяет из строки s подстроку длинной n символов начиная с символа номер poz
concat(s1,s2,…)	функция	Осуществляет объединение (конкатенацию) строк

		s1, s2 и т.д.
length(s)	функция	Возвращает число (byte), являющееся длиной строки s
pos(s1,s2)	функция	Результат – целое число, являющееся номером позиции первого символа подстроки s1 в строке s2
delete(s,poz,n)	процедура	Удаление n символов из строки s начиная с символа номер poz
insert(s1,s2,poz)	процедура	Вставка строки s1 в строку s2 начиная с символа номер poz

Процедуры преобразования типов:

Процедура	Описание
str(a,s)	Преобразует число, находящееся в переменной **a** (любого числового типа) в строку (переменная **s** строкового типа)
val(s,a,code)	Преобразует число, находящееся в переменной строкового типа **s** в число, которое помещается в переменную числового типа **a**. При этом в переменную **code** (типа integer) передается код ошибки преобразования: 0 –преобразование типа произошло успешно, иное число – местоположение ошибки (символ отличается от цифрового, точки и минуса) в строке.

Пример 1. Получить из строки «Симпозиум» строку «Диапозитив».

```
Program Transformation;
 uses crt;
 var s,t:string;
BEGIN
 ClrScr;
 s:='Симпозиум';
 delete(s,1,3);
 delete(s,5,2);
 t:="Диа"+s+"тив";
 writeln(t);
 readkey;
END.
```

Пример 2. Составить программу, определяющую является ли слово, введенное с клавиатуры, палиндромом. Палиндром – это слово, которое читается одинаково и слева направо и справа налево (например, наган и дед).

```
uses crt;
 var s,sr:string;
     i:byte;
BEGIN
 ClrScr;
 write('Введите слово: '); readln(s);
 sr:='';
 for i:=length(s) downto 1 do sr:=sr+copy(s,i,1);
 if sr=s then writeln('Палиндром.')
         else writeln('Не палиндром');
 readkey;
END.
```

Контрольные вопросы:

1. Перечислите стандартные функции Pascal. Каково их назначение?
2. Перечислите процедуры и функции по работе со строковым типом данных. Каково их назначение?
3. Для чего используются функции преобразования типов?

Задания:

1. Определить какое количество гласных букв встречается во введенном с клавиатуры слове.

2. Составить программу, определяющую является ли введённая с клавиатуры фраза, фразой-палиндромом (читается одинаково с обеих сторон, например, «А РОЗА УПАЛА НА ЛАПУ АЗОРА»).

Примечание: для этого требуется сначала удалить из фразы все пробелы, а затем перевернув её сравнить с оригиналом.

3. Определить является ли введенный с клавиатуры номер автобусного билета (шестизначный) счастливым (сумма трех первых цифр совпадает с суммой трех последних цифр).

4. Составьте программу определения кода нажатой клавиши.

5. Выясните, какая из букв первая или последняя, введенного с клавиатуры слова, встречается чаще. Выведите информацию о количестве данных букв в слове.

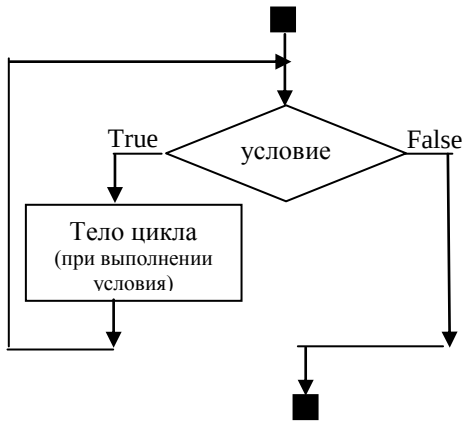
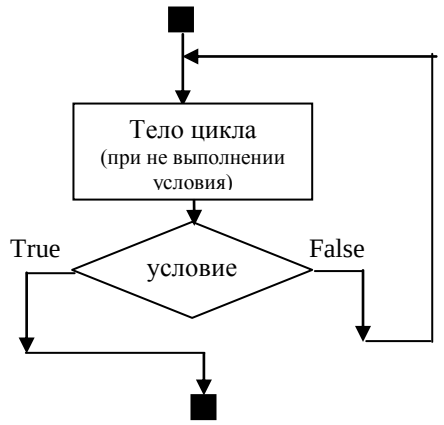
6. Составить программу, запрашивающую с клавиатуры фамилию, имя и отчество, а выводящую на экран фамилию и инициалы.

Занятие № 11. Циклы с условием

Рассмотрим ещё два вида циклов: цикл с предусловием и цикл с послеусловием.

У цикла со встроенным счётчиком есть один очень существенный недостаток – что если количество раз выполнения тела цикла мы не знаем и рассчитать ни коим образом не можем. В этом случае *следует найти условие, при котором надо завершить циклическое выполнение тела цикла (это и есть критерий применимости данного вида циклов)*. Это и позволяют сделать две вышеобозначенные циклические структуры.

Теперь для диагностики завершения выполнения циклических операций будем применять выполнение (либо невыполнение) условия (простого, либо составного).

Блок-схема цикла с предусловием	Блок-схема цикла с послеусловием
	
Структура цикла с предусловием в Pascal	**Структура цикла с послеусловием в Pascal**
`while <условие> do` `  <тело цикла>`	repeat <тело цикла> until <условие>
Особенности работы	
1. Цикл выполняется при выполнении условия. 2. Если условие заведомо не выполняется, то тело цикла не выполняется ни одного раза. 3. Если условие выполняется всегда (например, 13=13), то цикл превращается в бесконечный.	1. Циклы выполняется при **невыполнении условия.** 2. **Тело цикла будет выполнено хоть один раз всегда**, даже если условие заведомо выполняется. 3. Если условие не выполняется всегда (например, 1=2), то цикл превращается в бесконечный. 4. Если в качестве тела цикла используется более одного оператора, то они не нуждаются в обрамлении операторными скобками.

Пример 1. Рассчитать количество лет, необходимых для накопления суммы T начиная с суммы N в банке, предоставляющем ежегодный процент по вкладу P% с капитализацией вклада.

Решение: Следует понимать, что ежегодное начисление процентов с капитализацией вклада (то есть проценты за следующий год начисляются на всю сумму, вм5есте с процентами начисленными за предыдущий (ие) год(ы)) надо производить до тех пор, пока

сумма не превысит, либо не сравняется с итоговой суммой T. Тогда программа решения примет следующий вид:

```
program bank;
 uses crt;
 var t,n,p,s:real;
     y:word;
BEGIN
 ClrScr;
 write('Введите начальную сумму: '); readln(n);
 write('Введите желаемую итоговую сумму: '); readln(t);
 write('Введите годовой процент по вкладу: '); readln(p);
 s:=n; y:=0;
 while s<t do
  begin
   s:=s+s*p/100;
   inc(y);
  end;
 write('Для накопления суммы потребуется ',y,' лет (год,
года)');
 readkey;
END.
```

Пример 2. Найди наибольший общий делитель (НОД) двух чисел, пользуясь алгоритмом Евклида. Алгоритм Евклида заключается в следующем: следует делить m на n до тех пор, пока остаток от деления не будет равен 0. Если остаток не равен 0, то следует m присвоить n, а n присвоить остаток от деления. Причем перед началом выполнения алгоритма следует гарантировать, что m не меньше n.

```
program NOD;
 uses crt;
 var m,n,r:integer;
BEGIN
 ClrScr;
 write('Введите первое число: '); readln(n);
 write('Введите второе число: '); readln(m);
 if m<n then begin r:=m; m:=n; n:=r; end;
 repeat
  r:=m mod n;
  m:=n; n:=r;
 until r=0;
 writeln('НОД=',m);
 readkey;
END.
```

Контрольные вопросы:

1. Какие виды циклов с условием вами известны? Каковы их особенность?
2. Зарисуйте блок-схемы циклов с условием.
3. Какова структура циклов с условием в языке Pascal?

Задания:

1. Найти наименьшее общее кратное (НОК) двух чисел **a** и **b**, используя соотношение:

$$НОК(a, b) = \frac{a \cdot b}{НОД(a, b)}$$

2*. Составить программу, позволяющую определить, сколько можно купить быков, коров и телят, платя за быка 10 р., за корову – 5 р., а за теленка – полтинник (0,5 р.), если на 100 р. надо купить 100 голов скота.

Занятие № 12. Самостоятельная работа.

Занятие № 13. Введение в машинную графику в Pascal.

До сих пор мы использовали только текстовый режим экрана. В текстовом режиме экран разбивается на фиксированные позиции – знакоместа, куда и осуществляется вывод символов. В случае ***если требуется отображать точку в произвольном месте экрана нам потребуется графический режим***, о нём и поговорим более подробно в этом разделе. В среде программирования Turbo Pascal для реализации графических возможностей используется стандартная библиотека **Graph**, которую следует не забыть подключить в разделе Uses.

Для идентификации каждой точки вводится прямоугольная Декартовая система координат следующим образом:

Существует несколько стандартов разрешающей способности экрана (количество точек по осям) и возможности максимального отображения цветов. Приведём таблицу некоторых базовых поддерживаемых Pascal стандартов:

Стандарт	**Разрешение** (кол-во точек по горизонтали x кол-во точек по вертикали)	**Количество цветов**
CGA	320 x 200	4
MCGA	640 x 350	2
EGA	640 x 350	16
EGA64	640 x 480	16
EGAMono	640 x 480	2
IBM8514	1024 x 768	256
HercMono	720 x 348	2
ATT400	640 x 400	2
VGA	640 x 480	16
PC3270	720 x 350	2

Теперь разберём более подробно процедуры и функции графического режима:

Инициализация режимов экрана:

Название	**Вид**	**Формат**	**Описание**
InitGraph	процедура	InitGraph(Driver, Mode:Integer; PathToDriver: string);	Driver (0..10 или имя стандарта) Mode (режим, например для VGA 0..2) PathToDriver (путь к файлу драйвера соответствующего режима (*.bgi))
RestoreCRTMode	процедура	RestoreCRTMode	Восстановление текстового режима

CloseGraph	процедура	CloseGraph	Завершение работы в графическом режиме (возврат в текстовый)
DetectGraph	процедура	DetectGraph(GraphDriver, GraphMode: Integer);	Проверка оборудования и режима

Таким образом, фрагмент программы, подразумевающей работу в графическом режиме может принять следующий вид:

```
Program GraphProbe;
 uses graph;
 var gd,gm:integer;
BEGIN
 gd:=VGA; gm:=VGAHi;
 InitGraph(gd,gm,'');
{ ...}
 CloseGraph;
END.
```

или

```
Program GraphProbe;
 uses graph;
 var gd,gm:integer;
BEGIN
 gd:=Detect;
 InitGraph(gd,gm,'');
{ ...}
 CloseGraph;
END.
```

где Detect – функция, автоматически определяющая вид оборудования и соответственно драйвер для него.

В нашем случае мы пользовались VGA графикой, для которой возможен выбор трёх режимов (будем пользоваться в основном вторым режимом):

Название (константа)	**Значение**	**Разрешение**	**Количество поддерживаемых страниц**
VGALo	0	640 x 200	
VGAMed	1	640 x 350	
VGAHi	**2**	**640x 480**	

Вывод изображения осуществляется с помощью различных примитивов (прямоугольник, круг, полигон и т.д.). Для вывода существует два вида цвета: цвет фона и цвет рисунка. Прежде, чем применить ту, либо иную процедуру, следует воспользоваться процедурами установки цвета:

Формат	**Описание**
SetBkColor(color:word)	Установка цвета фона
SetColor(color:word)	Установка цвета рисунка

В случае графического экрана существует понятие графического курсора, относительно которого осуществляется вывод примитивов некоторыми процедурами. Позиционирование графического курсора осуществляется процедурой

```
MoveTo(x,y:integer)
```

и

```
MoveRel(x,y:integer)
```

для относительного сдвига.

Процедуры вывода примитивов на экран:

Формат	Описание
ClearDevice	Очистка экрана в графическом режиме
PutPixel(x,y:integer)	Точка цветом рисунка
Line(x1,y1,x2,y2:integer)	Построение линии цветом рисунка
LineTo(x2,y2:integer)	Построение линии от графического курсора к точке с координатами x2,y2 цветом рисунка
LineRel(dx,dy:integer)	Построение линии по относительным координатам (сдвиг) цветом рисунка
SetLineStyle(LineStyle, Pattern, Thickness: Word);	Установка характеристик линии. LineStyle: 0 – сплошная, 1 – пунктирная, 2 – штрихпунктирная, 3 – штриховая, 4 – заданная пользователем. Thickness: 1 – нормальная, 3 – толстая
Rectangle(x1,y1,x2,y2:integer)	Контур прямоугольника цветом рисунка
Bar(x1,y1,x2,y2:integer)	Закрашенный цветом фона прямоугольник
Bar3D(x1,y1,x2,y2:integer; Depth: Word; Top: Boolean)	Закрашенный параллелепипед глубиной Depth точек.
DrawPoly(NumPoints: Word; var PolyPoints)	Не закрашенный многоугольник
FillPoly(NumPoints: Word; var PolyPoints)	Закрашенный многоугольник
Circle(x,y: integer; Radius: word)	Круг с центром x,y и радиусом Radius цветом рисунка
Arc(x,y: integer; StAngle, EndAngle, Radius; word)	Дуга окружности начиная с угла StAngle до угла EndAngle, заданных в градуса цветом рисунка
Ellipse(x,y: integer; StAngle,EndAngle: word; XRadius, YRadius: word)	Дуга эллипса цветом рисунка.
FillEllipse(x,y: integer; XRadius, YRadius: word)	Закрашенный цветом фона эллипс
PieSlice(x,y: integer; StAngle, EndAngle, Radius: word);	Закрашенный цветом фона сектор круга
Sector(x, y: integer; StAngle, EndAngle, XRadius, YRadius: word);	Закрашенный цветом фона сектор эллипса
FloodFill(x, y: integer; Border: word)	Закраска **замкнутого одним цветом** контура. **Цвет контура Border**
SetFillStyle(Pattern: Word; Color: Word)	Установка способа (Pattern) и цвета (Color) закраски. Pattern: 0 – цвет фона, 1 – однородное заполнение цветом, 2 – горизонтальные линии, 3 – диагональные линии, 4 – диагональные толстые линии, 5 – диагональные линии, 6 – диагональные

	толстые линии, 7 – клетка, 8 – косая клетка, 9 – частая клетка, 10 – редкие точки, 11 – частые точки, 12 – определяется пользователем.

Дополнительные функции, позволяющие получить информацию о графическом режиме и цветах:

Формат	Описание
GetMaxX	Максимальное значение X
GetMaxY	Максимальное значение Y
GetX	Координата X графического курсора
GetY	Координата Y графического курсора
GetMaxColor	Максимальный код цвета
GetPaletteSize	Количество цветов в палитре
GetColor	Текущий установленный цвет рисунка
GetBkColor	Текущий установленный цвет фона

Существует достаточно широкий набор дополнительных возможностей реализуемых функциями и процедурами модуля Graph, однако, в данном пособии ограничимся рассмотрением вышеприведённых.

Пример. Составить программу вывода на экран следующего изображения:

```
Program DrawPicture;
 uses crt,graph;
 var gd,gm,i:integer;
BEGIN
 gd:=VGA; gm:=VGAHi;
 InitGraph(gd,gm,'');
{пирамида (фиолетовая)}
 setcolor(5);
 for i:=0 to 11 do
  rectangle(50+i*10,450-i*15,300-i*10,450-i*15+15);
{пумпушка на пирамиде (зелёная)}
 setcolor(10);
 ellipse(175,260,0,360,10,25);
{звёздное небо из 5000 случайно разбросанных
 серых звёзд}
 for i:=1 to 5000 do putpixel(random(640),random(200),7);
{жёлтая луна}
 setcolor(14);
 ellipse(500,80,270,90,30,40);
 ellipse(500,80,270,90,60,40);
 setfillstyle(1,14); floodfill(535,80,14);
```

```
 ReadKey; CloseGraph;
END.
```

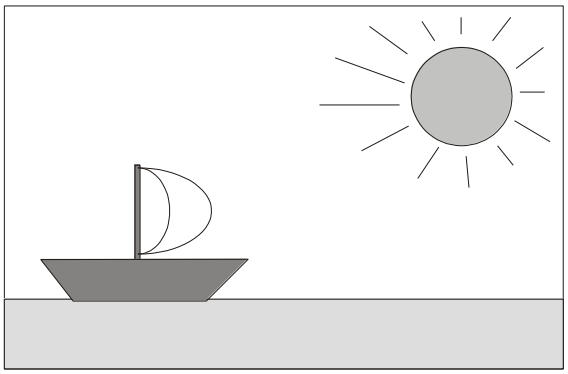
Контрольные вопросы:

1. Что подразумевает графический режим экрана?
2. Каким образом инициализировать графический режим экрана?
3. С помощью каких примитивов осуществляется рисование?
4. В чём отличие цвета фона от цвета рисунка?
5. Рисование каких примитивов осуществляется цветом фона, а каких цветом рисунка?
6. Каковы форматы процедур осуществляющих рисование с помощью примитивов?
7. Какая мера (градусная или радианная) используется в процедурах рисования эллиптических дуг и дуг окружности?

Задания:

1. Изобразить на экране картину звездного неба, состоящего из 10000 случайно расположенных звезд случайного цвета.

2. Изобразить на экране следующую картину:



3. Творческое задание. Самостоятельно придумать и составить программу вывода произвольного графического изображения.

Занятие № 14. Вывод текста в графическом режиме.

Графический режим экрана также предоставляется более широкие возможности по выводу текста. О них и поговорим конкретнее.

Если в текстовом режиме вывод символов осуществлялся только по знакоместам, то в случае графического экрана можно осуществлять произвольный вывод текста, ориентируя его не только по горизонтали, но и по вертикали и пользоваться различными шрифтами, а также осуществлять их масштабирование. В этом случае стандартные процедуры вывода не подойдут и на помощь к нам приходят также процедуры и функции модуля Graph.

Название	Вид	Формат	Описание
OutText	процедура	OutText(TextString: string)	Вывод строки с позиции графического курсора
OutTextXY	Процедура	OutTextXY (x,y: integer; TextString: string)	Вывод текстовой строки по указанной координате верхнего левого угла надписи
SetTextStyle	Процедура	SetTextStyle(Font, Direction, CharSize: Word)	Установка параметров вывода текста: **Font**: 0 – по умалчанию, 1 – TriplexFont, 2 – SmallFont, 3 – SansSerifFont, 4 – GothicFont (1..4 при наличии соответствующих файлов *.chr (для поддержки кириллицы требуются соответствующие русифицированные файлы шрифтов)) **Direction**: 0 – горизонтально, 1 – вертикально **CharSize**: коэффициент масштабирования
SetTextJustify	процедура	SetTextJustify(Horiz, Vert: Word)	Способ выравнивания текста: **Horiz:** 0 – LeftText (лево) 1 – CenterText (центр) 2 – RightText (право) **Vert:** 0 – BottomText (верх) 1 – CenterText (центр) 2 – TopText (низ)
TextHeight	функция	TextHeight(TextString: string): Word	Возвращает высоту строки в пикселях
TextWidth	функция	TextWidt(TextString: string): Word	Возвращает ширину строки в пикселях

Пример. Составить программу вывода в графическом режиме в верхней части экрана слово «Испытание» и далее слово «Example» различными шрифтами и цветом с указанием номер шрифта и цвета.

```
Program TryFonts;
 uses crt,graph;
 var gd,gm,i,c:integer;
```

```
     s,t:string;
BEGIN
 gd:=VGA; gm:=VGAHi;
 InitGraph(gd,gm,'');
 settextstyle(0,0,3); setcolor(12);
 outtextxy(200,10,'Испытание'); randomize;
 for i:=0 to 4 do
  begin
   settextstyle(i,0,1);
   c:=random(16); setcolor(c);
   str(i,s); str(c,t);
   outtextxy(10,i*40+100,'Example (font # '+s+
             ', color # '+t+').');
  end;
 ReadKey;
 CloseGraph;
END.
```

Контрольные вопросы:

1. В чём отличие вывода текста в текстовом и графическом режимах?
2. С помощью каких процедур осуществляется вывод текста в графическом режиме? В чём отличие между ними?
3. Какие дополнительные возможности предоставляет вывод текста в графическом режиме?

Задания:

1. Составить программу рисования шахматного поля с указанием подписей к клеткам.

2. Составить программу рисования осеннего (зимнего, летнего, весеннего) пейзажа с надписями названия картины и автора.

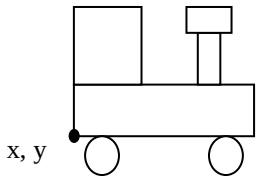
Занятие № 15. Простейшая анимация.

Большой интерес в использовании графического режима представляет анимация. Разберём наиболее простейшие её принципы. (Вообще при анимации объектов очень удобно пользоваться различными видеостраницами для формирования последовательности кадров, однако этот метод мы оставим на самостоятельно рассмотрение).

Итак, кино – это великий обман, изобретённый братьями Люмьерами. На самом деле за кажущимся движением на экране кроется череда статических кадров с фазами движения, быстрая смена которых, в силу инерционности человеческого зрения, даёт эффект реального движения. Вот и мы поступим таким же образом. Пусть требуется воспроизвести эффект движущегося паровозика. Для этого следует действовать по следующему алгоритму:

1. Зададим начальные координаты объекта.
2. Прорисуем объект с указанными координатами.
3. Немного подождём (чтобы зафиксировать изображение на сетчатке глаза)
4. Сотрём объект.
5. Изменим координаты местоположения объекта.
6. Если не достигнут предел движения перейдём к пункту 2.

Пример. Составить программу эмитирующую эффект движения паровозика:




```
program move_train;
 uses crt, graph;
 var gd,gm,x,y:integer;
BEGIN
 gd:=VGA; gm:=VGAHi; initgraph(gd,gm,'');
 x:=0; y:=350;
 while (not keypressed) and (x<640) do
  begin
    {рисуем пакровозик}
    setfillstyle(1,12);
    bar(x,y,x+80,y-30); rectangle(x,y-30,x+30,y-70);
    circle(x+15,y+10,10); circle(x+65,y+10,10);
    rectangle(x+60,y-30,x+70,y-60);
    rectangle(x+55,y-60,x+75,y-70);
    {ждём}
    delay(1000);
    {стираем паровозик}
    setfillstyle(1,0);
    bar(x,y+20,x+80,y-70);
    {изменяем координаты}
    inc(x);
  end;
 closegraph;
END.
```

Комментарии:

`Delay` – это процедура приостанавливающая работу ЭВМ на указанное количество миллисекунд (в нашем случае подбирается экспериментально в зависимости от быстродействия компьютера).

`KeyPressed` – функция не приостанавливающая работы программы и возвращающая True, если в момент обращения к ней была нажата какая-либо клавиша и False - в противном случае.

Контрольные вопросы:

1. В чём заключается идея анимации?
2. Для чего используется задержка в анимации объекта?
3. Придумайте свой способ анимации?

Задания:

1. Составить программу, изображающую на экране взлет ракеты.

2*. Составить программу, эмитирующую на экране движения бильярдного шара (с учётом отражения от бортов).

Занятие № 16. Построение графиков функций.

Графический режим предоставляет так же широкие возможности для математической графики, в частности для построения графиков функций.

Построение графиков функций можно производить двумя способами: поточечным и кусочно-линейным. В первом случае, график функции представляет собой совокупность точек, и чем гуще эти точки расставлены. Тем ближе внешний вид графика к реальному. Во втором случае, график строится из отрезков, и чем короче они, тем полученный график болебе точно отображает действительный.

Будем пользоваться для примеряя первым способом. Пусть нам требуется построить график некоторой функции y=F(x), тогда для построения графика потребуется произвести следующие действия:

1. Определить интервал аргумента, в котором будем рассматривать функцию.
2. Для выбранного интервала аргументов, прикинуть приблизительные максимальное и минимальное значения результатов.
3. Построить график по точкам, двигаясь от минимального значения аргумента к максимальному с заданным шагом.

Пример. Составить программу построения графика функции $y=x^2-100$ в интервале $-10 \le x \le 20$.

```
program plot_func;
 uses graph,crt;
 var gd,gm:integer;
     x,y,dx:real;
BEGIN
 gd:=VGA; gm:=VGAHi; initgraph(gd,gm,'');
{рисуем оси координат}
 line(0,240,640,240); line(320,0,320,480);
 x:=-20; dx:=0.05;
{строим график функции}
 while x<20 do
  begin
   y:=sqr(x)-100;
{ставим точки с учётом сдвига к центру экрана
    и перевортом оси ординат}
   putpixel(320+round(x),240-round(y),12);
{изменяем значение аргумента на подобранный шаг}
   x:=x+dx;
  end;
 readkey;
 closegraph;
END.
```

В приведённой программе есть один серьёзный недостаток – это ручной подбор параметров графика. Здесь мы встречаемся с проблемой автоматического подбора коэффициентов масштабирования. Автоматическое масштабирование легко произвести для функций не терпящих разрыв и не имеющих большого градиента в рассматриваемом интервале. Тогда автоматический расчёт коэффициентов масштабирования сводится к предварительному проходу по значениям функции и нахождению её экстремумов в рассматриваемом интервале.

Пример. Составить программу построения графика функции $y=sin2(x)-cos(x)$ в интервале $-4\pi \le x \le 3\pi$.

```
program plot_funcauto;
 uses graph,crt;
 var gd,gm:integer;
     x,y,dx,ymax,k,kx,ky:real;
BEGIN
 gd:=VGA; gm:=VGAHi; initgraph(gd,gm,'');
 line(0,240,640,240); line(320,0,320,480);
 x:=-4*pi;
{определения шага по оси абсцисс}
 dx:=(4*pi+3*pi)/640;
 ymax:=0;
{предварительный проход для определения максимального
 значения функции}
 repeat
   y:=sqr(sin(x))-cos(x);
   if abs(y)>ymax then ymax:=abs(y);
   x:=x+dx;
 until x>=3*pi;
{определение коэффициентов масштабирования по осям x и y}
 ky:=240/ymax; kx:=320/(4*pi);
{выбираем для масштабирования меньший коэффициент,
 чтобы график целиком поместился на экране}
 if kx>ky then k:=ky else k:=kx;
 x:=-4*pi;
{строим график}
 while x<3*pi do
  begin
   y:=sqr(sin(x))-cos(x);
   putpixel(320+round(x*k),240-round(y*k),12);
   x:=x+dx;
  end;
 readkey;
 closegraph;
END.
```

Контрольные вопросы:

1. Какие способы построения графиков функции вы знаете?
2. Какие действия следует выполнить в программе строящей график функции?
3. Как осуществить автоматическое масштабирование графика?
4. Какие существуют ограничения, накладываемые на функцию при её автоматическом масштабировании? С чем это связано?

Задания:

1. Построить график функции $y = \dfrac{\sin^2 x - \cos x}{\cos^2 x + 2}$
2. Построить график функции $\begin{cases} x = \cos^3 t \\ y = \sin^3 t \end{cases}$

Занятие № 17. Самостоятельная работа.

Занятие № 18. Вспомогательный алгоритм. Процедуры и функции пользователя.

Часто в программировании возникает ситуация, когда в программе встречаются фрагменты, выполняющие однотипные действия. В этом случае имеет смысл вынести этот фрагмент в отдельный алгоритм. Такой алгоритм получил название вспомогательного. Вспомогательный алгоритм – это алгоритм, вызываемый из тела основного. Вспомогательный алгоритм также получил название подпрограммы. В Pascal такие алгоритмы бывают двух типов: процедуры и функции. Отличие заключается в том, что функция, в результате своей работы, всегда возвращает значение и только одно. Процедура же может возвращать одно, либо более значений, а может и не возвращать их вовсе. Прежде, чем использовать какой-либо вспомогательный алгоритм в программе, его следует описать в описательной части (см. структуру программы, Занятие № 2). А именно, указать каковы будут аргументы и результаты (впрочем, их может и не быть). Описание вспомогательных алгоритмов в Pascal происходит последовательно и имеет следующую структуру:

Процедура

```
procedure <имя процедуры>[(<входящие параметры>;
                           var <исходящие параметры>)];
 [<описательная часть>];
begin
 <тело процедуры>
end;
```

[ ] – обозначают необязательную часть и при программировании опускаются.

[<входящие параметры>; var <исходящие параметры>] – являются необязательными.

[<входящие параметры>] – переменные (с указанием их типа), в которые передаются значения из основной программы.

[var <исходящие параметры>] – переменные (с указанием типа) значения которых возвращаются в качестве результата работы.

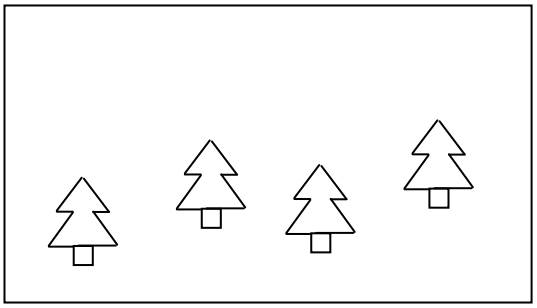
[<описательная часть>] – стандартная описательная часть, совпадающая с описательной частью программы.

Функция

```
function <имя функции>[(<входящие параметры>)]:<тип>;
 [<описательная часть>];
begin
 <тело функции>
end;
```

После описание вспомогательного алгоритма к нему можно обращаться стандартным образом. Приведём ряд примеров:

Пример 1 (использования процедуры). Написать программу, выводящую на экран следующее изображение:



Заметим, что достаточно создать вспомогательный алгоритм (процедуру), рисующий одну ёлочку, в качестве параметров которому передаются координаты (например) верхушки, и задача сводится к четырём её вызовам.

```
program four_tree;
 uses graph,crt;
 var gd,gm:integer;
     x,y,dx,ymax,k,kx,ky:real;
procedure tree(x,y:integer);
begin
 moveto(x,y); linerel(-20,30);
 linerel(10,0); linerel(-40,60);
 linerel(100,0); linerel(-40,-60);
 linerel(10,0); linerel(-20,-30);
 rectangle(x-10,y+90,x+10,y+110);
end;
BEGIN
 gd:=VGA; gm:=VGAHi; initgraph(gd,gm,'');
 tree(50,300); tree(200,200);
 tree(350,250); tree(500,150);
 readkey;
 closegraph;
END.
```

Для иллюстрации работы функции напишем программу, которая помогла бы нам найти НОД (наибольший общий делитель) трёх положительных чисел. Для этого воспользуемся формулой:

НОД(a,b,c)=НОД(a,НОД(b,c))

Для нахождения НОД двух чисел опишем вспомогательный алгоритм (функцию), которая будет иметь два аргумента – числа, и один результат – НОД этих двух чисел.

Для нахождения НОД применим алгоритм Евклида описанный в некоторых школьных учебниках (более эффективный алгоритм описан на занятии № 11). Он заключается в последовательном вычитании от большего числа меньшего (и запись результата на место большего) до тех пор, пока они не сравняются. Тогда программа может принять следующий вид:

```
program nod3;
 uses crt;
 var a,b,c:word;
function nod2(x,y:word):word;
begin
 while x<>y do
   if x>y then x:=x-y else y:=y-x;
 nod2:=x;
end;
BEGIN
 ClrScr;
 write('Введите первое число: '); readln(a);
 write('Введите второе число: '); readln(b);
 write('Введите третье число: '); readln(c);
 writeln('НОД(',a,',',b,',',c,')=',nod2(a,nod2(b,c)));
 readkey;
```

```
END.
```

Контрольные вопросы:

1. Что такое вспомогательный алгоритм?
2. Каково назначение вспомогательного алгоритма?
3. В чём отличие процедуры от функции?
4. Приведите примеры использования вспомогательного алгоритма?
5. Каков формат описание процедуры пользователя в Pascal?
6. Каков формат описание функции пользователя в Pascal?

Задания:

1. Составить программу нахождения НОД четырёх чисел.

2. Составить программу вычисления площади кольца по значениям внутреннего и внешнего радиусов, используя подпрограмму вычисления площади круга.

3. Составить программу определения какие из трёх введенных с клавиатуры целых положительных чисел являются простыми (для проверки числа на предмет простоты использовать вспомогательный алгоритм).

Занятие № 19. Структурные типы данных. Понятие линейного массива.

Наряду с простыми типами данных в Pascal существует возможность использования структурных (составных) типов данных, которые предоставляют достаточно гибкие возможности. Одним из таких типов данных являются массивы.

Поговорим более подробно о линейных массивах.

Линейный массив (вектор) – это совокупность пронумерованных данных одного типа обозначенных одним именем.

Линейный массив очень удобно представить в виде таблицы. Например, если бы меня интересовала бы информации о средней температуре воздуха каждого месяца за 2002 год, то эту информацию можно бы было представить в виде следующей таблицы:

	Месяц											
	1	2	3	4	5	6	7	8	9	10	11	12
t^0,C	-21,2	-20,5	-14,6	-5,4	+7,0	+15,2	+21,7	+20,1	+10,8	-2,3	-15,4	-17,9

Таким образом, в данном примере, данные о температуре представляют собой линейный массив.

Каждый элемент в линейном массиве однозначно идентифицируется его номером. А сам массив (в концепции Pascal) обладает именем и типом (подразумевается тип данных, которые будут храниться в нём).

Прежде, чем начать работу с линейным массивом его следует объявить в разделе **var** следующим образом:

var <имя>:array [<нач. №> . . <кон. №>] of <тип элементов>;

<имя> - имя массива (по правилам именования переменных);

<нач. №> - номер первого элемента в массиве;

<кон. №> - номер последнего элемента;

Количество элементов в массиве называется размерностью массива.

Нумерация элементов массива осуществляется одним из скалярных типов данных.

Например,

```
var tem: array [1..12] of real;
```

описывает массивы с именем **tem** в котором 12 элементов номера которых начинаются с **1**, а заканчиваются **12**, причём каждый элемент может содержать в себе число типа **real**. Размерность массива равна 12 элементам.

Обращение (чтение, запись) в массиве возможно только к одному его элементу в данный момент времени.

Например,

```
tem[3]:=-14.6;
```

придание значения элементу или

```
writeln(tem[10]);
```

вывод значения элемента на экран.

Существуют несколько способов заполнения массива.

Приведём примеры заполнения некоторыми из них:

1. Заполнение вводом с клавиатуры:

Опишем массив и переменную для цикла:

```
var p: array [0..5] of integer;
    i: byte;
```

Заполним массив:

```
for i:=0 to 5 do
  begin
   write('Введите номер ',i,'-го элемента: ');
   readln(p[i]);
  end;
```

2. Заполнение случайными числами (например, в интервале от -17 до 68)***:***

Опишем массив и переменную для цикла:

```
var i:byte;
     er:array [3..25] of integer;
```

Заполним массив:

```
 for i:=3 to 25 do
  er[i]:=random(86)-17;
```

3. Заполнение с использованием закономерностей (например, 1, 2, 4, 8, 16, 32, …)***:***

Опишем массив и переменную для цикла:

```
var i: byte;
     po: array [0..10] of integer;
```

Заполним массив:

```
 po[0]:=1;
 for i:=1 to 10 do
  po[i]:=po[i-1]*2;
```

4. Описание предустановленных массивов.

Описание осуществляется в разделе описания констант, где указывается не только размерность массива, но и значение каждого его элемента, например, так:

```
const arr1: array [1..5] of byte=(2,4,8,16,32);
```

5. Заполнение данными из внешнего файла.

(Рассмотрим более подробно на занятии № 27)

Вывод элементов массива на экран.

Это можно сделать тоже с помощью цикла. Например, для 3-го случая заполнения фрагмент программы вывода элементов массива на экран примет следующий вид:

```
for i:=0 to 10 do writeln(po[i]);
```

В некоторых случаях (2,3,4) можно также осуществлять вывод симметрично с заполнением. Например, для 2-го случая фрагмент программы преобразится к следующему виду:

```
for i:=3 to 25 do
  begin
   er[i]:=random(86)-17;
   writeln(er[i]);
  end;
```

Следует знать одну важную вещь: ***тип данных string на самом деле является линейным массивом элементов типа char (string ≡ array [0.255] of char)*** . Это означает, что мы можем обращаться к строке поэлементно, как и к элементам линейного массива. Например,

Опишем тип данных:

```
var s: string;
```

Фрагмент программы:

```
s:='аппроксимация';
writeln(s[10]);
```

В результате на экран будет выведена десятая буква слова (‘а’).

? Контрольные вопросы:

1. Что такое линейный массив?
2. Как описывается линейный массив?
3. Какие способы заполнения линейного массива существуют?
4. Что из себя представляет тип string?
5. Какие операции можно производить над элементами линейного массива?

✔ Задания:

1. Составить программу, заполняющую и выводящую на экран линейный массив размерность 20 элементов целыми случайными числами, сгенерированными на отрезке от –50 до 25.

2. Заполнить линейный массив A размерностью 40 элементов, используя следующую закономерность:

A[1]:=1; A[2]:=-2; A[3]:=3; A[4]:=-4; A[5]:=5; …

Заполненный массив вывести на экран.

3. Составить программу, заполняющую массив значениями функции $y = \frac{1}{2} \cdot \left(\cos(x) - \sin^2(x)\right)$, где x изменяется на отрезке от -2π до 2π, с шагом $\frac{\pi}{6}$

Замечание: количество элементов массива определить самостоятельно.

Занятие № 20. Простейшие задачи на операции с линейным массивом.

При работе с линейными массивами возникает класс стандартных задач, о которых мы немного расскажем на этом занятии.

1. Подсчёт количества элементов удовлетворяющих некоторому условию.

Идея алгоритма заключается в последовательном переборе всех элементов и сверке их с определённым. При этом, например, можно увеличивать счётчик количества найденных элементов.

Пример. Написать программу подсчёта всех элементов кратных трём в массиве размерностью 40 элементов, заполненном целыми случайными числами, сгенерированными в интервале от 2 до 100.

```
program les20_1;
 uses crt;
 var a:array [1..40] of byte;
     n,i:byte;
BEGIN
 ClrScr; Randomize;
 for i:=1 to 40 do
  begin a[i]:=random(99)+2; write(a[i]:4); end;
 writeln;
 n:=0;
 for i:=1 to 40 do
  if a[i] mod 3=0 then inc(n);
 writeln(n,' чисел кратных трём.');
 readkey;
END.
```

2. **Статистическая обработка массива.**
(сумма, произведение, количество и т.д.)

3. Раскрашивание массива.

4. Поиск минимального и максимального элементов.

5. копирование и перестановка элементов.

Занятие № 21. Сортировка линейного массива.

Решая задачи связанные с линейными массивами возникает ситуация его сортировки. Под сортировкой линейного массива понимают способ приведения элементов массива к упорядоченному состоянию. Выделяют вид сортировки и метод сортировки.

Под видом сортировки будем понимать то, как выглядит отсортированный массив. Существует два простых видов сортировки: по не убыванию и по не возрастанию. В случае отсутствия в массиве элементов с одинаковыми значениями сортировки могут носить названия: по возрастанию и по убыванию соответственно. Для массивов содержащих данные символьного типа сортировка по возрастанию – это сортировка в алфавитном порядке, по убыванию – сортировка в обратном алфавитном порядке.

Метод сортировки подразумевает реализацию процесса приведения массива к упорядоченному состоянию. Существует значительное количество различных методов сортировки, среди которых, есть и высокоскоростные (например, сортировка с разделением Хоара, сортировка методом слияний).

Мы разберём два наиболее простых метода.

1. Сортировка прямого обмена.

Предположим, что мы производим сортировку по возрастанию. Тогда на первом шаге нам следует найти в массиве элемент с минимальным значением и поменять его местами с первым элементом. Массив разбивается на две части: первая (состоящая пока из одного элемента) – отсортированная и вторая (начиная со 2-го) – не отсортированная. Теперь найдём в не отсортированной части элемент с минимальным значением и поменяем его местами теперь со вторым элементом в массиве (или с первым, но в не отсортированной части). Теперь отсортированная часть возросла (она состоит из двух элементов), а не отсортированная уменьшилась (на 1 элемент). Действуя аналогичным образом далее легко перевести элементы линейного массива к отсортированному состоянию. Для простоты понимания метода приведём таблицу, в которой проиллюстрируем описанный нами алгоритм:

№ элемента	Начальное состояние	Шаг				
		1	**2**	**3**	**4**	**5**
1	3	-5	-5	-5	-5	-5
2	7	7	0	0	0	0
3	-5	3	3	1	1	1
4	0	0	7	7	3	3
5	1	1	1	3	7	4
6	4	4	4	4	4	7

Тогда для массива размерностью N фрагмент программы сортировки примет следующий вид:

```
for i:=1 to N-1 do
 begin
  nmin:=i;
  for k:=i+1 to N do
    if a[nmin]>a[k] then nmin:=k;
  tmp:=a[nmin]; a[nmin]:=a[i]; a[i]:=tmp;
 end;
```

2. Обменная сортировка (метод «Пузырька»).

Будем производить сортировку по убыванию. Второй способ сортировки заключается в следующем: осуществляем циклический проход по элементам массива,

начиная с первого и до последнего (N-го). Будем сравнивать два соседних элемента. Если элементы расположены не правильно (в случае сортировки по убыванию первый (*i*-ый) меньше второго (*i+1*-го)), то поменяем их местами. Если в течение такого прохода по массиву произошла хотя бы одна перестановка, то следует повторить проход. Если ни одной перестановки не произошло, то массив уже находится в упорядоченном состоянии.

Проиллюстрируем описанный алгоритм в виде таблицы:

№ элемента	**Начальное состояние**	**Проход**				
		1	**2**	**3**	**4**	**5**
1	3	7	7	7	7	7
2	7	3	3	3	4	4
3	-5	0	1	4	3	3
4	0	1	4	1	1	1
5	1	4	0	0	0	0
6	4	-5	-5	-5	-5	-5

Формализация описанного алгоритма может принять следующий вид:

```
repeat
 t:=true;
 for i:=1 to n-1 do
  if a[i]<a[i+1] then
   begin
    t:=false; tmp:=a[i]; a[i]:=a[i+1]; a[i+1]:=tmp;
   end;
until t;
```

где `t: boolean` – переменная-признак были обмены или нет.

Занятие № 22. Понятие двумерного массива.

Проще всего представить двумерный массив данных как таблицу. Из такого представления видно, что каждый элемент должен определяться двумя координатами.

Двумерный массив (матрица) – это именованная совокупность данных одного типа, где каждое данное однозначно идентифицируется двумя координатами: номером строки и номером столбца.

Строго говоря, что указывать сначала номер строки, а затем номер столбца или наоборот всё равно, однако, принято указывать в качестве первой координаты номер строки, а затем номер столбца. Мы не будем отходить от данной концепции.

Приведём пример двумерного массива, заполненного случайными числами:

№	1	2	3	4	5
1	4	0	-3	2	3
2	3	2	-5	5	5
3	-1	7	1	4	-9

Представленный массив имеет размерность 3×5 = 15 элементам.

Первый вопрос – это вопрос об описании такого массива. Определение двумерного массива осуществляется так же как и определение линейного, однако, надо помнить, что мы имеем дело с двумя измерениями. Тогда, во-первых, можно объявить наш линейный массив как набор линейных массивов. Для приведённого примера, такое объявление может выглядеть, например, вот так:

```
var q: array[1..3] of array [1..5] of integer;
```

Во-вторых, осуществить явное объявление двумерного массива (что значительно проще). Для нашего случая, объявление примет следующий вид:

```
var q: array[1..3, 1..5] of integer;
```

Возможно также объявление двумерного массива через константы. Такое объявление как правило значительно удобнее предыдущих:

```
const n = 3; m = 5;
var q: array[1..n, 1..m] of integer;
```

Обращение к элементу двумерного массива производится так же как и к элементу линейного массива, но с указанием двух координат:

```
q[2,3]:=-5;
```

Вопрос о заполнении двумерного массива решается по тем же правилам, как и заполнение линейного массива, однако, теперь для прохода по всем элементам двумерного массива потребуется два цикла, один из которых будет вложенным. Приведём в качестве примера, два (из пяти) способа заполнение двумерного массива:

1) Заполнение двумерного массива вводом с клавиатуры

```
for i := 1 to n do
 for j := 1 to m do begin
  write('Введите элемент ',i,'-й строки, ',
        j,'-о столбца: ');
  readln(a[i,j]);
end;
```

2) Заполнение двумерного массива случайными числами (целыми, сгенерированными в интервале [–50; 50]):

```
randomize;
for i := 1 to n do
 for j := 1 to m do a[i,j]:=random(100)-50;
```

Следует сказать пару слов о выводе на экран двумерного массива. Вывод его на экран лучше всего оформить в виде таблицы. Тогда фрагмент программы вывода (для массива размерностью n×m) примет следующий вид:

```
for i:=1 to n do
 for j:=1 to m do begin
  gotoxy((j-1)*4+1, i); write(a[i,j]:4);
end;
```

Занятие № 23. Решение задач на тему «Двумерные массивы».

Проще всего представить двумерный массив данных как таблицу. Из такого

Занятие № 24. Самостоятельная работа.

Занятие № 25. Множества.

Множество — это структурированный тип данных, представляющий собой набор взаимосвязанных по какому-либо признаку или группе признаков объектов, которые можно рассматривать как единое целое. Каждый объект во множестве называется ***элементом множества.*** Все элементы множества должны принадлежать одному из скалярных (перечислимых) типов, кроме вещественного. Этот тип называется ***базовым типом*** множества. Базовый тип задается диапазоном или перечислением. ***Область значений типа множество*** — набор всевозможных подмножеств, составленных из элементов базового типа.

В выражениях на языке Pascal значения элементов множества указываются в квадратных скобках: [1,2,3,4], ['a','b','c'], ['a'..'z']. Если множество не имеет элементов, оно называется ***пустым*** и обозначается как [ ]. Количество элементов множества называется его ***мощностью.***

Для описания множественного типа используется словосочетание set of (множество из...)

Например:

```
var a: set of 1..12;
```

В этом случае описано множество a, которое может включать в себя элементы – целые числа в интервале от 1 до 12. Существует возможность описания константных множеств:

```
const EvenDigits: set of 0..9 = [0, 2, 4, 6, 8];
      Vowels: set of 'A'..'Z' = ['A', 'E', 'I', 'O', 'U',
                                 'Y'];
      HexDigits: set of '0'..'z' = ['0'..'9', 'A'..'F',
                                    'a'...f'];
```

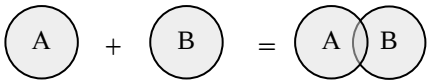
Над множествами можно производить некоторые операции. Для того. Чтобы проиллюстрировать эти операции введём два произвольных множества и обозначим их A и B.

1. Операции сравнения

Обозначение в Pascal	Описание
A = B	Проверка на равенство
A <> B	Проверка на неравенство (True в случае, если множества различаются своими элементами)
A >= B	Множество A больше (содержит все элементы множества B и хотя бы ещё один элемент, отсутствующий в множестве B), либо равно множеству B.
A <= B	Множество A меньше, либо равно множеству B.

2. Объединение множеств

Объединением множеств A и B называется множество, содержащее все элементы множества A и множества B. Для пояснения удобно использовать графическое обозначение этой операции:



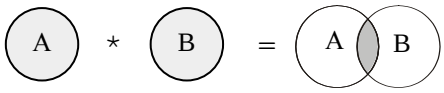
В Pascal эта операция обозначается знаком +. Посредством этой операции можно осуществлять включение необходимых элементов во множество, при условии, что они удовлетворяют базовому типу:

```
A := [1,3,5];
```

```
A := A+[7];
```

3. Пересечение множеств

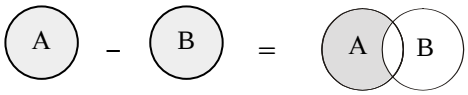
Пересечением множеств A и B называется множество, содержащая только те элементы, которые содержаться и в множестве A и в множестве B.



В Pascal эта операция обозначается обычным знаком *.

4. Разность множеств

Разностью множеств называется множество состоящее из всех элементов множества A, которые отсутствуют в множестве B.



5. Проверка вхождения элемента во множество

Осуществляется с помощью служебного оператора **in** следующим образом:

```
<элемент> in <множество>
```

Результат работы true, если элемент входит во множество и false в противном случае.

Приведём небольшой пример программы работы со множеством:

```
Type digits = set of 0..9;
var D1,D2,D3,D: digits;
BEGIN
 D1:=[2,4,6,8];
 D2:=[0..3,5];
 D3:=[1,3,5,7,9];
 D:=D1+D2; {D=[0,1,2,3,4,5,6,8]}
 D:=D+D3;  {D=[0,1,2,3,4,5,6,7,8,9]}
 D:=D-D2;  {D=[4,6,7,8,9]}
 D:=D*D1;  {D=[4,6,8]}
END.
```

В результате работы программы формируется множество D состоящее из 3 элементов [4, 6, 8].

Множество элементов является непечатаемым, поэтому если требуется вывести множество на экран потребуется проверить все возможные элементы на вхождение их во множество. Проиллюстрируем эту идею на примере программы:

```
program maltitude;
 uses crt;
 const n: set of 1..12=[1,3,5,7,8,10,12];
 var i: integer;
BEGIN
 clrscr;
 for i:=1 to 12 do
  if i in n then write(i:3);
 readkey;
END.
```

Задача: составить программу, определяющую количество дней в указанном месяце, указанного года.

Для удобства решения этой задачи напишем функции, возвращающую количество дней, а также опишем два множества номеров месяцев в которых 30 и 31 день

соответственно. Месяц февраль будем анализировать отдельно, т.к. количество дней в нём зависит от номера года.

```
program months;
 uses crt;
 const d31 = [1,3,5,7,8,10,12];
       d30 = [4,6,9,11];
 function DayInMonth(m: byte; y: word): byte;
  begin
   if m in d30 then DayInMonth:=30;
   if m in d31 then DayInMonth:=31;
   if m=2 then
    if (y mod 4=0) and (y mod 100<>0) or
       (y mod 100=0) and (y mod 400=0) then
         DayInMonth:=29
      else DayInMonth:=28;
  end;
 var y: word; m: byte;
BEGIN
 clrscr;
 write('Введите номер месяца и года: '); readln(m,y);
 writeln('Количество дней: ', DayInMonth(m,y));
 readkey;
END.
```

Контрольные вопросы:

1.

Задание:

1. Составить программу, осуществляющую вывод всех элементов множества A, заданного следующим образом: A: set of [1,4,6,7,19,25,30,42,90].

2. Подсчитать количество гласных букв во введенном с клавиатуры слове.

3. Составить программу, формирующую множество простых чисел в интервале от 1 до 100.

Занятие № 26. Записи.

Запись – это структурированный тип, описывающий набор данных различного типа. Иными словами, запись представляется собой структуру, обладающую одним именем, но содержащую в себе несколько данных одного или различного типа. Составляющие запись объекты получили название **полей записи.** Каждое поле имеет уникальное имя и обладает типом.

В начале следует описать тип данных запись в разделе описания типов:

```
Type <имя типа запись> = record
        <поле 1>: <тип>;
        <поле 2>: <тип>;
        ...
      end;
```

Например, опишем тип данных, включающий в себя информацию о фамилии, имени, росте некоторого человека:

```
type TPeaple_info = record
        name, sur_name: string;
        height: byte;
      end;
```

или тип данных, содержащий информацию о дате:

```
type TDateRec = record
      year: word;
      month: 1..12;
      day: 1..31;
     end;
```

Теперь можно создавать переменные указанного типа и работать с ними. Описание переменной осуществляется стандартным образом. Например:

```
var a, b: TPeaple_info;
    date: TDateRec;
```

Следует помнить, что в данный момент времени вы можете обратиться только к одному полю записи на предмет чтения, либо записи данных. Обращение осуществляется указанием имени поля переменной типа запись через точку:

```
<имя переменной типа запись>.<имя поля>
```

в нашем случае это может выглядеть следующим образом:

```
a.name := 'Иван';
a.sur_name := 'Иванов';
date.year := 1970;
writeln(a.sur_name:14, date.year:5);
```

Единственным исключением является переприсваение значений между переменными одного типа, т.е. в нашем примере переменные a и b мы описали, как переменные одного типа (TPeople_info), значит, мы можем значения всех полей одной переменной присвоить другой:

```
a := b;
```

Приведём также пример использования записи в программе.

Задача: составить программу, осуществляющую запись в массив данных о фамилии, имени и дате рождения 10 человек, а затем выводящую введённую информацию в алфавитном порядке.

Пояснение: при решении данной задачи будем использовать структурный тип данных запись, которым будет являться каждый элемент массива; осуществить сортировку массива предоставим самому читатель (см. Занятие № 21).

```
program record_demo;
```

```
uses crt;
const n = 10;
type TMen_info = record
      sur_name, name: string[14];
      year: word;
      month: 1..12;
      day: 1..31;
     end;
var a: array [1..n] of TMen_info;
    i: byte;
BEGIN
clrscr;
for i:=1 to n do begin
 writeln('Данные № ', i);
 write(' Введите фамилию: '); readln(a[i].sur_name);
 write(' Введите имя: '); readln(a[i].name);
 write(' Введите день, месяц и год рождения
        (через пробел): ');
 readln(a[i].day, a[i].month, a[i].year);
end;
{фрагмент сортировки линейного массива}
clrscr;
for i:=1 to n do
 writeln(a[i].sur_name:15, a[i].name:15,
         a[i].day:3, a[i].month:3, a[i].year:5);
readkey;
END.
```

При работе с записями, для облегчения доступа к полям можно использовать оператор with:

```
with <запись> do
  <непосредственное обращение к полям записи>;
```

Например, в предыдущем примере можно слегка уменьшить текст программы вывода массива записей на экран:

```
for i:=1 to n do
 with a[i] do
  writeln(sur_name:15, name:15, day:3, month:3, year:5);
```

Таким образом, видно, что использование структурных типов данных, позволяет создавать более читаемые и понятные программы.

Контрольные вопросы:

1. Что такое запись?
2. Что такое поле записи?
3. Возможно ли обращение ко всем данным записи одновременно?
4. Каким образом описывается структурный тип данных запись?
5. Какие операции можно выполнять над данными записи?
6. Для каких целей используется структура with?

Задание:

1. Разработать программу, формирующую список учащихся и их оценок по физике, химии, информатике. Подсчитать средний балл всех учащихся в списке и найти учащегося с максимальным средним баллом.

2. Создать программу, позволяющую определить количество дней между двумя датами. Для хранения даты использовать запись.

Занятие № 27. Файлы данных. Текстовые файлы.

При написании более сложных программ возникает необходимость доступа к внешним фалам данных. Например, запись какой-либо информации для длительного хранения; доступ к уже существующим фалам с целью чтения, либо записи данных и т.д.

Определимся с некоторыми понятиями.

Файл – это именованная область памяти (как правило дисковой) ЭВМ.

Все файлы можно классифицировать по различным основаниям (классификация довольно условная). Приведём две классификации наиболее интересующие нас.

1. По способу доступа:
   а) Файлы последовательного доступа – обращаться к этим файлам можно либо на предмет чтения, либо на предмет записи информации (но не одновременно), при этом для того, чтобы, например, считать пятое данное, требуется считать сначала первые четыре. Аналогичная ситуация возникает при записи.
   б) Файлы произвольного доступа – обращаться к таким файлам можно и для чтения и для записи. При этом имеется доступ к любому данному в таком файле (минуя предыдущие).
2. По наличию структуры:
   а) Текстовые файлы – файлы, содержащие в себе текст в некоторой кодировке без специальных символов форматирования.
   б) Типизированные файлы – файлы, имеющую чёткую структуру. Как правило, представляют собой как набор записей, имеющих единую структуру.
   в) Нетипизированные файлы – файлы, к которым осуществляется прямой доступ на предмет чтения (либо записи) байта, либо совокупности байт информации.

Следует заметить, что структура файла зависит, прежде всего, от нашего отношения к нему. Например, текстовый файл вполне возможно рассматривать как нетипизированный, типизированный как текстовый и т.д.

На этом занятии рассмотрим более подробно текстовые файлы последовательного доступа. Следует сразу же заметить, что приёмы, описанные на этом занятии, фактически универсальны для работы с любыми внешними файлами данных.

Для работы с внешним файлом данных, прежде всего, следует описать файловую переменную, относительно которой в дальнейшем будет происходить обращение к файлу. Для текстовых файлов такое описание осуществляется следующим образом:

```
var <файловая переменная>: text;
```

Теперь в программе следует связать файловую переменную с реальным файлом на диске:

```
assign(<файловая переменная>, <путь и имя файла>);
```

Следует помнить, что путь и имя файла это данные символьного типа.

После того как связь установлена можно приступить к открытию файла. Для текстового файла это можно сделать одним из трёх способов:

1) `reset(<файловая переменная>);` - открыть файл для чтения. Чтение осуществляется начиная с первой строки, либо первого символа в файле. Если файл не существует, возникает сообщение об ошибке, работа программы прерывается.
2) `rewrite(<файловая переменная>);` - открыть файл для чтения. Если файл не существует, то он создаётся. Если же файл уже существует, то все данные из него удаляются.
3) `append(<файловая переменная>);` - открыть файл для дозаписи. Запись осуществляется в конец файла. Если файл не существует, возникает сообщение об ошибке, работа программы прерывается.

После всех этих процедур следует непосредственно работа по записи, либо по чтению (в зависимости от способа открытия) данных из файла/в файл. Запись и чтение данных производится с помощью стандартных процедур ввода/вывода следующим образом:

1. Запись в файл:

```
write(<файловая переменная>, <записываемые данные>);
writeln(<файловая переменная>, <записываемые данные>);
```

2. Чтение из файла:

```
read(<файловая переменная>, <записываемые данные>);
readln(<файловая переменная>, <записываемые данные>);
```

И, наконец, по завершению работы с файлом следует его закрыть, воспользовавшись следующей процедурой:

```
close(<файловая переменная>);
```

Таким образом, алгоритм работы с файлом имеет следующую структуру:

1) создать файловую переменную;
2) связать файловую переменную с реальным файлом (по его пути и имени);
3) открыть файл;
4) осуществить чтение, либо запись данных;
5) закрыть файл.

При чтении данных из файла может возникнуть ситуация, в которой за ранее неизвестно количество считываемых данных. Однако, при попытке считывания несуществующих данных возникает ошибка. Выйти из данной ситуации поможет функция, позволяющая проверить есть ли далее данные для чтения. Это функция:

```
EOF(<файловая переменная>)
```

Функция возвращает True, в случае, если конец файла достигнут и False, если конец файла ещё не достигнут и можно продолжать чтение (работа с функцией демонстрируется в задаче № 3).

Приведём примеры работы с внешним текстовым файлом последовательного доступа.

Пример № 1. Записать во внешний файл данных OUTPUT.TXT 12 целых случайных чисел сгенерированных в интервале [0; 100], после которых записать также сообщение «Запись окончена».

```
program text_file_1;
 var f: text; i: byte;
BEGIN
 randomize;
 assign(f, 'output.txt'); rewrite(f);
 for i:=1 to 10 do writeln(f, random(101));
 writeln(f, 'Запись окончена');
 close(f);
END.
```

Пример № 2. Считать, записанные в предыдущей задаче данные из внешнего файла и вывести их на экран.

```
program text_file_2;
 uses crt;
 var f: text; i,d: byte;
     s: string;
BEGIN
 clrscr;
 assign(f, 'output.txt'); reset(f);
 for i:=1 to 10 do begin
  readln(f, d); writeln(d);
```

```
 end;
 readln(f, s); writeln(s); close(f);
 readkey;
END.
```

Пример № 3. Вывести на экран произвольный текстовый файл (поэкранно) и указать сколько в нём строк.

```
program text_file_2;
 uses crt;
 var f: text; n: word;
     s: string;
BEGIN
 clrscr; n:=0;
 write('Введите путь и имя файла: '); readln(s);
 clrscr;
 assign(f, s); reset(f);
 while not eof(f) do begin
   readln(f, s); writeln(s); inc(n);
   if n mod 23=0 then begin
    writeln; write('Нажмите любую клавишу...');
    readkey; clrscr;
   end;
 end;
 close(f); writeln; writeln('Всего ',n,' строк.');
 readkey;
END.
```

Контрольные вопросы:

1.

Задание:

Занятие № 28. Решение задач (текстовый файл).

Занятие № 29. Файлы данных. Типизированные файлы.

При работе с внешними файлами данных встречаются также ситуации, когда данные, записанные в файл, имеют чёткую структуру. В этом случае используют типизированные файлы данных произвольного доступа (напомним, что для таки файлов можно осуществлять и чтение и запись данных за один сеанс открытия файла, причём такой доступ осуществляется к любым данным).

Структура типизированного файла представлена совокупностью однотипных записей как одного из базовых типов, так и структурного типа данных. Соответственно, описание файловой переменной для работы с файлом будет происходить через базовый тип или определение структурного типа данных - запись следующим образом:

```
type TExample = record
          ...
       end;
var f1: file of byte;
    f2: file of real;
    f3: file of TExample;
```

После такого описания подразумевается, что файл, с которым будем работать далее, имеет именно указанную структуру: f1 – последовательность байт информации, f2 – последовательность чисел типа real, f3 – последовательность записей. Алгоритм работы с типизированным файлом очень похож на алгоритм, приведённый на занятии № 27, однако имеет некоторые отличия.

Также следует определить переменную точно такого же типа, как и тип файла для того, чтобы можно было осуществлять чтение и запись данных:

```
var a1: byte;
    a2: real;
    a3: TExample;
```

Следующее действие – это ассоциация файловой переменной с реальным файлом:

```
assign(<файловая переменная>, <путь и имя файла>);
```

Затем открываем файл в режиме произвольного доступа с помощью стандартной процедуры:

```
reset(<файловая переменная>);
```

Для очистки, либо создания файла можно также открыть его с помощью уже знакомой нам процедуры

```
rewrite(<файловая переменная>);
```

Для чтения и записи используются стандартные процедуры ввода/вывода соответственно:

```
read(<файловая переменная>, <записываемые данные>);
write(<файловая переменная>, <записываемые данные>);
```

Во время работы с типизированным файлом данных используется понятие указателя. ***Указатель – это невидимый маркер, указывающий данные, относительно которых в данный момент времени будет осуществляться чтение, либо запись.*** При открытии файла указатель автоматически позиционируется на первую запись, которая имеет номер 0. При записи или чтении указатель автоматически смещается на одну позицию. Позиционирование указателя в произвольное место осуществляется с помощью процедуры:

```
seek(<файловая переменная>, <№ позиции в файле>);
```

Например, если требуется записать (либо считать) сразу же 6-ю запись, то перед записью (чтением) следует применить процедуру:

```
seek(F, 5);
```

Во время работы с типизированным файлом удобно пользоваться также следующими функциями:

```
filepos(<файловая переменная>)
```

функция возвращает номер позиции указателя в файле.

```
filesize(<файловая переменная>)
```

функция возвращает количество записей в файле.

```
EOF(<файловая переменная>)
```

назначение этой функции нам уже знакомо с занятия № 27.

По завершении работы с типизированным файлом следует его закрыть:

```
close(<файловая переменная>);
```

Проиллюстрируем работу с внешним типизированным файлом данных произвольного доступа на примерах.

Пример № 1. Записать во внешний файл данных INFO.DAT информацию о фамилии, имени и отчестве 5 людей. Данные о человеке представить в виде записи.

```
program arbitrary_file_1;
 uses crt;
 type TPerson = record
       name, surname, patronymic: string[15];
      end
 var f: file of TPerson;
     a: TPerson; i: byte;
BEGIN
 clrscr;
 assign(f, 'INFO.DAT'); reset(f);
 for i:=1 to 5 do begin
  write('Введите фамилию: '); readln(a.surname);
  write('Введите имя: '); readln(a.name);
  write('Введите отчество: '); readln(a.patronymic);
  write(f, a);
 end;
 close(f);
END.
```

Пример № 2. Определить информационный объём указанного файла:

```
program arbitrary_file_2;
 uses crt;
 var f: file of byte; s: string;
     size: longint;
BEGIN
 clrscr; write('Введите путь и имя файла: ');
 readln(s);
 assign(f, s); reset(f);
 size:=filesize(f);
 writeln('Информационный объём ',size,' byte.');
 close(f); readkey;
END.
```

При иллюстрации этого примера напомним, что вид файла по способу доступа зависит прежде всего от нашего отношения к нему.

Пример № 3. Дописать в файл INFO.DAT (см. пример № 1) данные об ещё одном человеке.

Решение данной задачи приведём фрагментарно.

```
seek(f, filesize(f));
write(f, a);
```

Контрольные вопросы:

1.

Задание:

Занятие № 30. Самостоятельная работа.

Прежде чем указать ряд заданий решим одну важную задачу: определить существует ли указанный файл в указанном месте на диске. Задача эта действительно важна, т.к. в приведённых выше примерах, связанных с открытием файла для чтения, может сложиться ситуация, в которой открываемый файл не существует, что, в свою очередь, приведёт к неминуемому появлению ошибки периода выполнения и аварийному завершению программы.

Для исключения подобных ситуаций воспользуемся метаоператорами. ***Метаоператор – это оператор, влияющий на процесс компиляции и выполнения программы.*** Метаоператоры имеют следующую структуру:

{$<метаоператор><параметры>}

Метаоператор пишется именно в фигурных скобках, тем самым напоминая ремарку, однако ею не является.

Для решения нашей задачи нам потребуется метаоператор I с параметрами + и -. Этот метаоператор позволяет включить и выключить обработку ошибок ввода/вывода. Воспользуемся также функцией `IOResult`, возвращающий 0 в случае если ошибок ввода/вывода не происходило и код ошибки в противном случае. С учётом всего вышесказанного можно прийти к следующей программе:

```
program file_exist;
 uses crt;
 var f: file of byte; s: string;
BEGIN
 clrscr; write('Введите путь и имя файла: ');
 readln(s); assign(f, s);
{$I-}
 reset(f);
{$I+}
 if IOResult=0 then begin
     writeln('Файл существует.'); close(f);
    end
  else writeln('Файл не существует.');
 readkey;
END.
```

Занятие № 31. Библиотеки пользователя.

Замечательной возможностью среды программирования Turbo Pascal является организация библиотек (модулей) констант, типов, переменных и самое главное вспомогательных алгоритмов.

Дело в том, что программист, создавая вспомогательные алгоритмы, часто нуждается в их использовании в целом классе программ. И в этом случае, чтобы не переписывать (копировать) одни и те же фрагменты кода из программы в программу можно просто создать отдельную библиотеку и за тем, по мере надобности, обращаться к её содержимому из различных программ.

Структура библиотеки напоминает саму программу, однако, она не может быть выполнена – максимум скомпилирована.

Библиотека пишется, так же как и программа, в отдельном окне среды программирования Turbo Pascal. Структура библиотеки имеет следующий вид:

```
UNIT <имя модуля>; {желательно, чтобы заголовок модуля совпадал с именем
                    файла в котором он хранится физически}
Interface
 uses <список модулей>; {подключаемые библиотеки}
 const <список констант с указанием значений>; {описание констант}
 type <список типов с описанием>; {описание типов пользователя}
 var <список переменных с указанием типа>; {описание переменных}
 procedure …; {заголовки процедур}
 function … ; {заголовки функций}
Implementation
 uses <список модулей>; {подключаемые библиотеки}
 label <список меток для переходов>; {объявление меток}
 const <список констант с указанием значений>; {описание констант}
 type <список типов с описанием>; {описание типов пользователя}
 var <список переменных с указанием типа>; {описание переменных}
 procedure …; {описание процедур}
 function … ; {описание функций}
BEGIN
 <инициализационная часть (выполняется при подключении модуля к
  программе) >
END.
```

Следует заметить, что в структуре модуля 3 важные части:

1-я) начинается со служебного слова `Interface` – это область объявлений, в этой области объявляется всё то, что будет доступно в основной программе после подключения вашего модуля;

2-я) начинается со служебного слова `Implementation` – это область реализаций, в этой части описываются все объявленные вспомогательные алгоритмы и всё то (константы, переменные, типы, процедуры, функции и т.д.), что требуется для нужд самого модуля и не будет доступно подключившей его программе;

3-я) между служебными словами BEGIN … END - это область инициализации, т.е. код, который выполняется автоматически при подключении модуля к основной программе.

Приведём пример построения и использования модуля. Первым делом создадим модуль, состоящий из двух общедоступных вспомогательных алгоритмов (будем

предполагать, что сами вспомогательные алгоритмы читатель уже в состоянии написать самостоятельно):

```
UNIT MyUnit;
Interface
 procedure swap(var a,b: integer);
 function LeapYear(year: word): Boolean;
Implementation
 procedure swap(var a,b: integer);
   var c: integer;
  begin
    {Действия, позволяющие поменять местами значения переменных a и b
     посредством третьей переменной c}
  end;
 function LeapYear(year: word): Boolean;
  begin
    {Действия, позволяющие определить високосность года по его номеру}
  end;
BEGIN
END.
```

Создание на этом простейшего модуля закончено. Остаётся только сохранить его под именем MyUnit и откомпилировать, чтобы на диске появился файл с именем MyUnit.TPU.

Теперь откроем новое окно редактора (для удобства) и создадим программу, использующую один из вспомогательных алгоритмов этого модуля.

```
Program UnitTest;
 uses crt, MyUnit;
 var y: word;
BEGIN
 ClrScr; write('Введите номер года'); readln(y);
 If LeapYear(y) then writeln('год високосный.')
   else writeln('год не високосный.');
 readkey;
END.
```

Обратите внимание, что в этом примере происходит обращение к функции, которая находится во внешнем модуле. Теперь в любой программе, в которой потребуется определять високосность года достаточно подключить уже существующую библиотеку и воспользоваться уже готовым вспомогательным алгоритмом.

Т.о., использование и накопление библиотек является ещё одним средством для быстрого построения программ на основе уже существующих разработок.

Контрольные вопросы:

1.

Задание:

Занятие № 32. Некоторые процедуры и функции модуля DOS.

На этом занятии мы более детально познакомимся с процедурами, функциями и типами ещё одного модуля интегрированного в стандартный файл системы Turbo Pascal turbo.tpl. Эта библиотека носит имя DOS, так как обеспечивает доступ к некоторым функциям дисково-операционной системы. Не забудем, первым делом, подключить эту библиотеку в описательной части программы:

```
uses dos;
```

Первые две процедуры позволят вам удалить и переименовать существующий файл. Для удаления используется процедура:

```
erase(<файловая переменная>);
```

Соответственно, первым делом следует ассоциировать файловую переменную с реальным файлом, а затем производить удаление:

```
assign(<файловая переменная>,<путь и имя файла>);
erase(<файловая переменная>);
```

Следует помнить, что невозможно удаление открытого файла.

Еще одна процедура, действующая аналогичным образом, позволит вам переименовать файл:

```
rename(<файловая переменная>, <новое имя файла>);
```

Теперь рассмотрим две функции, позволяющие определить информационный объем общего и занятого пространства соответственно:

```
DiskSize(<код устройства>): Longint;
```

Функция возвращает полный информационный объём диска в byte. <код устройства> - это число, где 0 – устройство по умолчанию, 1 – диск A, 2 – диск B и т.д. Работа этой функции ограничена типом результата возвращаемого ею. Не сложные подсчёты покажут, что вам не удастся определить правильный информационный объём диска превышающего 2 Gb. Например, если мы хотим увидеть информационный объём диска D в Mb, то можно воспользоваться следующей строкой:

```
WriteLn(DiskSize(4)/1024/1024:0:2);
```

Ещё одна функция возвращает объём свободного дискового пространства:

```
DiskFree(<код устройства>): Longint;
```

Часто также встречается задача просмотра содержимого того, либо иного каталога. Решается она с помощью следующих двух процедур. Сначала следует воспользоваться процедурой:

```
FindFirst(<путь>; <атрибуты файлов>; <переменная типа
          SearchRec>);
```

которая позволяет найти первый удовлетворяющий требованиям файл из списка файлов. У неё следующие параметры:

<путь> - путь, по которому осуществляется поиск файлов, <атрибуты файлов> - сумма атрибутов которыми должны быть наделены искомые файлы:

Зарезервированная константа	Обозначение	Шестнадцатеричное значение
ReadOnly	Только для чтения	$01
Hidden	Скрытый	$02
SysFile	Системный	$04
VolumeID	Метка тома	$08
Directory	Каталог	$10
Archive	Архивный	$20
AnyFile	Все файлы, метки тома и каталоги	$3F

Последний параметр – это переменная куда помещаются результаты поиска. Она должна быть обязательно зарегистрированного в модуле DOS типа SearchRec, который выглядит следующим образом:

```
type SearchRec = record
      Fill: array[1..21] of Byte;
      Attr: Byte;
      Time: Longint;
      Size: Longint;
      Name: array[0..12] of Char;
     end;
```

Fill – поле содержит служебную информацию DOS, Attr – код атрибутов найденного первого файла, Time – дата и время создания файла в специальном упакованном 4-х байтовом формате, Size – информационный объём в byte, Name – имя файла.

Для распаковки упакованных даты и времени используется процедура:

```
UnpackTime(<переменная типа Longint>;<переменная типа
            DateTime>);
```

Тип DateTime также описан в модуле DOS и имеет следующую структуру:

```
type DateTime = record
          Year, Month, Day, Hour, Min, Sec: Word;
     end;
```

О назначении этих полей нетрудно догадаться.

Для последующего поиска файлов с описанными в FindFirst параметрами используется процедура:

```
FindNext(<переменная типа SearchRec>);
```

Возникает вопрос: когда закончится список найденных файлов? Чтобы отследить эту ситуацию следует обратиться к переменной:

```
DosError: integer
```

Если содержимое переменной равно 0, то можно успешно продолжать поиск, если отлично от нуля, то возникла ошибка – файлы в списке исчерпаны (2- файл не найден, 3 – путь не найден и т.д.) и именно в этом случае следует закончить поиск. Для лучшего понимания приведём фрагмент программы, позволяющий вывести на экран все файлы и каталоги корневого каталога диска C: с указанием года их создания:

```
 var FindRes: SearchRec;
     TimeRes: DateTime;
begin
 Clrscr;
 FindFirst('C:\*.*', AnyFile, FindRes);
 while DosError = 0 do begin
   UnpackTime(FindRes.Time, TimeRes);
   Writeln(FindRes.Name:12, TimeRes.Year:5);
   FindNext(FindRes);
 end;
 readkey;
end.
```

Следующие две процедуры достаточно просты. Они позволяют получить информацию о системной дате и системном времени:

```
GetDate(<год>,<месяц>,<день>,<день недели>);
GetTime(<часы>,<минуты>,<секунды>,<сотые доли секунд>);
```

Все параметры являются переменным типа Word в которые и возвращаются указанные значения. Дата возвращается в американском формате, поэтому 0 – соответствует воскресенью, 1 – понедельнику и т.д.

Обратную операцию по установки ваших системной даты и времени позволяют произвести процедуры:

```
SetDate(<год>,<месяц>,<день>);
SetTime(<часы>,<минуты>,<секунды>,<сотые доли секунд>);
```

Контрольные вопросы:

1.

Задание:

Занятие № 33. Рекуррентные последовательности и формулы. Рекурсивные алгоритмы.

В программировании существует один очень мощный метод с которым мы познакомимся на этом занятии. Он называется рекурсией. Начинающему программисту он может показаться довольно сложным именно поэтому начнём наше рассмотрение этого метода издалека. В математике достаточно часто встречаются последовательности чисел, в которых существует закономерности между каждым последующим числом в зависимости от предыдущего, либо предыдущих. Например:

```
1, 2, 3, 4, …
2, 4, 6, 8, …                                    (1)
2, 4, 8, 16, 32, …
```

Приведённые последовательности в математики называются прогрессиями (две первые арифметическими, последняя – геометрической) и выражаются специальными формулами, которые показывают как выражается некоторый произвольный член в зависимости от предыдущего. Например для арифметической прогресси можно записать следующую формулу:

$a_i=a_{i-1}+d$

Формулы, выражающие очередной член последовательности через один или несколько предыдущих членов, называются рекуррентными соотношениями.

Приведем иную последовательность: 1, 1, 2, 3, 5, 8, 13, …

В этой последовательности каждый следующий член равен сумме двух предыдущих членов. Такая последовательность получила название чисел Фибоначчи (под Фибоначчи - Леонардо Пизанский, итальянский математик, живший на рубеже ХП-ХШ в.в., известен "Книгой абака" в которой систематично изложил достижения арабских математиков для Западной Европы). Для такой последовательности можно записать следующую рекуррентную формулу:

$a_i=a_{i-1}+a_{i-2}$ $a_1=a_2=1$

Счетчик (типа a=a+1) является также примером рекуррентного вычисления величины.

Теперь попытаемся написать программу, которая будет вычислять n-ый член такой последовательности. Понятно, что в этом случае очень удобно воспользоваться циклом (например с параметром) и написание такой программы предоставим читателю на самостоятельную работу.

Программы производящие те же действия (расчёт n-го члена) можно написать и с помощью метода информатики – рекурсии.

Можно рассмотреть еще один пример такой последовательности:

последовательность 2, 3, 5, 9, 17, ..., задаваемую рекуррентным соотношением $a_i=2*a_{i-1}-1$, $a_i=2$. В этом случае можно легко написать алгоритм рекуррентного вычисления членов последовательности. ***Под рекурсией понимают ситуацию, когда вспомогательный алгоритм вызывает сам себя непосредственно, либо посредством дополнительных вспомогательных алгоритмов.***

Этот процесс не может продолжаться бесконечно, поэтому требуется также предусмотреть ***условие, при котором подобные вызовы прекратятся – условие выхода из рекурсии.***

Например, для вычисления n-го члена последовательности (1) можно написать следующую функцию:

```
function row(n: integer): integer;
 begin
  if n=1 then row:=2 else row:=2+row(n-1);
```

```
end;
```

Прокомментируем работу данной функции. Предположим, что нужно вычислить 3-й член для этого требуется узнать второй, а, в свою очередь, для этого требуется узнать первый, который известен – это 2. Т.о., знание первого члена числового ряда и есть условие выхода из рекурсии. Для того, чтобы узнать третий член потребуется вызов функции с параметром 3:

```
WriteLn(row(3));
```

При это функция вызывает себя с параметром 2, затем снова с параметром 1, однако, у нас стоит условие, где указывается, что в случае значения параметра n=1 требуется возвратить результат 2. Попытаемся пояснить также эту ситуацию на равенстве:

```
row(3)=2+row(2)=2+2+row(1)=2+2+2=6
```

Приведём ещё один пример. Составим рекурсивную функция для расчета n-го члена ряда Фибоначчи. Воспользуемся тем, что нам известны два первых члена ряда (условие выхода из рекурсии):

```
function fib(n: integer):integer;
 begin
  case n of
   1,2: fib:=1;
   else fib:=fib(n-1)+fib(n-2);
  end;
 end;
```

Теперь следует понять одну очень важную речь, что рекурсии можно распространить не только на числовые ряды, но и на ситуации. Например, для того, чтобы перевести целое положительное число из десятичной системы счисления в любую другую позиционную систему счисления по основанию P требуется производить поэтапное целочисленное деление на основание системы счисления в которую осуществляется перевод (P) и собирать целочисленные остатки в обратном порядке (см. школьный курс информатики). Эта ситуация рекурсивна, поэтому её можно запрограммировать с помощью нашего метода. Для простоты решения задачи создадим дополнительную функцию, позволяющую осуществлять преобразования числового типа данных в строковый, т.к. результат работы нашей основной функции будет строка, являющаяся числом в системе счисления по основанию P. Приведём функцию, позволяющую осуществлять преобразование десятичных чисел в двоичную систему счисления (P=2).

```
function IntToStr(n: integer): string;
  var t: string;
 begin str(n,t); IntToStr:=t; end;
function DecToBin(a: word): string;
 begin
  if a>0 then
       DecToBin:=DecToBin(a div 2)+IntToStr(a mod 2)
     else DecToBin:='';
 end;
```

Рекурсивные алгоритмы, могут быть нескольких видов.

Рекурсия бывает:

1. К*онечной* и *бесконечной*. Бесконечный рекурсивный вызов вспомогательного алгоритма приводит к ошибке переполнения стека, т.к. при каждом вызове происходит запоминание очередного адреса перехода к рекурсивному вспомогательному алгоритму, что требует выделения памяти стека.

2. *Прямой* и *косвенной*. Прямой рекурсия считается в том случае, если вызов вспомогательного алгоритма осуществляется непосредственно самим вспомогательным алгоритмом. При вызове вспомогательного алгоритма посредством других вспомогательных алгоритмов будем иметь дело с косвенной рекурсией.

3. *Терминальной* и *нетерминальной*. Терминальной рекурсия считается, если рекурсивный вызов осуществляется последней командой вспомогательного алгоритма. В противном случае рекурсия нетерминальная.

Например, рекурсия, использованная в первом алгоритме по расчёту чисел Фибоначчи является конечной, прямой, терминальной.

Контрольные вопросы:

1. Определите вид рекурсии в примере № 2 приведённом на занятии.

Задание:

1. Последовательность чисел 2, 3, 5, 9, 17, … задается следующим рекуррентным соотношением: $a_i=2*a_{i-1}-1, \quad a_i=2$

Вычислить значение n-го члена последовательности использовав рекуррентное соотношение:

а) с помощью цикла;

б) используя рекурсию.

2. Напишите функцию, осуществляющую рекурсивный поиск числа.

3. Найти рекуррентное соотношение и составить программу для расчета n-го члена следующей последовательности:

1, 2, 6, 24, 120, 720,…

4. Найти рекуррентное соотношение и составить программу для расчета n-го члена следующей последовательности:

1, 2, 11, 26, 57, 120,…

Занятие № 35. Элементы объектно-ориентированного программирования.

Современное программирование
Понятие объектно-ориентированного программирования.
(первое использование ООП в языке Симула (70-е) в Pascal с 1989 (5.5))
Понятие класса объектов. Описание классов.

Класс – определенный пользователем тип данных, включающий в себя состояние (представление класса) и некие операции (поведение класса). Класс имеет: внутренние данные; методы в виде процедур и функций. Назначение класса: описание общего поведения и характеристик набора аналогичных друг другу объектов.

Объект – экземпляр класса (м.б. переменная, тип которой является классом). Объекты реальны, т.е. во время выполнения программы хранятся в памяти. Соотношения между объектом и классом аналогичны соотношениям между переменной типом.

Объект – нечто существенное и определённое, для которого существует способ отличить один подобный объект от другого.

Объект – объединение подпрограмм и данных, предназначенных для их обработки. Объект – абстрактное понятие; экземпляр объекта – конкретное.

Принципы: инкапсуляция (объединение данных и обрабатывающих их методов внутри класса; объект скрывает в себе детали, которые несущественны для использования объекта), наследование (порождение новых объектов-потомков от предков, наследование всех свойств; определение новых объектов, используя свойства старых, дополняя или изменяя их), полиморфизм (методы различных объектов могут иметь одинаковые имена, но различные свойства).

```
type TCalendar=object
        constructor init;
        function GetDay: byte;
       private
        day, month: byte;
        year: word;
      end;
```

Свойства, методы, события объекта.

```
program ob_demo;
 uses crt, dos;
 type TCalendar=object
       day,month,year,wday: word;
       procedure Init;
       procedure SetSystemDate;
       procedure ShowDate;
       procedure SetMyDate(d,m,y: word);
       function LeapYear: boolean;
       procedure NextDay;
      end;
 procedure TCalendar.Init;
  begin
   SetSystemDate;
  end;
 procedure TCalendar.SetSystemDate;
  begin
   getdate(year, month, day, wday);
  end;
 procedure TCalendar.ShowDate;
  begin
   if day<10 then write('0',day) else write(day);
   if month<10 then write('-0',month) else write('-',month);
```

```
   writeln('-',year);
  end;
 procedure TCalendar.SetMyDate;
  begin
   day:=d; month:=m; year:=y;
  end;
 function TCalendar.LeapYear: boolean;
  begin
   if (year mod 100<>0) and (year mod 4=0) or
      (year mod 100=0) and (year mod 400=0) then LeapYear:=true
     else LeapYear:=false;
  end;
 procedure TCalendar.NextDay;
   const d31=[1,3,5,7,8,10]; d30=[4,6,9,11];
  begin
   if (month in d31) and (day<31) or (month in d30) and (day<30) then
     begin inc(day); exit; end;
   if (month in d31) and (day=31) or (month in d30) and (day=30) then
     begin inc(month); day:=1; exit; end;
   if (month=12) and (day<31) then begin inc(day); exit; end;
   if (month=12) and (day=31) then
     begin day:=1; month:=1; inc(year); exit; end;
   if LeapYear and (day<29) or not LeapYear and (day<28) then
     begin inc(day); exit; end else begin day:=1; month:=3; exit; end;
  end;

var a: TCalendar;

BEGIN
 clrscr;
 a.init;
 a.SetMyDate(28,2,2001);
 a.ShowDate;
 a.NextDay;
 a.ShowDate;
 readkey;
END.
```

Список литературы:

Дагане В.А., Григас Г.К., Аугутис К.Ф. 100 задач по программированию. – М.: Просвещение, 1993 – 255с.: ил.

Каймин В.А., Щёголев А.Г., Ерохина Е.А., Федюшин Д.П. Основы информатики и вычислительной техники 10-11. – М.: Просвещение, 1990 – 273 с.: ил.

Кнут, Дональд, Эрвин. Искусство программирования, том 1. Основные алгоритмы, 3-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 720 с.: ил.

Семакин И.Г., Шестаков А.П. Лекции по программированию. – Пермь: Издательство Пермского университета, 2000 – 189 с.