# *ИНФОРМАТИКА*

*для студентов 1 курса АВТФ направлений:*
*220200-Автоматизация и управление*
*230100-Информатика и ВТ*

*к.т.н., доц. каф. Автоматики*
*Саблина Галина Владимировна*
*VII-506*
*g_sablina@ngs.ru*

# *1.Введение.Организация персонального компьютера. Основные понятия об ОС и ее функциях.*

## *1.1. Введение*

*Структура курса. Тематика лекций, лабораторных и практических работ.*

1. *Ввод данных и корректировка информации в электронной таблице;*
2. *Линейные, разветвляющиеся, циклические и итерационные алгоритмы;*
3. *Функции комплексного переменного, матрицы, СЛАУ;*
4. *Линейные, разветвляющиеся, циклические и итерационные алгоритмы в VBA;*
5. *Функции комплексного переменного и матрицы в VBA;*
6. *Строковые данные. Подпрограммы-функции;*
7. *Строковые массивы. Подпрограммы-процедуры;*
8. *Ввод/вывод в файлы. Элементы управления.*

- *Преподавательский состав:*

*Худяков Дмитрий Сергеевич, к.т.н., доцент каф. Авт;*
*Веретельникова Евгения Леонидовна, к.т.н., доцент каф. Авт;*

*Модульно-рейтинговая система*

*В рамках данной системы производится оценивание работы на практических занятиях, своевременности выполнения и защит лабораторных работ.*

*Максимальная оценка – **320 баллов**.*

- *активная работа на практическом занятии – **10 баллов**;*
- *пассивная работа на практическом занятии – **5 баллов**;*
- *отсутствие на практическом занятии – **0 баллов**;*
- *выполнение лабораторной работы по графику – **10 баллов**;*
- *выполнение лабораторной работы на следующем занятии – **5 баллов**;*
- *выполнение лабораторной работы с отставанием на 2 и более занятий – **0 баллов**;*
- *защита лабораторной работы с первой попытки в день выполнения по графику – **10 баллов**;*
- *защита лабораторной работы со второй попытки в день выполнения по графику –**5 баллов**;*

- *защита лабораторной работы с третьей и более попытки в день выполнения по графику – **0 баллов**;*
- *защита лабораторной работы с первой попытки при ее выполнении на следующем по графику занятии – **5 баллов**;*
- *защита лабораторной работы со второй и более попытки при ее выполнении на следующем по графику занятии – **0 баллов**;*
- *защита лабораторной работы с любой попытки при ее выполнении с отставанием на два и более занятий – **0 баллов**;*
- *максимальная сумма баллов за своевременное выполнение и защиту лабораторной работы не может превышать – **20 баллов**;*
- *досрочное выполнение лабораторных работ не приносит дополнительных баллов;*
- *досрочная (до выполнения лабораторной работы) защита или защита без предоставления протокола не допускается;*
- *претензии по проставленным баллам не принимаются!*

*Количество баллов, дающее право на автоматический экзамен:*

- *с оценкой «отлично» от **304 до 320 баллов (от 95 до 100 %);***
- *с оценкой «хорошо» от **289 до 303 баллов (от 90 до 95 %);***
- *с оценкой «удовлетворительно» от **272 до 288 баллов (от 85 до 90 %)**.*

*При неудачной попытке повысить оценку на экзамене, заработанный «автомат» сохраняется!*

*Список рекомендованной литературы:*


- *Волченков Н.Г. Учимся программировать: VB-5: Учеб. пособие: - М. -Диалог МИФИ, 1998.*
- *Сока Джон, Рассмел Дэн, Комл Дебра. VB 5.0: - Минск, 1998.*
- *Гуревич Н., Гуревич О. Освой самостоятельно VB 5. - М.: Бином, 1998.*
- *Сана П. и др.Visual Basic для приложений (версия 5) в подлиннике: пер. с англ.- СПб.:BHV-Санкт-Петербург, 1999.-704 с.*
- *Король В.И. Visual Basic 6.0, Visual Basic for Applications 6.0. Язык программирования. Справочник с примерами. — М.: Издательство КУДИЦ, 2000.*

- *Васильев А., Андреев А. VBA в Office 2000. Учебный курс — С-Пб.: «Питер», 2001.*
- *Биллиг В.А. Средства разработки VBA-программиста. Офисное программирование. Том 1. — М.: Издательско-торговый дом «Русская редакция», 2001.*
- *Тюнина Л.В., Гунько А.В., Саблина Г.В.. Основы информатики. Методические указания к лабораторным работам для студентов 1 курса АВТФ. Новосибирск: НГТУ, 2003. – 35 с.*

## 1.2. Организация ПК

### Поколения компьютеров

- *1 поколение. Компьютеры на электронных лампах (с 1946г.)*

*Количество ЭВМ в мире – сотни.*

*Отличительные особенности:*

- *большие размеры:30 тонн;*
- *высокое потребление энергии;*
- *быстродействие: 10-20 тысяч операций в секунду;*
- *объем ОП: 2 Кбайт;*
- *устройства ввода/вывода: перфолента, перфокарта;*
- *программное обеспечение – машинные коды;*
- *Примеры : MARK1, ENIAC, EDSAC (Electronic Delay Storage Automatic Calculator) - первая машина с хранимой программой, UNIVAC (Universal Automatic Computer) (США), Минск-1,БЭСМ (СССР).*

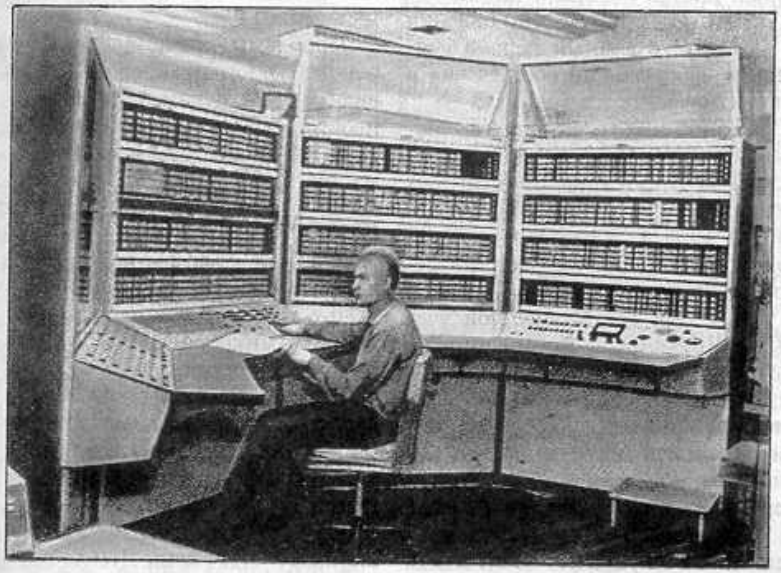
Компьютер первого поколения «Минск-1»

- *2 поколение. Компьютеры на транзисторах (с 1955 г.)*

*Количество ЭВМ в мире – тысячи;*

*Отличительные особенности:*

- *значительно меньшие размеры, чем у ламповых;*
- *быстродействие: 100 тысяч-1 миллион операций в сек.;*
- *объем ОП: 2-32 Кбайт;*
- *устройства ввода/вывода: магнитный барабан, магнитная лента;*
- *программное обеспечение: алгоритмические языки, пакетный режим;*
- *Примеры: Стретч (Англия), Атлас (США), БЭСМ6 (СССР).*



*Компьютер БЭСМ-6*

- *3 поколение.Компьютеры на интегральных схемах (с 1964г.)*

*Количество ЭВМ в мире – сотни тысяч;*

*Отличительные особенности:*

- *размер - миникомпьютер;*
- *быстродействие:1-10 миллионов операций в секунду (MIPS);*
- *объем ОП: 64 Кбайт;*
- *устройства ввода/вывода: многотерминальные системы;*
- *программное обеспечение: операционные системы, режим разделения времени;*
- *Примеры: IBM-360,370 (США),ЕС-1022,1066, СМ ЭВМ (СССР).*

Компьютер третьего поколения IBM/360

✓*4 поколение. Компьютеры на больших интегральных схемах (БИС) с 1975 г. (до 100 тысяч элементов) и сверхбольших интегральных схемах (СБИС) с 1985 г. (до 10 млн. элементов)*

*2 направления:*

*1. создание суперЭВМ - комплексов многопроцессорных машин. Быстродействие: несколько миллиардов операций в секунду. Способны обрабатывать огромные массивы информации. Примеры: ILLIAS-4, CRAY, CYBER, «Эльбрус-1», «Эльбрус-2».*

*2. дальнейшее развитие на базе БИС и СБИС микро-ЭВМ и персональных ЭВМ (ПЭВМ). Первыми представителями этих машин являются: Altair-8800 (Intel-8080, 1974 г.), Apple, IBM - PC ( XT , AT , PS /2), «Искра», «Электроника», «Мазовия», «Агат», «ЕС-1840», «ЕС-1841».*

*Начиная с этого поколения ЭВМ стали называть компьютерами.*

*Состав современного ПК:*

*В базовый (основной) комплект современного ПК входит 4 устройства:*

- *системный блок;*
- *монитор;*
- *клавиатура;*
- *манипулятор «мышь».*

*Все основные устройства ПК связаны системной магистралью (шиной).*

*В основу современных ПК положен магистрально-модульный принцип, позволяющий пользователю самому комплектовать нужную конфигурацию и при необходимости производить модернизацию.*

*Системная шина состоит из трех многоразрядных шин: адреса (ША), данных (ШД), управления (ШУ).*

*По ШД передаются данные между различными устройствами, например, от оперативной памяти к процессору для обработки и обратно в ОЗУ для хранения.*

*Размер ШД определяется разрядностью процессора, т.е.кол-вом двоичных разрядов, которое процессор обрабатывает за 1 такт. В настоящее время максимальная разрядность процессора- 64 бита.*

*Выбор устройства или ячейки памяти, куда пересылаются или откуда считываются данные по ШД определяет процессор. Каждое устройство или ячейка ОЗУ имеет свой адрес. Адрес передается по ША, причем сигналы передаются в одном направлении (от процессора к ОЗУ и устройствам). Разрядность ША определяет адресное пространство процессора, т.е. кол-во ячеек ОЗУ, которые могут иметь уникальные адреса.*

*Кол-во адресуемых ячеек может быть рассчитано по формуле:*

$$N=2^{l}$$

*где l – разрядность шины адреса. В современных ПК разрядность ША -32 бита, т.о.*

$$N=232=4294967296 \text{ шт.}$$

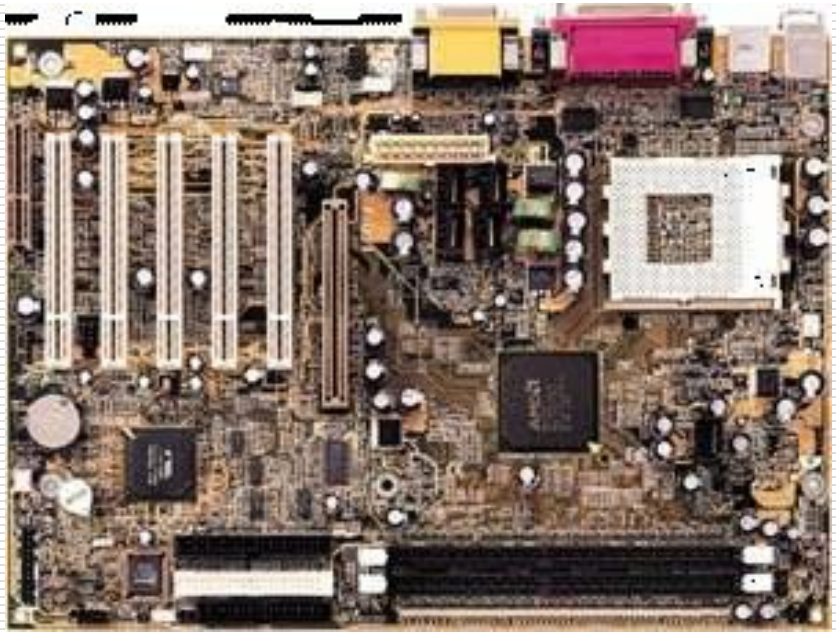
*По ШУ передаются сигналы, определяющие характер обмена информацией по магистрали. Они определяют, какую операцию (считывание или запись) информации нужно производить, синхронизируют обмен информации между устройствами и т.д.*

## *Системный блок (СБ)*

*Все основные компоненты находятся внутри СБ: системная плата с процессором и ОЗУ, накопители на жестких и гибких дисках, приводы CD/DWD-RW, блок питания.*

## *Системная плата (СП)*

*Является основным аппаратным компонентом. На ней реализована магистраль обмена информацией, имеются разъемы для установки процессора и ОЗУ, а также слоты для установку контроллеров внешних устройств.*



*Частота процессора, системной шины и шин периферийных устройств.*

*Быстродействие различных компонентов ПК может существенно различаться. Для согласования быстродействия на СП устанавливаются специальные микросхемы (чипсеты), включающие в себя контроллер ОЗУ (северный мост) и контроллер периферийных устройств(южный мост).*

*Северный мост обеспечивает обмен информацией между процессором и ОЗУ по системной шине. В процессоре используется внутреннее умножение частоты, поэтому частота процессора в несколько раз выше частоты системной шины.*

*Северный мост соединяется с южным специальной мостовой шиной. К южному мосту подключена шина PCI –Express, которая обеспечивает обмен информации с контроллерами периферийных устройств.*

*Контроллеры периферийных устройств (звуковая плата, сетевая плата) устанавливаются в слоты расширения СП.*

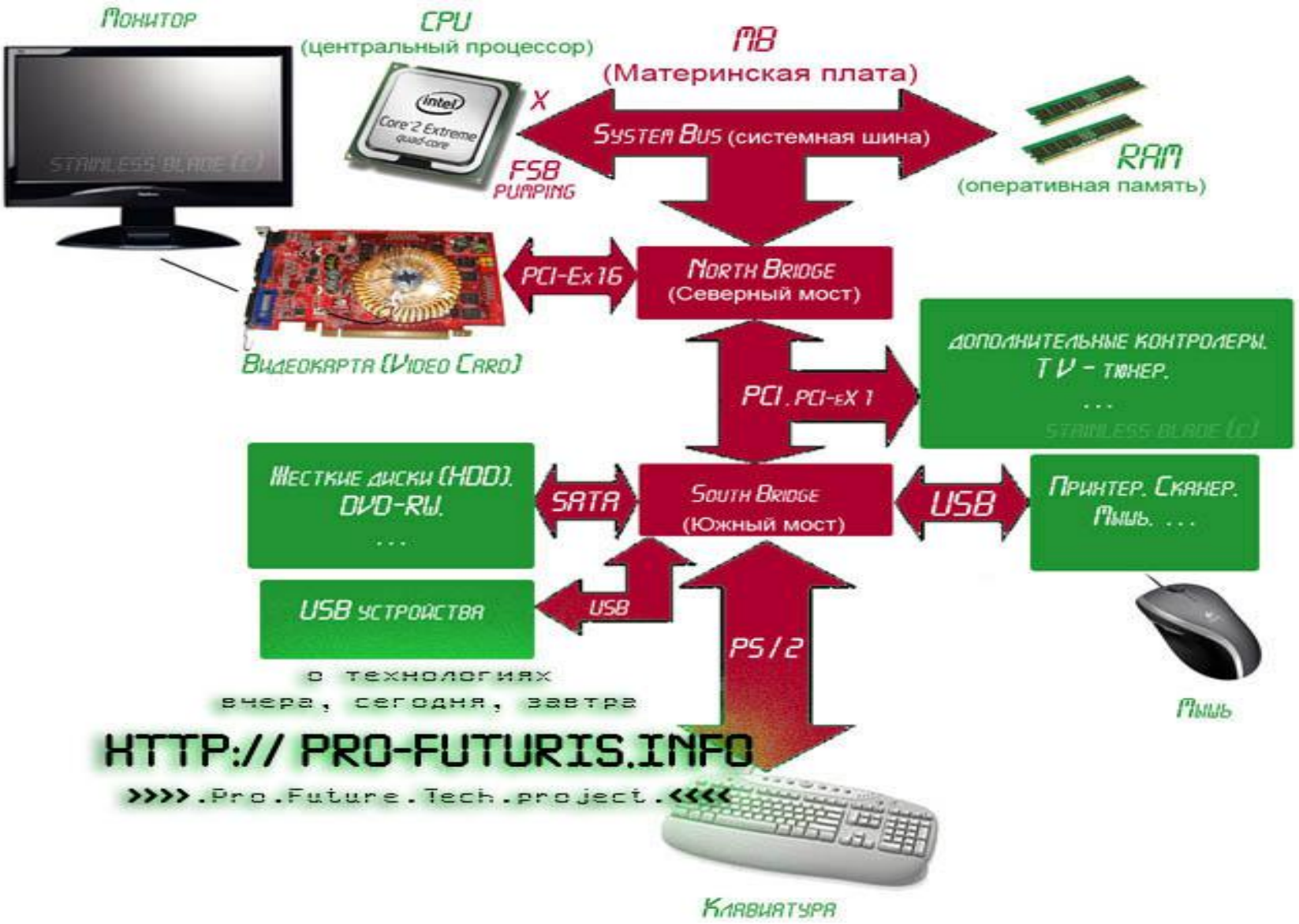
*Для подключения видеоадаптера через северный мост используется шина PCI-Express 16.*

*Южный мост обеспечивает обмен информации между северным мостом и портами для подключения периферийных устройств. Жесткие диски, CD/DWD-RW подключаются к южному мосту по последовательной шине SATA, обеспечивающей максимальную скорость передачи данных до 150 Мбит/с.*

*Принтер, сканер, мышь подключаются по южному посту при помощи интерфейса USB, который обеспечивает высокоскоростное подключение к ПК сразу нескольких устройств.*

*По этому же интерфейсу при необходимости могут быть подключены цифровая камера, цифровой фотоаппарат и т.п.*

*Клавиатура подключается к южному мосту по интерфейсу PS/2.*


Монитор
CPU
(центральный процессор)
MB
(Материнская плата)
X
System Bus (системная шина)
FSB
PUMPING
RAM
(оперативная память)
PCI-Ex16
North Bridge
(Северный мост)
Видеокарта (Video Card)
PCI, PCI-eX 1
дополнительные контролеры.
TV - тюнер.
...
Жесткие диски (HDD).
DVD-RW.
...
SATA
South Bridge
(Южный мост)
USB
Принтер. Сканер.
Мышь. ...
USB устройства
USB
PS/2
о технологиях
вчера, сегодня, завтра
HTTP:// PRO-FUTURIS.INFO
>>>>.Pro.Future.Tech.project.<<<<
Мышь
Клавиатура

*Виды памяти ПК*

*Память ПК делится на внутреннюю и внешнюю.*

*Внутренняя память состоит из ОЗУ, ПЗУ И КЭШ.*

*В ОЗУ загружаются программы, которые выполняются в данный момент. Часть ОЗУ, называемая «видеопамять», содержит данные о текущем изображении на экране. При отключении питания содержимое ОЗУ стирается. Быстродействие ПК напрямую зависит от величины ОЗУ,*

*Которая в современных ПК достигает 4 Гбайт.*

*ПЗУ постоянно хранит информацию, которая записывается туда при изготовлении ПК. В ПЗУ:*

*тестовые программы, проверяющие при каждом включении ПК правильность его работы;*

*программы для управления основными периферийными устройствами (монитором, клавиатурой);*

*информация о том, где на диске расположена операционная система.*

*КЭШ- память –это быстродействующая память, расположенная между процессором и основной памятью. Ее действие эквивалентно быстрому доступу к основной памяти.*


*Внешняя память:*

*накопители на ЖД; (1 Тбайт)*

*накопители на гибких дисках; (1.44 Мбайт)*

*CD/DVD-RW; (до 24 Гбайт)*

*накопители на FLASH – картах. (до 32 Гбайт).*

*Процессор*

*Устройство, обеспечивающее преобразование информации и управление другими устройствами ПК. Конструктивно представляет собой СБИС. Чем больше элементов на кристалле, тем выше производительность. Процессор выполняет арифметические и логические операции. Самая важная характеристика процессора – производительность (MIPS).*

*Состоит из УУ и АЛУ.*

# Периферийные устройства


# *Виды программного обеспечения*

# *2.Разработка алгоритма. Блок-схема. Структуры алгоритмов (4 час)*

## *2.1 Разработка алгоритма.*

*Алгоритм - это*

*1. Описание последовательности действий для решения задачи или достижения поставленной цели;*

*2. Правила выполнения основных операций обработки данных;*

*3. Описание вычислений по математическим формулам.*

*Перед началом разработки алгоритма необходимо четко уяснить задачу: что требуется получить в качестве результата, какие исходные данные необходимы и какие имеются в наличии, какие существуют ограничения на эти данные. Далее требуется записать, какие действия необходимо предпринять для получения из исходных данных требуемого результата.*

*На практике наиболее распространены следующие формы представления алгоритмов:*

*1. Словесная (записи на естественном языке);*

*2. Графическая (изображения из графических символов);*

*3. Псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);*

*4. Программная (тексты на языках программирования).*

*Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.*

***Пример**. Записать алгоритм нахождения **наибольшего общего делителя (НОД)** двух натуральных чисел.*

*Алгоритм может быть следующим:*

1. *задать два числа;*
2. *если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;*
3. *определить большее из чисел;*
4. *заменить большее из чисел разностью большего и меньшего из чисел;*
5. *повторить алгоритм с шага 2.*

*Определим наибольший общий делитель чисел 125 и 75.*

*125*
*75*
*50*
*25*
*25*

*Словесный способ не имеет широкого распространения по следующим причинам:*

- *такие описания строго не формализуемы;*
- *страдают многословностью записей;*
- *допускают неоднозначность толкования отдельных предписаний.*

*Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным.*

*При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.*

*Такое графическое представление называется схемой алгоритма или* ***блок-схемой****.*

*Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов.*

*Он занимает промежуточное место между естественным и формальным языками.*

*С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.*

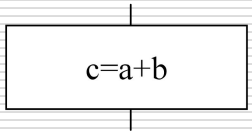
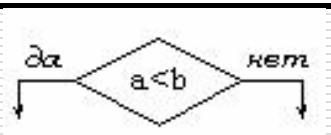
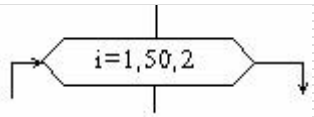
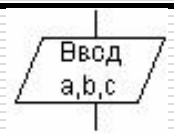
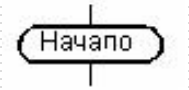
***В псевдокоде не приняты строгие синтаксические правила для записи команд**, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя.*

*Однако в псевдокоде обычно **имеются некоторые конструкции, присущие формальным языкам**, что облегчает переход от записи на псевдокоде к записи алгоритма на формальном языке.*

## *2.2 Блок-схема*

*Блок-схемой называют графическое представление алгоритма, в котором он изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.*

*В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий.*

| Название символа | Обозначение и пример заполнения | Пояснение |
|---|---|---|
| Процесс |  | Вычислительное действие или последовательность действий |
| Решение |  | Проверка условий |
| Модификация |  | Начало цикла |
| Предопределенный процесс |  | Вычисления по подпрограмме, стандартной подпрограмме |
| Ввод-вывод |  | Ввод-вывод в общем виде |
| Пуск-останов |  | Начало, конец алгоритма, вход и выход в подпрограмму |
| Документ |  | Вывод результатов на печать |

*Пример. Составить блок-схему алгоритма определения высот* ***ha, hb, hc*** *треугольника со сторонами* ***a, b, c,*** *если*

$$ha = \left(2/a\right) \bullet \sqrt{p \bullet \left(p-a\right) \bullet \left(p-b\right) \bullet \left(p-c\right)}$$

$$hb = \left(2/b\right) \bullet \sqrt{p \bullet \left(p-a\right) \bullet \left(p-b\right) \bullet \left(p-c\right)}$$

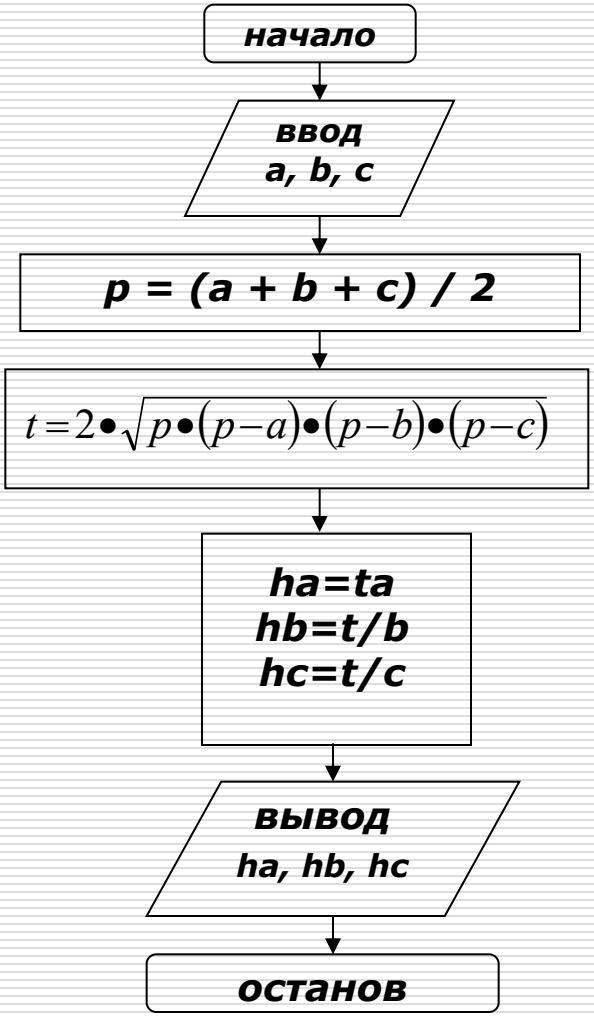
$$hc = \left(2/c\right) \bullet \sqrt{p \bullet \left(p-a\right) \bullet \left(p-b\right) \bullet \left(p-c\right)}$$

$$p = (a + b + c) / 2.$$

*Решение. Введем обозначение* $t = 2 \bullet \sqrt{p \bullet \left(p-a\right) \bullet \left(p-b\right) \bullet \left(p-c\right)}$

*тогда* ***ha = t/a, hb = t/b, hc = t/c.***

*Блок-схема должна содержать начало, ввод* ***a, b, c,*** *вычисление* ***p, t, ha, hb, hc,*** *вывод результатов и останов*


начало
ввод
a, b, c
p = (a + b + c) / 2
$t = 2 \bullet \sqrt{p \bullet (p-a) \bullet (p-b) \bullet (p-c)}$
ha=ta
hb=t/b
hc=t/c
вывод
ha, hb, hc
останов

## *2.3 Структуры алгоритмов.*

*Алгоритмы можно представлять как некоторые структуры, состоящие из отдельных базовых (т.е. основных) элементов. Естественно, что при таком подходе к алгоритмам изучение основных принципов их конструирования должно начинаться с изучения этих базовых элементов*

*Логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур: следование, ветвление, цикл.*

*Характерной особенностью базовых структур является наличие в них одного входа и одного выхода.*

*Базовая структура следование.*

*Образуется из последовательности действий, следующих одно за другим:*

*Базовая структура ветвление.*

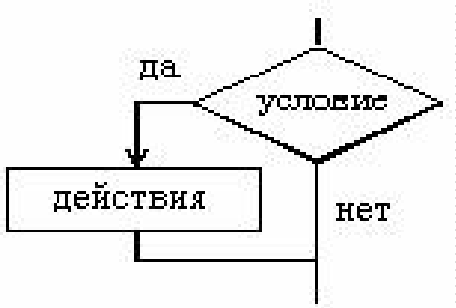
*Обеспечивает в зависимости от результата проверки условия (**да** или **нет**) выбор одного из альтернативных путей работы алгоритма. Каждый из путей ведет к **общему выходу**, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран.*

*Структура **ветвление** существует в четырех основных вариантах:*

- *если-то;*
- *если-то-иначе;*
- *выбор;*
- *выбор-иначе.*

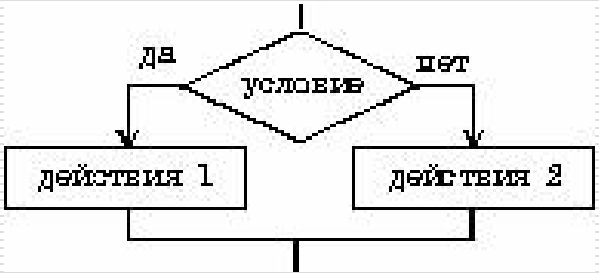
*а) если-то*

*если условие*
*то действия*
*конец если*



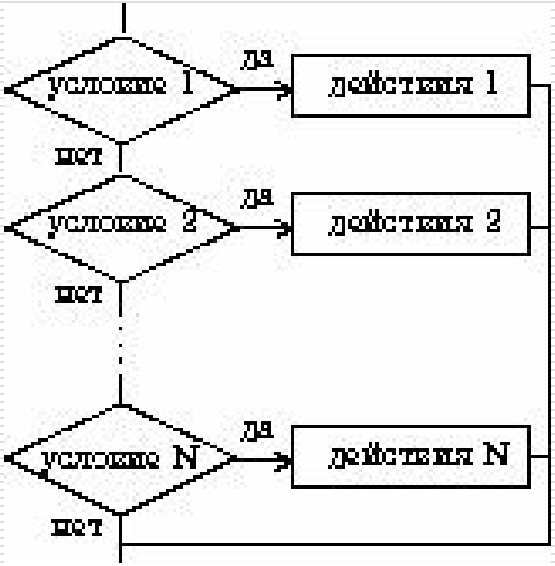
б) *если-то-иначе*

*если* условие
*то* действия 1
*иначе* действия 2
*конец если*




с) *выбор*

*выбор*
*при* условии 1: действия 1
*при* условии 2: действия 2
. . . . . .
*при* условии N: действия N
*конец выбора*



*д) выбор-иначе*

*выбор*

*при условии 1: действия 1*

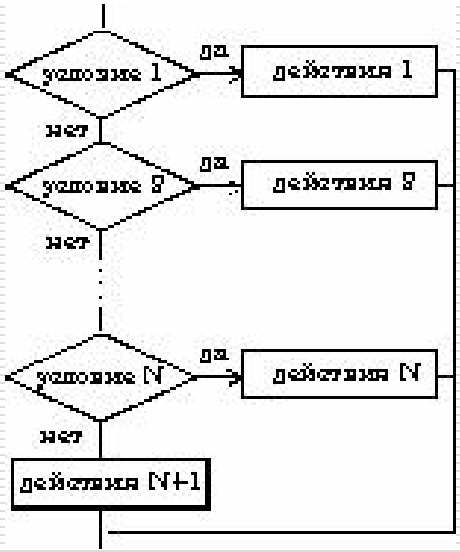
*при условии 2: действия 2*

*. . . . . .*

*при условии N: действия N*

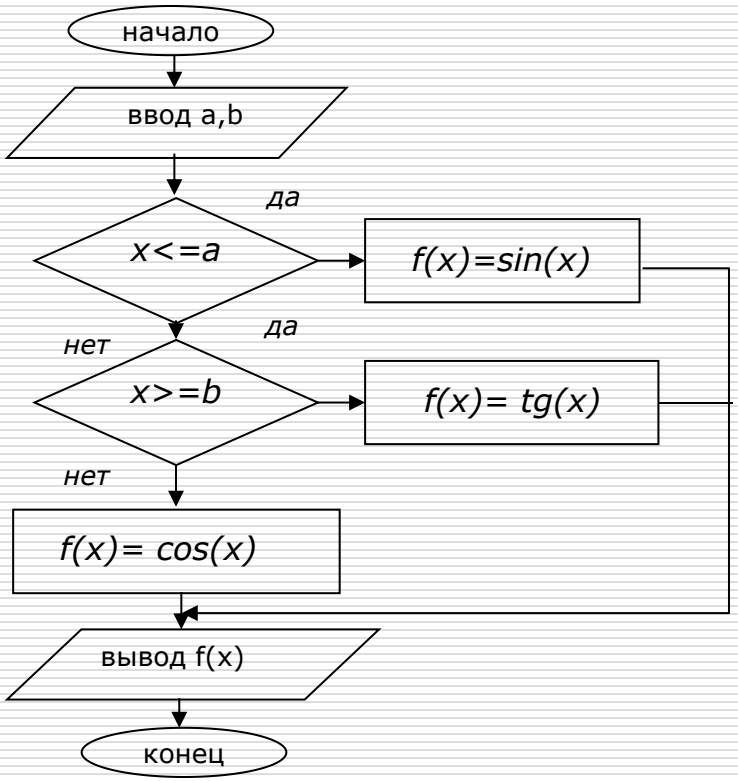
*иначе действия N+1*

*конец выбора*



*Пример. Составить блок-схему алгоритма вычисления функции:*

$$f(x)=\begin{cases}\sin(x), & \text{если } x \le a\\ \cos(x), & \text{если } a < x < b\\ tg(x), & \text{если } x \ge b\end{cases}$$

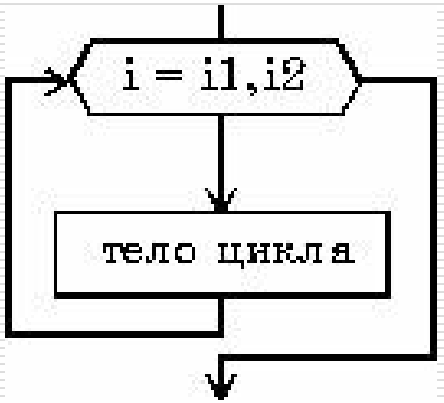


*3. Базовая структура цикл.*

*Обеспечивает многократное выполнение некоторой совокупности действий, которая называется телом цикла.*

*Структура цикл существует в трех основных вариантах:*

- *для;*
- *пока;*
- *делать-пока.*




*а) Цикл типа для.*

*Предписывает выполнять тело цикла для всех значений некоторой переменной (параметра цикла) в заданном диапазоне.*

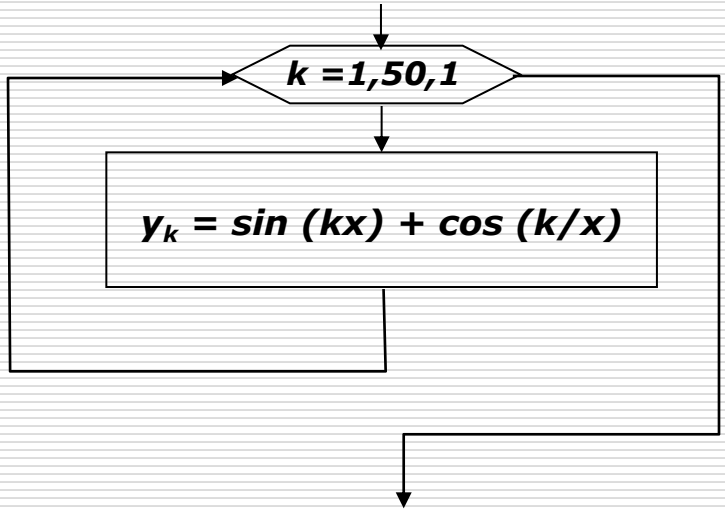
*цикл для i от i1 до i2 шаг i3*

*тело цикла (последовательность действий)*

*конец цикла*

*Пример. Составить блок-схему алгоритма вычисления функции*

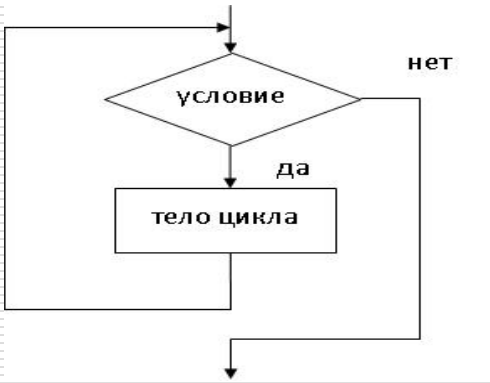
$$yk = \sin(kx) + \cos(k/x),\ k = 1, 2, \ldots, 50$$



*б) Цикл типа пока.*

*Предписывает выполнять тело цикла до тех пор, пока выполняется условие, записанное после слова пока.*

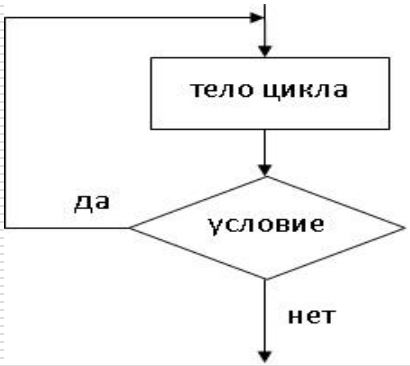
*цикл пока условие*
*тело цикла (последовательность действий)*
*конец цикла*




*в) Цикл типа делать - пока.*

*Предписывает выполнять тело цикла до тех пор, пока выполняется условие, записанное после слова пока. Условие проверяется после выполнения тела цикла.*

*цикл делать*
*тело цикла (последовательность действий)*
*пока условие*
*конец цикла*



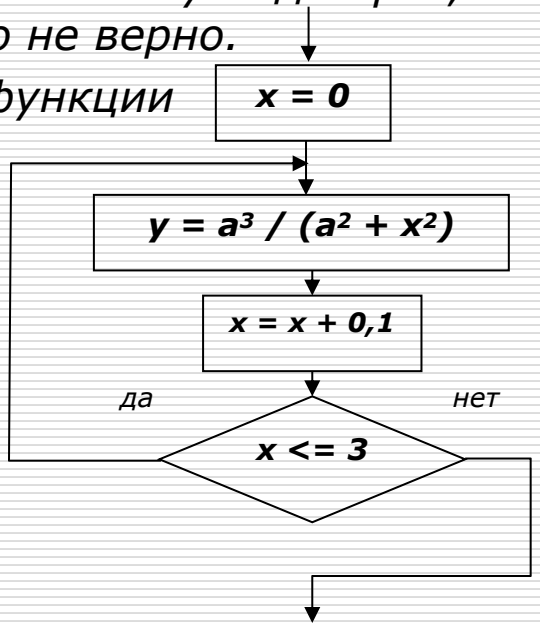
*Заметим, что циклы для и пока называют также циклами с предпроверкой условия а цикл делать – пока называют циклом с постпроверкой условия. Иными словами, тела циклов для и пока могут не выполниться ни разу, если условие окончания цикла изначально не верно. Тело цикла делать – пока выполнится как минимум один раз, даже если условие окончания цикла изначально не верно.*

*Пример. Составить блок-схему вычисления функции*

$y = a^3 / (a^2 + x^2)$

*при x изменяющимся от x = 0 до x = 3*

*с шагом dx = 0,1*



*Итерационные циклы. Особенностью итерационного цикла является то, что число повторений операторов тела цикла заранее неизвестно. Для его организации используется цикл типа пока. Выход из итерационного цикла осуществляется в случае выполнения заданного условия.*

*На каждом шаге вычислений происходит последовательное приближение и проверка условия достижения искомого результата.*

*Пример. Составить алгоритм вычисления суммы ряда*

$$S = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots (-1)^{i-1}\frac{x^i}{i} + \ldots$$

*с заданной точностью (для данного знакочередующегося степенного ряда требуемая точность будет достигнута, когда очередное слагаемое станет по абсолютной величине меньше $\varepsilon$).*

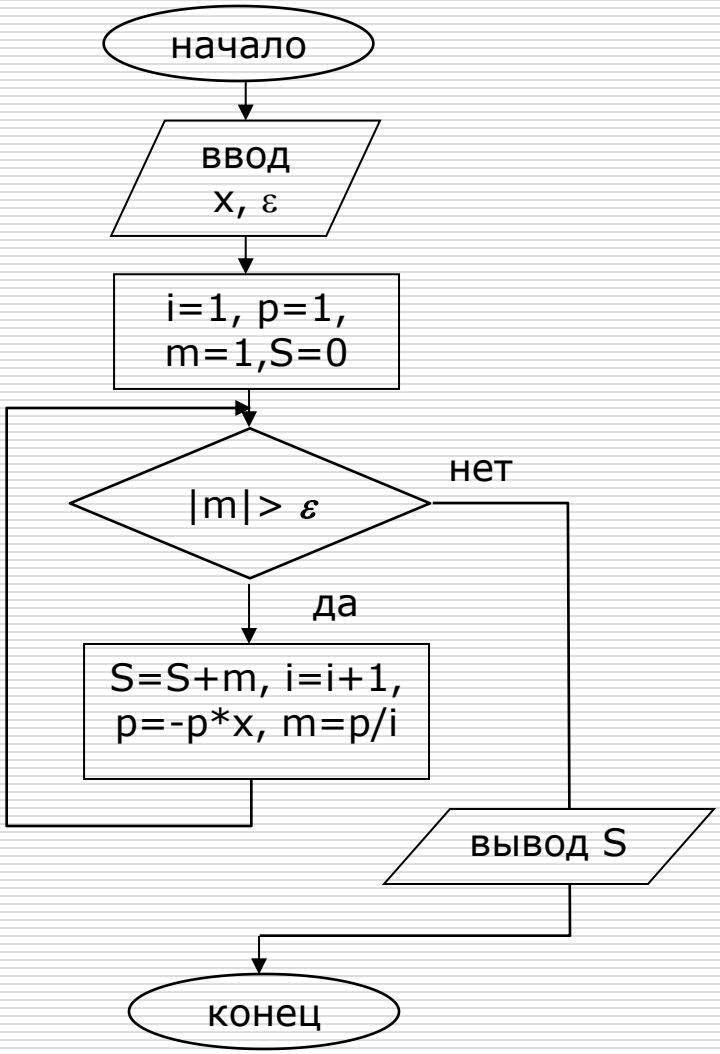
*Решая эту задачу "в лоб" путем вычисления на каждом i-ом шаге частичной суммы:*

*S=S+(-1)^(i-1)*x^i/i*

*мы получим очень неэффективный алгоритм, требующий выполнения большого числа операций. Лучше организовать вычисления следующим образом: если обозначить числитель какого-либо слагаемого буквой p, то у следующего слагаемого числитель будет равен -p*x (знак минус обеспечивает чередование знаков слагаемых), а само слагаемое m будет равно p/i, где i - номер слагаемого.*

*Алгоритм, в состав которого входит итерационный цикл, называется итерационным.*

*Алгоритм, в состав которого входит итерационный цикл, называется итерационным алгоритмом. Итерационные алгоритмы используются при реализации итерационных численных методов. В итерационных алгоритмах необходимо обеспечить сходимость итерационного процесса. В противном случае произойдет зацикливание алгоритма, т.е. не будет выполняться основное свойство алгоритма - результативность.*

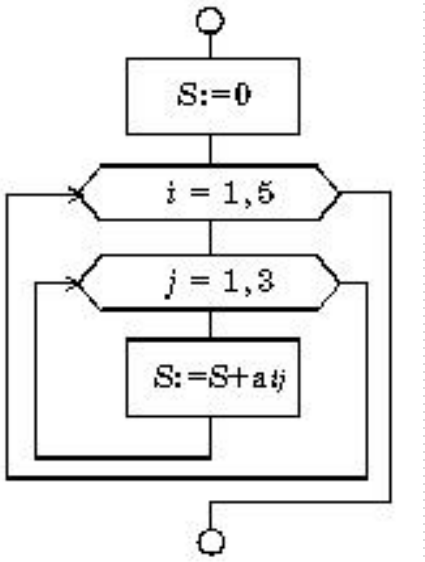

начало
ввод
x, ε
i=1, p=1,
m=1,S=0
|m|> ε
нет
да
S=S+m, i=i+1,
p=-p*x, m=p/i
вывод S
конец

*Вложенные циклы.*

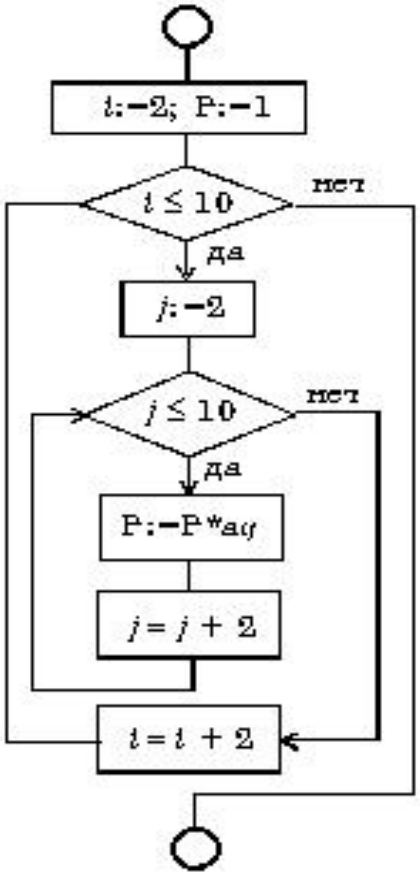
*Возможны случаи, когда внутри тела цикла необходимо повторять некоторую последовательность операторов, т. е. организовать внутренний цикл. Такая структура получила название цикла в цикле или вложенных циклов. Глубина вложения циклов (то есть количество вложенных друг в друга циклов) может быть различной.*

*При использовании такой структуры для экономии машинного времени необходимо выносить из внутреннего цикла во внешний все операторы, которые не зависят от параметра внутреннего цикла.*

*Пример вложенных циклов для. Вычислить сумму элементов заданной матрицы A(5,3).*



*Пример вложенных циклов пока.*
*Вычислить произведение тех элементов заданной матрицы A(10,10), которые расположены на пересечении четных строк и четных столбцов.*



# *3. Представление данных в памяти, типы данных, идентификаторы, переменные, массивы (2 час)*

*3.1 Программный способ записи алгоритмов.*

*Алгоритм, предназначенный для исполнения на компьютере, должен быть записан на "понятном" ему языке. И здесь на первый план выдвигается необходимость точной записи команд, не оставляющей места для произвольного их толкования.*

*Следовательно, язык для записи алгоритмов должен быть формализован. Такой язык принято называть языком программирования, а запись алгоритма на этом языке — программой для компьютера.*

*В настоящее время в мире существует несколько сотен реально используемых языков программирования. Для каждого есть своя область применения.*

*Любой алгоритм есть последовательность предписаний, выполнив которые можно за конечное число шагов перейти от исходных данных к результату.*

*В зависимости от степени детализации предписаний обычно определяется уровень языка программирования — чем меньше детализация, тем выше уровень языка.*

*По этому критерию можно выделить следующие уровни языков программирования:*

- *машинные;*
- *машинно-ориентированные (ассемблеры);*
- *машинно-независимые (языки высокого уровня).*

*Машинные языки и машинно-ориентированные языки — это языки низкого уровня, требующие указания мелких деталей процесса обработки данных.*

*Языки высокого уровня делятся на:*

*алгоритмические (Basic, Pascal, C и др.), которые предназначены для однозначного описания алгоритмов;*

*логические (Prolog, Lisp и др.), которые ориентированы не на разработку алгоритма решения задачи, а на систематическое и формализованное описание задачи с тем, чтобы решение следовало из составленного описания.*

*объектно-ориентированные (Object Pascal, C++, Java и др.), в основе которых лежит понятие объекта, сочетающего в себе данные и действия над нами. Программа на объектно-ориентированном языке, решая некоторую задачу, по сути описывает часть мира, относящуюся к этой задаче. Описание действительности в форме системы взаимодействующих объектов естественнее, чем в форме взаимодействующих процедур*.

*Алгоритмический язык (как и любой другой язык) образуют три его составляющие: алфавит, синтаксис и семантика.*

*Алфавит — это фиксированный для данного языка набор основных символов, т.е. "букв алфавита", из которых должен состоять любой текст на этом языке — никакие другие символы в тексте не допускаются. Из символов алфавита формируются лексемы языка:*

- *константы;*
- *идентификаторы;*
- *знаки операций;*
- *ключевые (служебные, иначе зарезервированные) слова;*
- *разделители (знаки пунктуации).*

*Синтаксис — это правила построения фраз, позволяющие определить, правильно или неправильно написана та или иная фраза. Точнее говоря, синтаксис языка представляет собой набор правил, устанавливающих, какие комбинации символов являются осмысленными предложениями на этом языке.*

*Семантика определяет смысловое значение предложений языка. Являясь системой правил истолкования отдельных языковых конструкций, семантика устанавливает, какие последовательности действий описываются теми или иными фразами языка и, в конечном итоге, какой алгоритм определен данным текстом на алгоритмическом языке.*

*Понятие языка определяется во взаимодействии синтаксических и семантических правил. Синтаксические правила показывают, как образуется данное понятие из других понятий и букв алфавита, а семантические правила определяют свойства данного понятия*

*Основными понятиями в алгоритмических языках обычно являются данные, имена, операции и выражения, операторы.*

*3.2 Представление данных в памяти.*

*Данные — величины, обрабатываемые программой. Имеется три основных вида данных: константы, переменные и массивы.*

*Константы — это данные, которые зафиксированы в тексте программы и не изменяются в процессе ее выполнения. Константы представляются в виде лексем, изображающих фиксированные числовые, логические, символьные или строковые значения.*

*Числовые константы могут быть целыми, вещественными (с фиксированной или плавающей точкой) и перечислимыми.*

*Целые константы могут быть десятичными, восьмеричными и шестнадцатиричными. Десятичная целая константа определена как последовательность десятичных чисел, начинающаяся не с нуля, если это не число нуль. Восьмеричные константы в Visual Basic for Application начинаются с префикса &O и содержат числа от 0 до 7. Шестнадцатиричные числа начинаются с префикса &H и содержат числа от 0 до 9 и латинские буквы от A до F.*

*Примеры целых констант: 123, &O247, &H1F.*

*Вещественные константы записываются в десятичной системе счисления и в общем случае содержат целую часть (десятичная целая константа), десятичную точку, дробную часть (десятичная целая константа), признак (символ) экспоненты E и показатель десятичной степени (десятичная целая константа, возможно со знаком).*

*Примеры вещественных констант: 123.456, 3.402823E38.*

*Перечислимые константы – это набор обычных целочисленных констант. Перечисляемый набор может содержать конечный набор уникальных целых значений, каждое из которых имеет особый смысл в текущем контексте. Перечисляемые наборы являются удобным инструментом, обеспечивающим выбор из ограниченного набора параметров. Например, если пользователь должен выбрать цвет из списка, то можно установить соответствие: черный = 0, белый = 1 и т.д.*

*Логические (булевы) константы могут иметь лишь одно из двух значений: да (истина, TRUE), нет (ложь, FALSE).*

*Символьные и строковые константы. В отличие от большинства языков программирования, где существуют отдельно символьные (содержащие один символ алфавита) и строковые (массив символов) константы, в VBA существуют только строковые, имеющие два типа значений:*

- *Строки переменной длины, которые могут содержать до приблизительно 2 миллиардов (2^31) символов.*
- *Строки постоянной длины, которые могут содержать от 1 до приблизительно (2^16) символов.*

*Примеры строковых констант: "abcde", "информатика", "".*

*В зависимости от значения константы по разному представляются в памяти ПК. Целые представляют последовательный набор фиксированного количества байтов, а вещественные имеют другую форму внутреннего представления, обусловленную применением арифметики с плавающей точкой. В общем случае форма представления данных в памяти ПК определяется типом данных.*

*3.3 Типы данных.*

| ***Тип данных*** | ***Размер*** | ***Диапазон значений*** |
|---|---|---|
| ***Byte*** *(байт)* | 1 байт | От 0 до 255. |
| ***Boolean*** *(логический)* | 2 байт | True или False. |
| ***Integer*** *(целое)* | 2 байт | От –32 768 до 32 767. |
| ***Long*** *(длинное целое)* | 4 байт | От -2 147 483 648 до 2 147 483 647. |
| ***Single*** *(с плавающей точкой обычной точности)* | 4 байт | От -3,402823E38 до -1,401298E-45 для отрицательных значений; от 1,401298E-45 до 3,402823E38 для положительных значений. |
| ***Double*** *(с плавающей точкой двойной точности)* | 8 байт | От -1,7976931348623 2E308 до -4,940656458412 47E-324 для отрицательных значений; от 4,940656458412 47E-324 до 1,7976931348623 2E308 для положительных значений. |
| ***Currency*** *(денежный)* | 8 байт | От –922 337 203 685 477,5808 до 922 337 203 685 477,5807. |

| ***Тип данных*** | ***Размер*** | ***Диапазон значений*** |
|---|---|---|
| ***Decimal*** *(масштабируемое целое)* | 14 байт | +/-79 228 162 514 264 337 593 543 950 335 без дробной части; +/-7,9228162514264337593543950335 с 28 знаками справа от запятой; минимальное ненулевое значение имеет вид +/-0,0000000000000000000000000001. |
| ***Date*** *(даты и время)* | 8 байт | От 1 января 100 г. до 31 декабря 9999 г. |
| ***Object*** *(объект)* | 4 байт | Любой указатель объекта. |
| ***String*** *(строка переменной длины)* | 10 байт + длина строки | От 0 до приблизительно 2 миллиардов. |
| ***String*** *(строка постоянной длины)* | Длина строки | От 1 до приблизительно 65 400. |
| ***Variant*** *(числовые подтипы)* | 16 байт | Любое числовое значение вплоть до границ диапазона для типа Double. |
| ***Variant*** *(строковые подтипы)* | 22 байт + длина строки | Как для строки (String) переменной длины. |
| *Тип данных, определяемый пользователем (с помощью ключевого слова* ***Type****)* | Объем определяется элементами | Диапазон каждого элемента определяется его типом данных. |

***Byte –*** *Массивы данного типа служит для хранения двоичных данных, например, изображений. Использование данного типа предохраняет двоичные данные во время преобразования формата.*

***Boolean –*** *для хранения логических (булевых) значений. По умолчанию значением булевской переменной является* ***False****- ложь.*

***Currency -*** *для хранения чисел с дробной частью до четырех цифр и целой частью до 15 цифр, то есть данных с фиксированной десятичной точкой, удобных для денежных вычислений. Числа с плавающей десятичной точкой (Single, Double) имеют больший диапазон значений, но могут приводить к ошибкам округления.*

***Decimal -*** *в версии 5.0 поддерживается использование типа данных Decimal только в пределах типа Variant, т.е. невозможно описать переменную с типом Decimal. Пользователь, однако, имеет возможность создать переменную типа Variant с подтипом Decimal с помощью функции CDec.*

***Object –*** *поскольку VBA является объектно-ориентированным языком, в нем можно манипулировать различными объектами, адреса расположения которых в памяти (указатели) имеют этот тип.*

***String –*** *по умолчанию данные строкового типа имеют переменную длину и могут удлиняться или укорачиваться. Однако такие строки занимают на 10 байт памяти больше, поэтому можно объявить строки фиксированной длины, явно указав количество символов. Если количество символов будет меньше объявленного, то свободные места заполняются пробелами, при попытке занесения большего количества символов лишние отбрасываются.*

***Variant –*** *может быть использован для хранения данных всех базовых типов без выполнения преобразования (приведения) типов. Применение данного типа позволяет выполнять операции, не обращая внимание на тип данных, которые они содержат. Удобен для объявления переменных, тип которых заранее неизвестен. Переменные этого типа могут содержать специальные значения: Empty, Null, Error.*

## *3.4 Идентификаторы, переменные, массивы.*

*Имена (идентификаторы) — употребляются для обозначения объектов программы (переменных, массивов, процедур и др.). В VBA имена констант, переменных и процедур должны удовлетворять следующим требованиям:*

- *должны начинаться с буквы;*
- *не могут содержать точки и символов объявления типа;*
- *не могут быть длиннее 255 символов. Длина имен объектов не должна превышать 40 символов.*
- *не могут быть ключевыми словами (именами операций, операторов, встроенных функций).*

*Переменные представляют собой зарезервированное место в памяти ПК для хранения значения.*

*Переменные обозначаются именами – словами, используемыми для ссылки на значение, которое содержит переменная, и характеризуются типом, определяющим вид данных, которые можно хранить в переменной.*

*Могут изменять свои значения в ходе выполнения программы. По умолчанию имеют тип данных Variant, если в модуле отсутствует инструкция* ***DefТип****.*

*Для явного указания типа переменной можно в конце ее имени указать символ описания типа:*

| | | |
|---|---|---|
| *Currency - @* | *Long - &* | *Integer - %* |
| *Double - #* | *Single - !* | *String - $* |

*или символ объявления переменных.*

*Объявить переменную – значит заранее сообщить программе о ее существовании. Объявление переменной производится специальным оператором. Одновременно с объявлением переменной после ее имени можно записать ключевое слово As, после которого задается тип переменной. Например: Dim a1 as Integer, b1 as Single*

*Инструкции* ***DefТип*** *используются на уровне модуля (т.е. их нельзя использовать внутри процедур) для задания типа данных, используемого по умолчанию для переменных, имена которых начинаются с соответствующих символов.*

***DefBool*** *диапазонБукв[, диапазонБукв] . . .*

***DefByte*** *диапазонБукв[, диапазонБукв] . . .*

***DefInt*** *диапазонБукв[, диапазонБукв] . . .*

***DefLng*** *диапазонБукв[, диапазонБукв] . . .*

***DefCur*** *диапазонБукв[, диапазонБукв] . . .*

***DefSng*** *диапазонБукв[, диапазонБукв] . . .*

***DefDbl*** *диапазонБукв[, диапазонБукв] . . .*

***DefDec*** *диапазонБукв[, диапазонБукв] . . .*

***DefDate*** *диапазонБукв[, диапазонБукв] . . .*

***DefStr*** *диапазонБукв[, диапазонБукв] . . .*

***DefObj*** *диапазонБукв[, диапазонБукв] . . .*

***DefVar*** *диапазонБукв[, диапазонБукв] . . .*

*Обязательный аргумент диапазонБукв имеет следующий синтаксис:* ***буква_1[-буква_2]***

*Аргументы буква_1 и буква_2 указывают диапазон имен, для которых задается тип данных по умолчанию. Каждый аргумент представляет первую букву имени переменной и может быть любой буквой алфавита. Регистр символов в аргументе диапазонБукв не существенен.*

*Пример. DefInt A-K*

*Константы – имена, идентифицирующие некоторые неизменяемые числовые значения или строки текста. В отличие от переменных их нельзя изменить или назначить им новые значения.*

*В VBA различают внутренние или системные константы, которые имеют префикс vb, например vbOK, и символические или определяемые пользователем с помощью ключевого слова Const константы. Имена констант имеют те же ограничения, что и имена переменных, тип данных, хранящихся в константе, задается так же, как для переменных.*

*Массивы — последовательности логически связанных элементов одного типа, которым присвоено одно имя. Положение элемента в массиве однозначно определяется его индексами (одним, в случае одномерного массива, или несколькими, если массив многомерный). Имена массивов имеют те же ограничения, что и имена переменных, тип данных, хранящихся в массиве, задается так же, как для переменных. Однако если массив имеет тип Variant, его отдельные элементы могут содержать данные разных типов.*

*В VBA существуют два вида массивов: массив фиксированного размера и динамический массив, размер которого может изменяться во время выполнения программы с помощью специальных операторов.*

*При объявлении массива фиксированной длины за его именем в круглых скобках задаются через запятую верхние границы каждой размерности, не превышающие максимального значения типа Long. По умолчанию нижняя граница равна 0, но ее можно задать явно с помощью ключевого слова* ***To:***

***Lines1(100) As String***

***Lines2 (100 To 120) As String***

*При объявлении динамического массива список его размерностей оставляют пустым, а затем с помощью специального оператора назначают действительное число размерностей и элементов массива.*

*Примечание. В VBA массивы любых типов данных требуют 20 байт памяти плюс 4 байт на каждую размерность массива плюс число байт, требуемых для хранения данных. Объем памяти, требуемый для сохранения данных, рассчитывается как произведение числа элементов на размер элемента. Например, данные в одномерном массиве, который содержит четыре элемента типа Integer, требующих по 2 байт на элемент, занимают 8 байт. Вместе с 20 байт на массив и 4 байт на размерность общий требуемый объем составляет 32 байт.*

*Значение типа Variant, содержащее массив, требует 12 байт в дополнение к объему, требуемому массивом.*

*Структуры – в VBA нет понятия структуры, но есть определяемый пользователем с помощью инструкции **Type** тип данных. Типы данных, определяемые пользователем, могут содержать один или несколько элементов любого типа данных, массивы или ранее определенные пользователем типы.*

*Например:*

```
Type MyType
MyName As String          ' Имя записывается в строковую переменную.
MyBirthDate As Date       ' День рождения записывается в переменную даты.
MySex As Integer          ' Пол записывается в целую переменную
End Type                  ' (0 для женщины, 1 для мужчины).
```

## *3.5 Операции, выражения, операторы.*

*Операции. В VBA существуют следующие типы операций:*

*арифметические операции, используемые для выполнения математических вычислений:* ***^, *, /, \, Mod, +, -.***

*Здесь* ****** *- Возвращает результат целого деления двух чисел;*

***Mod -*** *Возвращает остаток при целом делении двух чисел (значение по модулю). Например:*

*a=49\5* *b=49Mod5*

*c=49.5\3.6* *d=37.3Mod3.2*

*операции сравнения (отношения), используемые для выполнения операций сравнения:*

***< , > , <=, >= , = , <> ;***

*операция конкатенации (&)символьных значений друг с другом с образованием одной длинной строки.*

*логические операции, используемые для выполнения логических операций.*

*Рассмотрим логические операции.*

*And - Возвращает результат конъюнкции (логического И) для двух выражений с операциями сравнения, либо выполняет поразрядное сравнение двух числовых выражений.*

*Or - Выполняет операцию логического ИЛИ (сложения) для двух выражений.*

*Eqv - Используется для проверки логической эквивалентности двух выражений с операциями сравнения, либо выполняет поразрядное сравнение двух числовых выражений.*

| Операнд1 | Операнд2 | Результат |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| Операнд1 | Операнд2 | Результат |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| Операнд1 | Операнд2 | Результат |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Imp - Выполняет операцию логической импликации для двух выражений с операциями сравнения, либо выполняет поразрядное сравнение двух числовых выражений.*

| Операнд1 | Операнд2 | Результат |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Xor - Выполняет операцию исключающего ИЛИ для двух выражений.*

| Операнд1 | Операнд2 | Результат |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Not - Выполняет над выражением операцию логического отрицания, а также поразрядное изменение значений каждого разряда переменной.*

| Операнд | Результат |
|---|---|
| 0 | 1 |
| 1 | 0 |

*Выражения — предназначаются для выполнения необходимых вычислений, состоят из констант, переменных, функций (например, exp(x)), объединенных знаками операций.*

*Выражения записываются в виде* ***линейных последовательностей символов*** *(без подстрочных и надстрочных символов, "многоэтажных" дробей и т.д.), что позволяет вводить их в компьютер, последовательно нажимая на соответствующие клавиши клавиатуры.*

*Различают выражения арифметические, логические и строковые.*

*Арифметические выражения служат для определения одного числового значения.*

*Например, (1+sin(x))/2.*

*Значение этого выражения при x=0 равно 0.5, а при x=pi/2 - единице.*

*Логические выражения описывают некоторые условия, которые могут удовлетворяться или не удовлетворяться. Таким образом, логическое выражение может принимать только два значения — "истина" или "ложь" (да или нет).*

*Рассмотрим в качестве примера логическое выражение:*

$$x*x + y*y < r*r$$

*определяющее принадлежность точки с координатами (x,y) внутренней области круга радиусом r с центром в начале координат. При x=1, y=1, r=2 значение этого выражения — "истина", а при x=2, y=2, r=1 — "ложь".*

*Значения строковых выражений — тексты. В них могут входить литерные константы, литерные переменные и литерные функции, разделенные знаком операции сцепки.*

*Например, A & B означает присоединение строки B к концу строки A.*

*Если A = "куст ", а B = "зеленый", то значение выражения A&B есть "куст зеленый".*

*Операторы (команды).*

*Оператор — это наиболее крупное и содержательное понятие языка: каждый оператор представляет собой законченную фразу языка и определяет некоторый вполне законченный этап обработки данных.*

*В состав операторов входят:*

- *ключевые слова;*
- *данные;*
- *выражения и т.д.*

*Операторы подразделяются на исполняемые и неисполняемые. Неисполняемые операторы предназначены для описания данных и структуры программы, а исполняемые — для выполнения различных действий (например, оператор присваивания, операторы ввода и вывода, условный оператор, операторы цикла, оператор процедуры и др.).*

# *4. Структура программ на VBA, операторы описания, операторы присваивания (2 час)*

## *4.1 Структура программ на VBA.*

*Программы на VBA хранятся в проектах. Проект содержит модули различных типов, а модули включают различные процедуры.*

*Проект может содержать несколько модулей. Имеется три типа модулей:*

*Стандартные модули – это модули, в которых можно описать доступные во всем проекте процедуры.*

*Модули класса содержат описание объекта, который является членом класса. Процедуры, написанные в модуле класса, используются только в этом модуле. Среди модулей класса выделяют модули форм и отчетов, которые связаны с конкретной формой или отчетом.*

*Модули форм и отчетов часто содержит процедуры обработки событий, которые срабатывают в ответ на событие в форме или отчете. Процедуры обработки событий используются для управления поведением форм и отчетов и их реакцией на действия пользователя типа щелчка мыши на кнопке.*

*Модули содержат описания и процедуры – наборы описаний и инструкций, сгруппированных для выполнения. Существует три типа процедур:*

*процедура Sub – набор команд, с помощью которого можно решить определенную задачу. При ее запуске выполняются команды процедуры, а затем управление передается в приложение пакета MS Office или процедуру, которая вызвала данную процедуру.*

*процедура Function (функция) также представляет собой набор команд, который решает определенную задачу. Различие заключается в том, что такие процедуры обязательно возвращают значение, тип которого можно описать при создании функции.*

*процедура Property используется для ссылки на свойство объекта. Данный тип процедур применяется для установки или получения значения пользовательских свойств форм и модулей.*

*Для создания модуля в любом приложении MS Office необходимо выбрать команду меню Сервис/Макрос/Редактор Visual Basic. В окне «Проект» необходимо щелкнуть правой кнопкой мыши на любом элементе либо в окне редактора выбрать команду меню Insert (Вставка), а далее тип модуля. При выборе формы (Userform) для перехода к ее модулю используется команда Вид/Программа или кнопка «Программа» в окне «Проект».*

*Самое начало модуля называется общей областью, в которой располагаются общие описания, например, типа данных, используемого по умолчанию (DefТип), инструкция Option Explicit, требующая явного описания всех используемых в модуле переменных, а также описания общих (глобальных) для всех модулей и для данного модуля переменных.*

## *4.2 Операторы описания.*

*Объявление переменной производится одним из операторов:*

*Dim, Static, Private, Public, за которым следует имя переменной и необязательная часть с ключевым словом As, после которого задается тип переменной, например Dim name [As type].*

*Оператор Public используется только вне модуля, в его общей части и делает описываемую переменную доступной из всех процедур всех модулей проекта.*

*Оператор Private служит для объявления переменной уровня модуля, доступной только процедурам данного модуля. Можно использовать также оператор Dim, но применение Private предпочтительнее как противоположное Public.*

*Переменные могут быть объявлены внутри процедуры операторами Dim или Static. Такие переменные называют также локальными, поскольку доступны только в той процедуре, в которой они объявлены. Данное свойство (область видимости) позволяет использовать одинаковые имена переменных в разных процедурах, не опасаясь конфликтов или случайных изменений значений переменных.*

*Время жизни локальных переменных, объявленных с помощью оператора Dim равно времени работы процедуры и по ее окончании значения таких переменных теряются.*

*Переменные, объявленные с помощью оператора Static сохраняют свои значения в течении всего времени выполнения приложения. При повторном входе в процедуру, где описана такая переменная, ее значение сохраняется.*

*Операторы Public и Private можно применять при описании констант и процедур, что позволяет указать их область видимости. Для процедур возможно также применение оператора Static, что позволяет сделать все переменные в процедуре статическими:*

***Static Function Total (num) as Integer***

*Это приводит к тому, что все локальные переменные в процедуре становятся статическими, независимо от того, как они определены: операторами Static, Dim, Private  или неявным образом.*

## *4.3 Операторы присваивания.*

*Инструкция Let присваивает значение выражения переменной или свойству:*

***[Let] имяПеременной = выражение***

*Явное использование ключевого слова Let зависит от вкуса пользователя, обычно это слово опускают.*

*Значение выражения может быть присвоено переменной, только если оно имеет совместимый с этой переменной тип данных. Невозможно присвоить строковое выражение числовой переменной или числовое выражение строковой переменной. Такая попытка приведет к ошибке во время компиляции.*

*Переменным типа Variant могут присваиваться как строковые, так и числовые выражения. Однако обратное не всегда верно. Любое значение типа Variant, за исключением значения Null, допускает присвоение строковой переменной, но только значение типа Variant, которое может рассматриваться как число, может быть присвоено числовой переменной.*

*Пользуйтесь функцией IsNumeric для определения возможности преобразования значения Variant в числовое значение.*

*Внимание! Присвоение выражения с одним из числовых типов переменной с другим числовым типом данных преобразует значение выражения в тип данных результирующей переменной.*

*Инструкция Let может быть использована для присвоения одной переменной-записи другой, только если обе переменные имеют одинаковый определяемый пользователем тип. Для присвоения переменных-записей различных определяемых пользователем типов используется инструкция LSet. Для присвоения переменным ссылок на на объекты применяется инструкция Set.*

# *5. Понятие макроса, создание, отладка, использование среды для отладки программ*

## *5.1 Понятие макроса и действия с ним*

*Макрос – процедура на внутреннем языке приложения, в которой записаны действия пользователя приложения. В программах Word, Excel и PowerPoint макрос записывается на VBA. В Access макросы создаются с помощью собственного языка макросов, а не записываются автоматически, и к тому же не являются процедурами VBA.*

*Создание макроса*

*Прежде чем записать макрос, требуется продумать свои действия, то есть составить алгоритм. Особое внимание следует уделить таким положениям:*

- *Какие условия должны выполняться при запуске макроса (какой файл должен быть открыт, где должен находиться курсор, в каком режиме должно работать приложение).*
- *Какие действия должен выполнять макрос.*
- *Какие действия необходимо проделать при завершении работы макроса.*

*Чтобы начать запись макроса необходимо:*

- *Активизировать приложение.*
- *Открыть документы, используемые при записи макроса.*
- *Выбрать команду «Сервис/Макрос».*
- *Выбрать команду «Начать запись ». Выводится диалоговое окно «Запись макроса».*
- *Ввести имя записываемого макроса в поле «Имя макроса».*
- *Выбрать в списке «Макрос доступен для» документ, в который требуется поместить макрос.*
- *Нажать ОК, чтобы начать запись.*

*После этого записываются все выполняемые в приложении действия.*

*Чтобы остановить запись макроса необходимо:*

*Нажать на панели инструментов «Остановка записи» кнопку «Остановить запись» или выбрать команду «Сервис/Макрос/Остановить запись».*

*Вновь созданный макрос содержится в документе, который был активным при запуске макроса. При записи документа на диск макрос сохраняется вместе с проектом.*

*Для выполнения записанного макроса необходимо:*

- *Выбрать команду «Сервис/Макрос/Макросы». Выводится диалоговое окно «Макрос».*
- *Выбрать имя требуемого макроса*
- *Нажать кнопку «Выполнить» для запуска макроса.*

*Для редактирования записанного макроса необходимо:*

- *Выбрать команду «Сервис/Макрос/Макросы». Выводится диалоговое окно «Макрос».*
- *Выбрать имя требуемого макроса*
- *Нажать кнопку «Изменить». Макрос выводится в окне редактора Visual Basic for Applications.*

## *5.2 Отладка, использование среды для отладки программ*

*В большинстве систем разработки программ имеются инструменты, с помощью которых можно решить проблемы, возникающие в процессе программирования.*

*В VBA также есть средства, которые позволяют либо исключить ошибки при разработке, либо задать обработку ошибок при выполнении программ.*

***Отладка программ** - это проверка и внесение исправлений в программу при ее разработке. Отладка позволяет идентифицировать ошибки, допущенные при программировании (синтаксические – ошибки в выражениях и именах, и логические – в логике работы программы).*

***Обработка ошибок** – это задание реакции на ошибки, которые возникают при выполнении программы. Их причиной могут быть как ошибки программиста, так и внешние факторы – отсутствие нужных файлов, отказы аппаратуры, неправильные действия пользователя.*

***Типы ошибок.***

*Ошибки в программе делятся на три категории:*

*Ошибки компиляции – возникают, когда компилятор не может интерпретировать введенный текст. Некоторые ошибки компиляции обнаруживаются при вводе, а другие – перед выполнением программы. Такие ошибки легко определить и исправить, поскольку VBA выявляет их автоматически, а сами ошибки очевидны.*

*Примечание. VBA автоматически компилирует программу каждый раз при запуске на выполнение после внесения изменений. Можно также запустить компиляцию командой Отладка/Компилировать.*

*Ошибки выполнения - возникают при выполнении программы после успешной компиляции. Их причиной обычно является отсутствие данных или неправильная информация, введенная пользователем. Такие ошибки идентифицируются VBA с указанием инструкции, при выполнении которой произошла ошибка. Для исправления таких ошибок обычно приходится выводить значения переменных или другие данные, которые влияют на успешное выполнение программы.*

*Логические ошибки - трудно заметить и устранить. Они не приводят к прекращению компиляции или выполнения, однако являются причиной того, что программа не выдает желаемых результатов. Выявление таких ошибок производят путем тщательной проверки с помощью средств отладки VBA.*

***Средства отладки.***

*В VBA имеется большое количество средств, предназначенных для отладки программ.*

*К ним относятся: использование оператора Option Explicit, пошаговое выполнение программы, работа в режиме прерывания, использование точек останова, вывод значений переменных.*

*Использование Option Explicit.*

*Данный оператор описания требует явного задания переменных в программах. При его использовании возникает ошибка компиляции при неправильном написании имени переменной или использовании неописанной переменной.*

*Кроме того, явное описание переменных позволяет обойтись без использования типа данных Variant и связанных с его использованием ошибок при неявном приведении типов данных.*

*Пошаговое выполнение программы.*

*Этот режим служит для локализации ошибок в теле программы. Для его запуска используются команды меню, клавиатура или панель инструментов «Отладка», отображаемая командой «Вид/Панели инструментов/Отладка». В меню и на панели инструментов имеются четыре команды (кнопки) для выполнения программы в пошаговом режиме.*

- *Команда «Отладка/Шаг с заходом – Debug/Step into» (клавиша F8) позволяет выполнить одну строку программы и перейти к следующей. Если следующая строка – вызов процедуры, то происходит переход к первому выполняемому оператору этой процедуры.*
- *Команда «Отладка/Шаг с обходом- Debug/Step Over» (клавиши Shift+F8) также выполняет одну строку программы, но если строкой является вызов процедуры, то она выполняется как одна инструкция. Данная команда используется, если известно, что эта процедура работает правильно.*
- *Команда «Отладка/Шаг с выходом - Debug/Step Out» (клавиши Ctrl+Shift+F8) заканчивает выполнение текущей процедуры и останавливается на следующей после вызова текущей процедуры инструкции в вызывающей подпрограмме.*
- *Команда «Отладка/Выполнить до текущей позиции – Debug/Run To Cursor» (клавиши Ctrl+F8) выполняет программу от текущей до выбранной инструкции. Перед выбором данной команды требуется установить курсор в окне модуля на требуемую позицию.*

*Работа в режиме прерывания.*

*Переход в данный режим выполняется:*

- *При нажатии кнопки «Отладка» в окне сообщения об ошибке выполнения.*
- *При прерывании работы программы нажатием клавиш «Ctrl+Break». Текущая строка программы выделяется в окне модуля.*
- *По достижении точки останова.*
- *По достижении оператора «Stop».*
- *При пошаговом выполнении программы.*

*В режиме прерывания можно:*

- *Вывести значение переменной.*
- *Вычислить выражение в окне отладки.*
- *Сбросить программу.*
- *Выполнить программу в пошаговом режиме.*
- *Продолжить выполнение программы.*

*Для выхода из режима прерывания используется команда «Запуск/Сброс – Run/Reset».*

*Использование точек останова.*

*Точка останова – это строка в процедуре, на которой приостанавливается выполнение программы.*

*Все команды, находящиеся выше точки останова, выполняются с обычной скоростью, а по достижении контрольной точки программа переходит в режим прерывания. Затем можно отлаживать процедуру в пошаговом режиме, либо использовать различные способы вывода значений переменных.*

*Кроме того, имеется возможность остановить выполнение или сбросить процедуру командами меню «Запуск - Run» или кнопками панели инструментов «Отладка-Debug». В одном проекте можно задать несколько точек останова, причем в различных процедурах.*

*Чтобы установить или снять точку останова, используется команда «Отладка/Точка останова – Debug/Toggle Breakpoint» или клавиша «F9», либо кнопка «Точка останова» панели инструментов «Отладка».*

*Можно также установить или снять точку останова, щелкнув левой кнопкой мыши на полосе индикатора против требуемой строки. Точка останова отмечается коричневой жирной точкой на полосе индикатора, а сама строка выделяется коричневым цветом.*

*Чтобы быстро удалить все точки останова открытого проекта, используется команда «Отладка/Снять все точки останова – Debug/Clear All Breakpoints» или комбинация клавиш «Ctrl+Shift+F9».*

*Для данной команды не предусмотрена кнопка на панели инструментов «Отладка».*

*Вывод значений переменных.*

*При наличии тестового примера вывод значений переменных позволяет сравнить ожидаемые и полученные значения переменных.*

*Для отображения значений переменных в режиме прерывания необходимо:*

- *При установленном флажке «Подсказки значений переменных» в окне «Сервис/Параметры» достаточно переместить указатель мыши на требуемую переменную для отображения имени и значения переменной во всплывающей подсказке.*
- *Выбрать команду «Отладка/Контрольное значение- Debug/Ouick Watch» (клавиши Shift+F9) для вывода диалогового окна «Контрольное значение». При этом курсор должен находиться возле переменной, значение которой надо контролировать. В окне «Контрольное значение» отображается контекст (имя модуля и процедуры), выделенное выражение (переменная) и кнопки «Добавить» и «Отмена».*
- *При нажатии кнопки «Добавить» откроется окно «Контрольные значения», содержащее имена переменных (выражения), их значения, тип данных и контекст.*
- *Для добавления других контрольных значений используется команда «Отладка/Добавить контрольное значение».*

- *Выбрать команду «Вид/Окно локальных переменных – View/Locals Window». Откроется окно «Локальные переменные», в котором в режиме прерывания отображаются имена, значения и типы всех переменных модуля.*
- *Выбрать команду «Вид/Окно отладки – View/Debug Window». В нем немедленно выполняется введенная в него инструкция, обычно операция отображения значения выражения вида:*

  *Print имя,*

  *или операция присваивания значения переменной. Выполняемая программа также может выводить информацию в это окно с помощью выражения* ***Debug.Print*** *имя переменной.*

*Для отображения значений переменных в режиме нормальной работы необходимо ввести в тело программы вызов функции*

***MsgBox (сообщение, [кнопки, заголовок]).***

*Эта функция отображает диалоговое окно, содержащее сообщение длиной до 1024 символов, в которое с помощью операции конкатенации можно включить значение переменных, а также (необязательно) кнопки для реакции на отображения окна (по умолчанию только кнопка ОК) и заголовок окна (строковое выражение).*

*Пример:*

***MsgBox "Значение val=" & val***

*Для вывода значений выражений и переменных в активный документ удобно создавать пользовательские процедуры. Например, для вывода значения в ячейку рабочего листа «Лист1» активной книги Excel можно записать в модуле и вызывать процедуру вида:*

```
Sub out(name As String, val As Variant)
Лист1.Range(name).Value = val
End Sub
```

*Здесь **name** – координаты ячейки, записанные в кавычках, а **val** – имя выводимой переменной.*

*Для ввода значений переменных в программу применяют функцию:* ***InputBox(сообщение[, заголовок] [, значение по умолчанию] [, координата x] [, координата y])***.

*Эта функция отображает диалоговое окно, содержащее окно ввода, кнопки ОК и Отмена, сообщение (подсказку для ввода) и (необязательно) заголовок окна, значение, вводимое по умолчанию, координаты окна по горизонтали и вертикали в твиках.*

*Функция InputBox всегда (даже при нажатии кнопки Отмена) возвращает значение строкового типа, поэтому вызов ее должен иметь вид:*

***name = InputBox("Введи адрес ячейки", "Ввод", “a1”, 100, 200)***

*Для преобразования введенного значения к нужному типу данных используются функции явного приведения типа, такие как:*

***CDbl(выражение), CInt(выражение), CLng(выражение), CSng(выражение), CVar(выражение), CStr(выражение).***

*Для ввода значений переменных из активного документа удобно создавать пользовательские процедуры. Например, для ввода значения из ячейки рабочего листа «Лист1» активной книги Excel можно записать в модуле и вызывать процедуру вида:*

```
Sub read(name As String, val As Variant)
val = Лист1.Range(name).Value
End Sub
```

*Здесь* ***name*** *– координаты ячейки, записанные в кавычках, а* ***val*** *– имя вводимой переменной.*

# *6. Управляющие структуры VBA. If . . . Then, If . . . Then . . . Else, Select Case.*

*Управляющие структуры позволяют управлять последовательностью выполнения программы.*

*Без операторов управления все операторы программы будут выполняться слева направо и сверху вниз. Однако иногда требуется многократно выполнять некоторый набор инструкций автоматически, либо решить задачу по-другому в зависимости от значения переменных или параметров, заданных пользователем во время выполнения. Для этого служат конструкции управления и циклы.*

*VBA поддерживает следующие конструкции принятия решений:*

***If . . . Then***

***If . . . Then . . . Else***

***Select Case***

*6.1 Конструкция If . . . Then*

*Конструкция **If . . . Then** применяется, когда необходимо выполнить один или группу операторов в зависимости от некоторого условия.*

*Синтаксис этой конструкции позволяет задавать ее в одной строке или в нескольких строках программы:*

1) ***If условие Then выражение***
2) ***If условие Then***
   ***выражение***
   ***End If***

*Обычно условие является простым сравнением, но оно может быть любым выражением с вычисляемым значением. Это значение интерпретируется как **False** (Ложь), если оно нулевое, а любое ненулевое рассматривается как **True** (Истина). Если условие истинно, то выполняются все выражения, стоящие после ключевого слова **Then**.*

*Для условного выполнения одного оператора можно использовать как синтаксис для одной строки, так и синтаксис для нескольких строк (блоковую конструкцию).*

*Следующие два оператора эквивалентны:*

***1) If anyDate < Now Then anyDate = Now***

***2) If anyDate < Now Then***

***anyDate = Now***

***End If***

*Заметим, что синтаксис оператора **If . . . Then** для одной строки не использует оператор **End If**. Чтобы выполнить последовательность операторов, если условие истинно, следует использовать блоковую конструкцию **If . . . Then . . . End If**.*

```
If anyDate < Now Then
anyDate = Now
Timer.Enabled = False   ' Запретить таймер.
End If
```

*Если условие ложно, то операторы после ключевого слова **Then** не выполняется, а управление передается на следующую строку (или строку после оператора **End If** в блочной конструкции).*

*6.2 Конструкция If . . . Then . . . Else*

*определяет несколько блоков операторов, один из которых будет выполняться в зависимости от условия:*

***If условие1 Then***
***выражение1***
***ElseIf условие2 Then***
***выражение2***
***. . .***
***Else***
***выражение-n***
***End If***

*При выполнении сначала проверяется условие1. Если оно ложно, VBA проверяет следующее условие2 и т. д., пока не найдет истинного условия. Найдя его, VBA выполняет соответствующий блок операторов и затем передает управление инструкции, следующей за оператором End if. В данную конструкцию можно включить блок оператора Else, который VBA выполняет, если не выполнено ни одно из условий.*

*Рассмотрим пример вычисления функции:*

$$f(x)=\begin{cases}\sin(x), & если \quad x \le a\\ \cos(x), & если \quad a < x < b\\ tg(x), & если \quad x \ge b\end{cases}$$

```
Sub пример1()
Dim a As Single, b As Single, x As Single
Dim f As Double
Call read("A1", a)
Call read("B1", b)
Let x = CSng(InputBox("введи x", "Ввод данных", 0))
If x <= a Then
f = Sin(x)
ElseIf x >= b Then
f = Tan(x)
Else: f = Cos(x)
End If
Call out("C1", f)
End Sub
```

## *6.3 Конструкция Select Case*

*Конструкция* ***Select Case*** *является альтернативой конструкции* ***If . . . Then . . . Else*** *в случае выполнения блока, состоящего из большого набора операторов.*

*Конструкция* ***Select Case*** *предоставляет возможность, похожую на возможность конструкции* ***If . . . Then . . . Else****, но в отличие от нее она делает код более читаемым при наличии нескольких вариантов выбора.*

*Конструкция* ***Select Case*** *работает с единственным проверяемым выражением, которое вычисляется один раз при входе в эту конструкцию. Затем VBA сравнивает полученный результат со значениями, задаваемыми в операторах* ***Case*** *конструкции. Если найдено совпадение, выполняется блок операторов, ассоциированный с оператором* ***Case****.*

***Select Case проверяемое_выражение***
***[Case список_выражений1***
***[блок_операторов1]]***
***[Case список_выражений2***
***[блок_операторов2]]***
***. . .***
***[Case Else***
***[блок_операторов-n]]***
***End Select***

*Каждый список выражений является списком из одного или более значений. Если в одном списке больше одного значения, они отделяются запятыми. Каждый блок операторов содержит несколько операторов или ни одного. Если окажется, что вычисленному значению проверяемого выражения соответствуют значения из нескольких операторов **Case**, то выполняется блок операторов, ассоциированный с первым оператором **Case** из всех найденных соответствий. VBA выполняет блок операторов, ассоциированный с оператором **Case Else** (заметим, что он необязателен), если не найдено ни одного соответствия проверяемого значения выражения и значений из всех списков операторов **Case**.*

*Рассмотрим пример вычисления функции:*

$$f(x) = \begin{cases} \sin x, & \text{если } x = -\pi/2, \\ \cos x, & \text{если } x = 0, \\ tg\ x, & \text{если } x = \pi/2. \end{cases}$$

```
Sub пример2()
Const pi2 as Single = 1.57
Dim x As Single, f As Double
Let x = CSng(InputBox("введи x", "Ввод данных", 0))
Select Case x
Case -pi2
f = Sin(x)
Case 0
f = Cos(x)
Case pi2
f = Tan(x)
Case Else
MsgBox "Неверные исходные данные!"
Exit Sub
End Select
Call out("D1", f)
End Sub
```

# *7. Операторы цикла. Вложенные циклы.*

## *7.1 Операторы цикла.*

*Циклы позволяют выполнить одну или несколько строк кода несколько раз. VBA поддерживает следующие циклы:*

***For…Next*** ***Do… Loop***

***For Each…Next*** ***While….Wend***

*Конструкция* ***For . . . Next.***

*Когда число повторений известно заранее, используют цикл* ***For . . . Next****. В цикле* ***For*** *используется переменная, называемая переменной цикла или счетчиком цикла, которая увеличивается или уменьшается на заданную величину при каждом повторении цикла. Синтаксис этой конструкции следующий:*

***For counter = start To end [Step increment]***

***операторы***

***Next [counter]***

*Параметры counter (счетчик), start (начало цикла), end (конец цикла) и increment (приращение) являются числовыми.*

*Примечание. Параметр increment может быть как положительным, так и отрицательным. Если он положителен, параметр start должен быть меньше или равен параметру end, иначе цикл не будет выполняться. Если параметр increment отрицателен, то параметр start должен быть больше или равен значению параметра end, чтобы выполнялось тело цикла. Если параметр* ***Step*** *не задан, то значение параметра increment по умолчанию равно 1.*

*VBA выполняет цикл For в следующей последовательности:*

*1. Устанавливает значение переменной цикла counter в значение start.*

*2. Сравнивает значение переменной цикла counter и значение параметра end. Если переменная counter больше, VBA завершает выполнение цикла. (Если значение параметра increment отрицательно, то VBA прекращает выполнение цикла при условии, что значение переменной цикла counter меньше значения параметра end.)*

*3. Выполняет операторы тела цикла statements.*

*4. Увеличивает значение переменной цикла counter на 1 или на величину значения параметра increment, если он задан.*

*5. Повторяет шаги со 2 по 4.*

*Рассмотрим пример вычисления значения функции f(t)*
*при заданных a, b, n, если t меняется от a до b с шагом dt=(b-a)/(n-1).*

$$f(t) = \begin{cases} -1, & a \le t \le -1, \\ t, & -1 < t \le 1, \\ 1, & 1 < t \le b. \end{cases}$$

```
Sub пример3()
Dim f() As Single
Dim a!, b!, t! dt!, i%, n%
Call read("a1", a) : Call read("b1", b) : Call read("c1", n)
ReDim f(1 To n - 1)
dt = (b - a) / (n - 1) : t = a
Call out("a2", "i") : Call out("b2", "t") : Call out("c2", "f(t)")
For i = 1 To n - 1
If t <= -1 Then
f(i) = -1
ElseIf t > 1 Then
f(i) = 1
Else : f(i) = t
End If
Call out("a" & (2 + i), i) : Call out("b" & (2 + i), t) : Call out("c" & (2 + i), f(i))
t = t + dt
Next i
End Sub
```

*Конструкция* ***For Each . . . Next***

*Цикл* ***For Each . . . Next*** *похож на цикл* ***For . . . Next****, но он повторяет группу операторов для каждого элемента из набора объектов или из массива, вместо повторения операторов заданное число раз.*

*Особенно полезен, когда неизвестно, сколько элементов содержится в наборе.*

*Синтаксис конструкции цикла* ***For Each . . . Next*** *таков:*

***For Each element In group***

***операторы***

***Next element***

*Следует помнить следующие ограничения при использовании цикла* ***For Each . . . Next****:*

*1. Для наборов параметр element может быть только переменной типа variant, общей переменной типа object или объектом, перечисленным в Object Browser;*

*2. Для массивов параметр element может быть только переменной типа Variant;*

*3. Нельзя использовать цикл* ***For Each . . . Next*** *с массивом, имеющим определенный пользователем тип, так как переменная типа variant не может содержать значение определенного пользователем типа.*

*Пример использования цикла* ***For Each . . . Next***

```
Sub пример4()
Dim TestArray(1 To 3) As String, i As Variant, sum As String
TestArray(1) = "ла"
TestArray(2) = "м"
TestArray(3) = "па"
For Each i In TestArray
sum = sum + i
Next i
MsgBox sum
End Sub
```

*Конструкция* ***Do...Loop***

*Цикл* ***Do*** *применяется для выполнения блока операторов неограниченное число раз. Существует несколько разновидностей конструкции* ***Do . . . Loop****, но каждая из них вычисляет выражение-* ***условие****, чтобы определить момент выхода из цикла. Как и в случае конструкции* ***If . . . Then условие*** *должно быть величиной или выражением, принимающими значение* ***False*** *(нуль) или* ***True*** *(не нуль).*

*В следующей конструкции* ***Do . . . Loop*** *операторы выполняются до тех пор, пока значением условия является* ***True*** *(Истина):*

***Do While условие***

***операторы***

***Loop***

*Выполняя этот цикл, VBA сначала проверяет условие. Если условие ложно (****False****), он пропускает все операторы цикла. Если оно истинно (****True****), VBA выполняет операторы цикла, снова возвращается к оператору* ***Do While*** *и снова проверяет условие.*

*Следовательно, цикл, представленный данной конструкцией, может выполняться любое число раз, пока значением условия является не нуль или* ***True*** *(Истина). Отметим, что операторы тела цикла не выполняются ни разу, если при первой проверке условия оно оказывается ложным (****False****).*

*Рассмотрим пример: Вычислить сумму ряда*

$$S = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots (-1)^{i-1} \frac{x^i}{i} + \ldots$$

*с заданной точностью ε.*

```
Sub пример5()
Dim e As Single, x As Single, s As Single
Dim m As Single, p As Single, i As Integer
Call read("a1", x) : Call read("b1", e)
i = 1: p = x: m = x: s = 0
Call out("a2", "i") : Call out("b2", "m") : Call out("c2", "s")
Do While Abs(m) >= e
s = s + m
p = -p * x
m = p / i
Call out("a" & (2 + i), i) : Call out("b" & (2 + i), Abs(m)) : Call out("c" & (2 + i), s)
i = i + 1
Loop
End Sub
```

*Другая разновидность конструкции* ***Do . . . Loop*** *сначала выполняет операторы тела цикла, а затем проверяет условие после каждого выполнения. Эта разновидность гарантирует, что операторы тела цикла выполнятся по крайней мере один раз:*

***Do***
***операторы***
***Loop While условие***

*Две другие разновидности конструкции цикла аналогичны предыдущим, за исключением того, что цикл выполняется, пока условие ложно (****False****):*

*Цикл не выполняется вообще или выполняется много раз:*

***Do Until условие***
***операторы***
***Loop***

*Цикл выполняется по крайней мере один раз:*

***Do***
***операторы***
***Loop Until условие***

## *7.2 Вложенные циклы.*

*Можно помещать структуры управления внутрь других структур управления (например, блок **If . . . Then** внутрь цикла **For . . . Next**). Говорят, что структура управления, помещенная внутрь другой структуры управления, является вложенной.*

*Глубина вложения управляющих структур в VBA не ограничена. Для улучшения читаемости кода принята практика смещения тела конструкции принятия решения или цикла в программе в случае использования вложенных структур управления.*

*При вложении в цикл одного или несколько других циклов говорят о вложенных циклах, в которых различают внешние (охватывающие) и внутренние (вложенные) циклы.*

*Пример суммирования элементов* ***Aij*** *матрицы* ***A(1 to n, 1 to m)*** *построчно.*

```
Sub пример6()
Dim a() As Single, s() As Single
Dim n As Integer, m As Integer, i As Integer, j As Integer
Call read("a1", n): Call read("b1", m)
ReDim a(1 To n, 1 To m), s(1 To n)
'Чтение матрицы
For i = 1 To n
For j = 1 To m
Call readcell(i + 1, j, a(i, j))
Next j
Next i
'Вычисление и вывод
For i = 1 To n
s(i) = 0
For j = 1 To m
s(i) = s(i) + a(i, j)
Next j
Call outcell(i + 1, m + 1, s(i))
Next i
End Sub
```

*Заметим, что первый оператор **Next** закрывает внутренний цикл **For**, а последний оператор **Next** закрывает внешний цикл **For**. Точно так же и для вложенных операторов **If**, операторы **End If** автоматически применяются для закрытия ближайшего к нему оператора **If**.*

*Вложенные структуры **Do . . . Loop** работают подобным же образом: самый дальний оператор **Loop** соответствует самому дальнему оператору **Do**.*

*При вводе/выводе элементов двумерного массива на рабочий лист Microsoft Excel удобно применять пользовательские процедуры ввода/вывода:*

```
Sub readcell(i As Integer, j As Integer, val As Variant)
val = Лист1.Cells(i, j).Value
End Sub
Sub outcell(i As Integer, j As Integer, val As Variant)
Лист1.Cells(i, j).Value = val
End Sub
```

*где i – номер строки, j – номер столбца рабочего листа.*

*7.3 Выход из структур управления*

*Оператор **Exit** позволяет выходить непосредственно из цикла **For**, цикла Do, процедуры Sub или процедуры Function.*

***For counter = start To end [Step -increment]***

***[блок операторов]***

***[Exit For]***

***[блок операторов]***

***Next [counter]***

***Do [{While | Until} условие]***

***[блок операторов]***

***[Exit Do]***

***[блок операторов]***

***Loop***

***Exit For*** *внутри цикла* ***For*** *и* ***Exit Do*** *внутри цикла* ***Do*** *могут появиться сколько угодно раз.*

*Оператор* ***Exit Do*** *работает со всеми разновидностями синтаксиса цикла* ***Do****.*

*Операторы* ***Exit For*** *и* ***Exit Do*** *применяются, если необходимо завершить цикл немедленно, не продолжая дальнейших итераций или не ожидая выполнения блока операторов в теле цикла.*

*При использовании оператора* ***Exit*** *для выхода из цикла значения переменной цикла зависят от того, каким образом завершается выполнение цикла:*

- *При нормальном завершении цикла значение переменной цикла имеет на единицу больше верхней границы числа циклов;*
- *При преждевременном завершении цикла переменная цикла сохраняет свое значение, которое она получила с учетом обычных правил;*
- *При завершении цикла по концу набора переменная цикла имеет значение Nothing (Ничего), если она является переменной типа object (Объект), или значение Empty (Пусто), если она является переменной типа Variant.*

# *8. Символьные данные. Операции и функции для работы с символьными данными.*

*8.1 Символьные данные.*

*Напомним, что по умолчанию данные строкового типа имеют переменную длину и могут удлиняться (до длины в 2^31 символов) или укорачиваться. Однако такие строки занимают на 10 байт памяти больше, поэтому можно объявить строки фиксированной длины, явно указав количество символов.*

***Dim s as String*20***

*Если количество символов будет меньше объявленного, то свободные места заполняются пробелами, при попытке занесения большего количества символов лишние отбрасываются.*

*8.2 Операции и функции для работы с символьными данными.*

*Единственная операция, применяемая при работе со строками – конкатенация.*

*Зато встроенных функций для работы достаточно много.*

## Функции для работы с символьными данными

| **Действие** | **Ключевые слова** |
|---|---|
| Сравнение двух строк. | StrComp |
| Преобразование строк. | StrConv |
| Изменение регистра. | Lcase, UCase |
| Создание строк с повторяющимися символами. | Space, String |
| Определение длины строки. | Len |
| Форматирование строки. | Format |
| Выравнивание строки. | Lset, RSet |
| Обработка строк. | InStr, Left, LTrim, Mid, Right, RTrim, Trim |
| Выбор типа сравнения строк. | Option Compare |
| Работа с кодами ASCII и ANSI. | Asc, Chr |

*Сравнение двух строк.*

*Функция **StrComp** возвращает значение типа Variant (Integer), представляющее результат сравнения строк.*

***StrComp(string1, string2[, compare])***

*Синтаксис функции StrComp содержит следующие аргументы:*

*string1 - Обязательный. Любое допустимое строковое выражение.*

*string2 - Обязательный. Любое допустимое строковое выражение.*

*compare - Необязательный. Указывает способ сравнения строк. Аргумент compare может быть опущен или иметь значение 0, 1 или 2. Чтобы выполнить двоичное сравнение, следует указать 0 (это значение используется по умолчанию). Чтобы выполнить посимвольное сравнение без учета регистра, следует указать 1. Если аргумент compare опущен, способ сравнения строк определяется значением параметра инструкции Option Compare.*

*Возвращаемые значения:*

| | |
|---|---|
| *string1 меньше чем string2* | *-1* |
| *string1 равняется string2* | *0* |
| *string1 больше чем string2* | *1* |

*Пример:*

```
Dim MyStr1$, MyStr2$, MyComp%
MyStr1 = "ABCD": MyStr2 = "abcd"
' Определяем переменные.
MyComp = StrComp(MyStr1, MyStr2, 1)
' Возвращает 0.
MyComp = StrComp(MyStr1, MyStr2, 0)
' Возвращает -1.
MyComp = StrComp(MyStr2, MyStr1)
' Возвращает 1.
```

*Выбор типа сравнения строк.*

*Применяется на уровне модуля для задания используемого по умолчанию способа сравнения строковых данных.*

***Option Compare {Binary | Text}***

*Инструкция Option Compare при ее использовании должна находиться в модуле перед любой процедурой и указывает способ сравнения строк (Binary или Text) для модуля. Если модуль не содержит инструкцию Option Compare, по умолчанию используется способ сравнения Binary*

*Инструкция Option Compare Binary задает сравнение строк на основе порядка сортировки, определяемого внутренним двоичным представлением символов. В Microsoft Windows порядок сортировки определяется кодовой страницей символов. В следующем примере представлен типичный результат двоичного порядка сортировки:*

***A < B < E < Z < a < b < e < z < Б < Л < Ш < б < л < ш***

*Инструкция Option Compare Text задает сравнение строк без учета регистра символов на основе системной национальной настройки. Тем же символам, что и выше, при сортировке с инструкцией Option Compare Text соответствует следующий порядок:*

***(A=a) < (B=b) < (E=e) < (Z=z) < (Б=б) < (Л=л) < (Ш=ш)***

*Преобразование строк.*

***StrConv(string, conversion)***

*string - строковое выражение, которое следует преобразовать, conversion - значение типа Integer, Сумма значений, указывающих тип преобразования, которое следует выполнить.*

*vbUpperCase 1 Преобразование строки к верхнему регистру.*

*vbLowerCase 2 Преобразование строки к нижнему регистру.*

*vbProperCase 3 Преобразование первой буквы каждого слова в строке в прописную.*

*Вместо одной этой функции можно применять функции **Lcase, Ucase.***

***Lcase** - возвращает значение типа String, представляющее строку, преобразованную к нижнему регистру.*

***LCase(строка)***

*Обязательный аргумент строка представляет любое допустимое строковое выражение. Если строка имеет значение Null, возвращается значение Null.*

*К нижнему регистру преобразуются только прописные буквы; строчные буквы и прочие символы остаются неизмененными.*

***Ucase** - возвращает значение типа Variant (String), содержащее строку, преобразованную к верхнему регистру.*

***UCase(строка)***

*Обязательный аргумент строка представляет любое допустимое строковое выражение. Если строка имеет значение Null, возвращается значение Null.*

*К верхнему регистру преобразуются только строчные буквы; прописные буквы и прочие символы остаются неизменными.*

*Создание строк, содержащих повторяющиеся символы.*

*Для создания таких строк применяют функции **Space и String**. Функция **Space(**число**)** формирует строку, а обязательный аргумент число указывает нужное число пробелов в строке. Ее удобно использовать для форматирования и очистки данных в строках фиксированной длины.*

*Функция **String(number, character)** содержит следующие аргументы:*

*number - Значение типа Long. Длина возвращаемой строки.*

*character - Значение типа Variant. Код символа или строковое выражение, первый символ которого используется при создании возвращаемой строки.*

*Определение длины строки.*

*При работе со строками переменной длины удобно использовать функцию **Len(строка)**, возвращающую значение типа Long, содержащее число символов в строке.*

*Выравнивание строки.*

*Оператор **Lset** - выравнивает строку по левому краю строковой переменной или копирует переменную одного определяемого пользователем тип в переменную другого типа, также определенного пользователем.*

***LSet переменная = строка***

***LSet имяПеременной1 = имяПеременной2***

*Здесь строка - строковое выражение, которое следует выровнять по левому краю строки переменная. Все оставшиеся символы в строке переменная LSet заменяет на пробелы. Если строка длиннее, чем переменная, LSet копирует в переменную столько начальных символов строки, сколько в ней поместится.*

***Rset** – то же самое, но выравнивает строку по правому краю строковой переменной. Если длина строки переменная больше, чем строка, инструкция RSet заменяет все оставшиеся символы вплоть до начала строки переменная пробелами.*

*Обработка строк.*

*Для разнообразной обработки строк применяются функции* ***InStr, Left, LTrim, Mid, Right, RTrim, Trim****.*

***InStr*** *- Возвращает значение типа Long, указывающее позицию первого вхождения одной строки внутри другой строки.*

***InStr([start, ]string1, string2[, compare])***

*Синтаксис функции InStr содержит следующие аргументы:*

*start - числовое выражение, задающее позицию, с которой начинается каждый поиск. Если этот аргумент опущен, поиск начинается с первого символа строки. Указание аргумента start является обязательным, если указан аргумент compare.*

*string1 - Строковое выражение, в котором выполняется поиск.*

*string2 - Искомое строковое выражение.*

*compare - Указывает способ сравнения строк. Аргумент compare может быть опущен или иметь значение 0 или 1. Чтобы выполнить двоичное сравнение, следует указать 0 (это значение используется по умолчанию). Чтобы выполнить посимвольное сравнение без учета регистра, следует указать 1. Если аргумент compare опущен, способ сравнения строк определяется значением параметра инструкции Option Compare.*

*Возвращаемые значения*

| | |
|---|---|
| *string1 является пустой строкой* | *0* |
| *string2 является пустой строкой* | *start* |
| *string2 не найдена* | *0* |
| *string2 найдена в string1* | *Позиция обнаруженной подстроки* |
| *start > string2* | *0* |

***Left*** *- возвращает значение типа String, содержащее указанное число первых символов строки.*

***Left(string, length)***

*Здесь string - строковое выражение, из которого извлекаются символы. length - значение типа Variant (Long), числовое выражение, указывающее число возвращаемых символов. Если 0, возвращается пустая строка (""). Если значение length больше либо равняется числу символов в строке string, возвращается вся строка.*

*Для определения числа символов в строке string следует использовать функцию Len.*

***Ltrim, Rtrim, Trim*** *- возвращают значение типа String, содержащее копию строки, из которой удалены пробелы, находившиеся в начале строки (LTrim), в конце строки (RTrim) или в начале и конце строки (Trim).*

***LTrim(строка), RTrim(строка), Trim(строка)***

*Обязательный аргумент строка представляет любое допустимое строковое выражение.*

***Mid -*** *Возвращает значение типа Variant (String), содержащее указанное число символов строки.*

***Mid(string, start[, length])***

*Здесь string - строковое выражение, из которого извлекаются символы, start - значение типа Long. Позиция символа в строке string, с которого начинается нужная подстрока. Если start больше числа символов в строке string, функция Mid возвращает пустую строку ("").*

*length - значение типа Variant (Long). Число возвращаемых символов. Если этот аргумент опущен или превышает число символов, расположенных справа от позиции start, то возвращаются все символы от позиции start до конца строки.*

***Right*** *- Возвращает значение типа String, содержащее указанное число последних символов строки.*

***Right(string, length)***

*Здесь string - строковое выражение, из которого извлекаются символы. length - значение типа Variant (Long). Числовое выражение, указывающее число возвращаемых символов. Если 0, возвращается пустая строка (""). Если превышает число символов в строке string, возвращается вся строка.*

*Работа с кодами ASCII.*

***Asc** - Возвращает значение типа Integer, представляющее код символа для первого символа строки.*

***Asc(строка)***

*Аргумент строка является любым допустимым строковым выражением. Если строка не содержит символов, возникает ошибка выполнения. Возвращаемые значения лежат в диапазоне 0 – 255.*

***Chr** - возвращает значение типа String, содержащее символ, соответствующий указанному коду символа.*

***Chr(кодСимвола)***

*Обязательный аргумент кодСимвола является значением типа Long, определяющим символ.*

*Коды 0–31 соответствуют стандартным управляющим символам ASCII. Например, Chr(10) возвращает символ перевода строки. Обычным диапазоном значений аргумента кодСимвола является интервал 0–255.*

*Пример: Заменить в строке буквы А, Б, В на 1, 2, 3 соответственно.*

```
Sub пример7()
Dim s As String, sn As String, t As String
Dim l As Integer, i As Integer
Call read("g1", s)
l = Len(s)
For i = 1 To l
t = Mid(s, i, 1)
Select Case t
Case "А": sn = sn + "1"
Case "Б": sn = sn + "2"
Case "В": sn = sn + "3"
Case Else: sn = sn + t
End Select
Next i
Call out("h1", sn)
End Sub
```

*Пример: Из набора слов вывести только те, у которых одинаковые первые буквы.*

```
Sub пример8()
Dim first() As String, s As String, n As Integer, i As Integer, k As Integer
i = 0: k = 0
Do While Лист1.Range("g" & i+1).Text <> ""
i = i + 1
s = Лист1.Range("g" & i).Text
ReDim Preserve first(1 To i)
first(i) = Left(s, 1)
Loop
Лист1.Range("h1", "h" & i).Clear
For n = 1 To i
If n <> i Then
For k = n + 1 To i
If first(k) = first(n) Then
Лист1.Range("h" & n).Value = Лист1.Range("g" & n).Text
Лист1.Range("h" & k).Value = Лист1.Range("g" & k).Text
End If : Next : End If : Next
End Sub
```

# *9.Булевские вектора и операции для работы с ними*

*Напомним, что в VBA имеется тип **Boolean –** для хранения логических (булевых) значений, которые сохраняются как 16-разрядные (двухбайтовые) числа, но могут иметь только значения **True** или **False**.*

*Переменные типа **Boolean** отображаются как строковые значения **True** или **False** (при использовании метода **Print**. Для присваивания переменным одного из двух логических значений ИСТИНА или ЛОЖЬ следует использовать ключевые слова **True** или **False**.*

*При преобразовании других числовых типов данных к типу Boolean значение 0 преобразуется в **False**, а все остальные значения преобразуются в True. Если значения типа **Boolean** преобразуются к другим типам данных, то **False** превращается в 0, а **True** в -1.По умолчанию значением булевской переменной является **False**- ложь.*

*Любое число можно представить в двоичном виде, а полученный набор нулей и единиц рассматривать как булевский вектор. Для получения представления числа в двоичном виде можно использовать следующий код:*

```
Sub пример9()
Dim b As String
Dim s As Single
Dim a As Integer
b = ""
s = CSng(InputBox("Введи десятичное", "Ввод данных", 255))
Do While s > 0
a = s Mod 2
b = CStr(a) + b
s = (s) \ 2
Loop
MsgBox b, vbOKOnly, "Результат"
End Sub
```

*Для преобразования двоичного представления числа в его десятичное значение можно использовать следующий код:*

```
Sub пример10()
Dim b As String
Dim i As Integer, l As Integer, k As Integer, j As Integer
b = InputBox("Введи двоичное", "Ввод данных", 11111111)
l = Len(b)
i = 0
For k = l To 1 Step -1
j = CInt(Mid(b, k, 1))
i = i + j * 2 ^ (l - k)
Next
MsgBox i, vbOKOnly, "Результат"
End Sub
```

*При работе с булевскими векторами часто приходится выделять из них конкретные разряды, тетрады (4 разряда), байты (2 тетрады) и слова (2 байта, 4 тетрады). Для этого применяются операции маскирования и/или сдвига. Маскирование представляет собой наложение на число (с помощью операции И) некоторой маски – числа, содержащего единицы в нужных разрядах и нули – в ненужных:*

***0001 0111 AND 0000 1111 = 0000 0111***

*Сдвиг представляет собой операцию добавления к числу слева или справа без изменения его разрядности указанного количества нулей. Вытолкнутые разряды при этом теряются:*

***0001 0111 SHL 4 = 0111 0000,***

***0001 0111 SHR 4 = 0000 0001***

*Для сдвига вправо двоичного представления числа **b** на **i** разрядов можно использовать код:*

```
shr = Left(b, Len(b) - i)
shr=String(i,”0”) +shr
```

*Для сдвига влево двоичного представления числа **b** на **i** разрядов можно использовать код:*

```
shl = Right(b, Len(b) - i)
shl=shl+String(i,”0”)
```

*Для работы с булевскими векторами применяются логические операции* ***And, Eqv, Imp*** *,* ***Not, Or, Xor.***

*Рассмотрим пример: Выделить в двухбайтовом слове 1-ю и 3-ю тетрады и объединить их по исключающему ИЛИ.*

*Для решения этой задачи необходимо сохранить заданное число (больше 255) во вспомогательной переменной, ввести маску*

***0000 0000 0000 1111*** *для выделения 1 тетрады, перевести ее в десятичный вид и объединить ее по И с заданным числом, ввести маску*

***0000 1111 0000 0000*** *для выделения 3 тетрады, перевести ее в десятичный вид и объединить ее по И с копией заданного числа и сдвинуть вправо на 8 разрядов, после чего объединить полученные значения с помощью операции* ***XOR.***

*Представленная ниже программа использует процедуру* ***tobin*** *для получения двоичного представления числа и функции* ***toint*** *для перевода двоичного представления к десятичному значению и* ***shr*** *для сдвига вправо.*

```
Sub пример11()
Dim a As Integer, b As Integer, m1 As Integer, m2 As Integer, rez As Integer
a = CInt(InputBox("Введи число", "Ввод данных", 256))
Call tobin(a): b = a
m1 = toint("111100000000")
m2 = toint("1111")
a = a And m1
Call tobin(a)
a = toint(shr(a, 8))
b = b And m2
Call tobin(b)
rez = a Xor b
Call tobin(rez)
End Sub
```

# *10.Подпрограмма-процедура*

*Можно упростить программирование, разбивая задачу на небольшие логические компоненты. Эти компоненты, называемые процедурами, могут впоследствии стать строительными блоками, которые позволят усилить и расширить VBA.*

*При программировании с использованием процедур можно выделить два основных преимущества:*

*Процедуры позволяют разбивать программы на конечное число логических единиц, каждую из которых легче отладить, чем всю программу без процедур;*

*Процедуры, разработанные для одной программы, могут выступать в качестве строительных блоков для других программ, обычно с небольшими изменениями или совсем без них;*

*В VBA используется несколько видов процедур:*

- *Процедуры Sub (не возвращают значения);*
- *Процедуры Function (возвращают значения);*
- *Процедуры Property (могут возвращать и присваивать значения), а также устанавливать ссылки на объекты.*

***Процедуры Sub***

*Процедура Sub (или подпрограмма) — это блок кода, который выполняется в ответ на событие. Разбивая код модуля на процедуры Sub, намного легче читать или модифицировать код приложения.*

*Синтаксис процедуры Sub таков:*

***[Private | Public][Static]Sub <Имя процедуры> (<аргументы>)***

***<Операторы>***

***End Sub***

*При вызове процедуры выполняются операторы между ключевыми словами Sub и End Sub. Процедуры Sub можно помещать в стандартные модули, модули классов и форм. По умолчанию процедуры Sub во всех модулях имеют атрибут public, который означает, что их можно вызывать из любого места приложения.*

*Задание параметров процедуры подобно объявлению переменных.*

```
Sub tobin(i As Integer)
Dim b As String, a As Integer, s As Single
b = "": s = CSng(i)
Do While s > 0
a = s Mod 2
b = CStr(a) + b
s = (s) \ 2
Loop
MsgBox b, vbOKOnly, "Результат"
End Sub
```

***Общие процедуры***

*Общая процедура указывает приложению, как выполнять конкретную задачу. Однажды определенная, она должна каждый раз специально вызываться приложением. В противоположность ей процедура обработки события после вызова остается в состоянии ожидания событий, вызванных пользователем или инициированных системой.*

*Зачем надо создавать общие процедуры? Одна причина заключается в том, что нескольким разным процедурам обработки событий может потребоваться выполнить одни и те же действия. Общие операторы помещают в отдельную процедуру (общую процедуру), а в процедуры обработки событий помещают вызовы этой процедуры. Это исключает дублирование кода и облегчает поддержку приложения.*

***Создание новых процедур***

*Для создания новой общей процедуры следует в окне кода набрать на клавиатуре заголовок процедуры и нажать клавишу Enter. Заголовок процедуры состоит из слова Sub или Function, за которым следует имя процедуры. Например:*

***Sub UpdateForm ()***

***Function GetCoord ()***

*В результате VBA заполняет шаблон для новой процедуры, т.е. дописывает строку End Sub или End Function, например:*

***Sub UpdateForm()***

***. . .***

***End Sub.***

***Просмотр существующих процедур***

*Чтобы просмотреть процедуру в текущем модуле:*

*Существующую общую процедуру можно увидеть, выбрав элемент (General) в списке Object в окне кода и затем процедуру в списке Procedure.*

*Существующую процедуру обработки события можно увидеть, выбрав соответствующий объект в списке Object в окне кода и затем процедуру в списке Procedure.*

*Чтобы просмотреть процедуру в другом модуле, следует:*

1. *Выбрать команду Object Browser (Просмотр объектов) меню View (Вид).*
2. *Выбрать проект в списке Project/Library (Проект/Библиотека).*
3. *Выбрать модуль в списке Classes (Классы) и процедуру в списке Members of (Члены)*
4. *Выбрать опцию View Definition (Посмотреть определение).*

***Вызов процедур Sub***

*Процедура Sub отличается от процедуры Function тем, что ее нельзя вызвать по имени в выражении. Ее вызов осуществляется в отдельном операторе. Также процедура sub, в отличие от функции, не возвращает значения. Однако, как и функция, она может изменять значения любых переменных, переданных ей в качестве параметров.*

*Существуют два способа вызова процедуры sub – при помощи ключевого слова Call и без него:*

*' Оба эти оператора вызывают процедуру Sub с именем MyProc.*

***Call MyProc (FirstArgument, SecondArgument)***

***MyProc FirstArgument, SecondArgument***

*Заметим, что при наличии ключевого слова **Call**, параметры заключены в круглые скобки. Если ключевое слово **Call** опускается, следует опустить и скобки вокруг списка параметров.*

***Вызов процедур из других модулей***

*Процедуры, находящиеся в других модулях, могут быть вызваны из любого места проекта. Возможно, придется указать модуль, в котором содержится вызываемая процедура. Способы вызова открытых процедур разнообразны и зависят от того, где расположена процедура - в модуле формы, модуле класса или стандартном модуле.*

***Процедуры в формах***

*Для вызова процедуры, находящейся во внешнем модуле, т.е. в не в том модуле, в котором находится код, из которого производится вызов процедуры, перед именем процедуры должно быть указано имя модуля, в котором находится код процедуры. К примеру, если код процедуры с именем SomeSub находится в модуле формы, названном Form1, то вызвать процедуру можно следующим оператором:*

***Call Form1.SomeSub(<аргументы>)***

***Процедуры в стандартных модулях***

*Процедура является уникальной, если она определена только в одном месте.*

*Если имя процедуры уникально, то включать имя модуля в вызов процедуры не обязательно. Внешние или внутренние вызовы процедуры будут ссылаться на это уникальное имя.*

*Если два или более модулей содержат процедуры с одинаковыми именами, при вызове таких процедур необходимо уточнять их с помощью имени соответствующего модуля. Вызов общей процедуры из модуля, ее содержащего, запускает процедуру этого модуля. Например, если процедура с именем CommonName существует и в модуле Module1, и в модуле Module2, то вызов CommonName из модуля Module2 запустит процедуру CommonName модуля Module2, а не процедуру CommonName модуля Module1.*

*Вызов общей процедуры из другого модуля должен уточнять модуль, в котором расположена процедура. Например, вызвать процедуру с именем CommonName модуля Module2 из модуля Module1 можно следующим образом:*

***Module2.CommonName(<аргументы>)***

# *11.Подпрограмма-функция.*

*VBA содержит встроенные или стандартные функции, например, sqr, cos или chr.*

*Кроме того, с помощью оператора Function можно писать собственные процедуры Function. Эти функции называют нестандартные или пользовательскими.*

*Синтаксис процедуры Function таков:*

***[Private|Public][Static]Function <имя процедуры> (<аргументы>) [As type <имя типа>]***

***<операторы>***

***End Function***

*Приведем конкретный пример:*

```
Function toint(b As String) As Integer
Dim i As Integer, l As Integer, k As Integer, j As Integer
l = Len(b): i = 0
For k = l To 1 Step -1
j = CInt(Mid(b, k, 1))
i = i + j * 2 ^ (l - k)
Next
toint = i
End Function
```

*Как и процедура Sub, процедура Function является самостоятельной и может принимать параметры, выполнять ряд операторов и изменять значения своих параметров. В отличие от процедуры Sub, имя процедуры Function может возвращать значение в вызывающую процедуру.*

*Существуют три различия между процедурами Sub и Function:*

*Возвращаемое процедурой Function значение присваивается самому имени <имя процедуры> процедуры. Возвращаемое процедурой Function значение можно использовать в выражениях в программе.*

*Как и переменные процедуры Function имеют тип, который определяет тип возвращаемого значения. (В отсутствие ключевого слова As в операторе определения процедуры ей назначается по умолчанию тип variant.)*

*Вызов процедуры Function, или просто функции, в основном осуществляется заданием ее имени и параметров в правой части большого оператора или в составе выражения.*

*Вот, например, функция, вычисляющая гипотенузу прямоугольного треугольника при заданных катетах:*

```
Function Hypotenuse (A As Integer, B As Integer) As String
Hypotenuse = Sqr(A^2 + B^2)
End Function
```

*В VBA процедура Function вызывается точно так же, как и любая встроенная функция:*

***strX = Hypotenuse(Width, Height)***

***Вызов процедур Function***

*Если функция не имеет аргументов, скобки после имени функции можно не ставить:*

*‘Все эти операторы вызывают не имеющую аргументов процедуру Function* ***ToDec****.*

***Debug.Print 10 * ToDec***

***X = ToDec***

***If ToDec =10 Then Debug. Print "Out of Range"***

***X = AnotherFunction(10 * ToDec)***

*Обычно вызов пользовательской функции аналогичен вызову встроенной (стандартной) функции VBA, например Abs.*

*Функцию можно вызывать так же, как и процедуру sub.*

*‘Следующие операторы вызывают одну и ту же функцию:*

***Call Year(Now)***

***Year Now***

*Значение функции при ее вызове подобным образом игнорируется.*

***Механизмы передачи параметров.***

*Обычно процедуре для выполнения требуются некоторые исходные данные. Эти данные могут передаваться процедуре при помощи параметров процедуры. При этом различают формальные параметры, то есть описанные в заголовке процедуры и используемые в ее теле, и фактические – те, что были указаны при ее вызове. Заметим, что имена формальных и фактических параметров не обязаны совпадать.*

***Типы данных параметров***

*По умолчанию параметры процедур имеют тип variant. Можно объявлять для параметров и другие типы данных. Заметим, что если типы фактически передаваемых в функцию значений отличаются от типа формальных параметров, то происходит неявное преобразование типов.*

*Поэтому рекомендуется описывать в заголовке функции типы ее формальных параметров, а при вызове функции проверять соответствие типов фактических и формальных параметров.*

***Передача параметров по значению***

*При передаче параметра по значению **(by value)** процедуре передается копия переменной, выступающей в качестве параметра процедуры. Если процедура изменяет значение параметра, это затрагивает только копию переменной, а не саму переменную. Значение переменной- оригинала в вызывающей процедуру программе сохраняется прежним.*

*Для передачи параметров по значению используется ключевое слово ByVal, например:*

***Sub PostAccounts(ByVal intAcctNum as Integer)***

***<операторы тела процедуры>***

***End Sub***

*Фактические параметры, заданные константами, всегда передаются по значению.*

***Передача параметров по ссылке***

*Передача процедуре параметров по ссылке **(by reference)** открывает ей доступ к области памяти, где хранится содержимое переменной. В результате процедура может изменять значение переменной, являющейся ее параметром. По умолчанию в VBA все параметры передаются по ссылке.*

*Если задается тип данных параметра, передаваемого по ссылке, то передаваемый параметр должен содержать значение специфицированного типа данных. Можно обойти это правило, передавая процедуре в качестве параметра выражение. В этом случае VBA вычислит выражение и передаст его значение процедуре, преобразовав его в нужный тип, если это окажется возможным.*

*Проще всего представить переменную как выражение, заключив ее в круглые скобки.*

*Например, чтобы передать процедуре целое число через строковый параметр, можно использовать следующий код:*

```
Private Sub Form_Load()
Dim intX As Integer
intX = 12 * 3
ВызываемаяПроцедура(intX)
End Sub

Sub ВызываемаяПроцедура (Bar As String)
MsgBox Bar   'Значение переменной Bar — строка "36"
End Sub
```

***Необязательные параметры***

*С помощью ключевого слова **Optional** в списке параметров можно задавать необязательные параметры (optional arguments) процедуры.*

*Если какой-то аргумент задан как необязательный, то и все последующие аргументы в списке аргументов должны быть необязательными и объявляются с ключевым словом optional.*

*В этом коде все аргументы не являются обязательными:*

```
Dim strИмя As String
Dim strАдрес As String
Sub Text(Optional x As String, Optional y As String)
MsgBox x
MsgBox y
End Sub

Private Sub Exec()
strИмя = "ВашеИмя"
strАдрес = 12345   ' Передаются два параметра.
Call Text(strИмя, strАдрес)
End Sub
```

*В этом коде некоторые аргументы обязательны:*

```
Dim strИмя As String
Dim varАдрес As Variant
Sub Text(x As String, Optional y As Variant)
MsgBox x
If IsMissing(y)=true Then
MsgBox y
End If
End Sub

Private Sub Exec()
strИмя = "ВашеИмя"
Call Text(strИмя) ' Второй параметр не передается.
End Sub
```

*Если необязательный параметр отсутствует, то он рассматривается как параметр с типом variant, имеющий значение Empty. В предыдущем при мере показано, как с помощью функции* ***IsMissing*** *проверять необязательные параметры.*

***Значения по умолчанию для необязательных параметров***

*В следующем примере процедура возвращает значение по умолчанию необязательного параметра, если при ее вызове он опущен:*

```
Sub ListText (x As String, Optional y As Integer = 12345)
MsgBox x
MsgBox y ' Выводится значение по умолчанию: 12345
End Sub

Private Sub Exec()
strName$ = "yourname"
Call Text (strName) ' Второй параметр не передается.
End Sub
```

***Неизвестное число параметров***

*Обычно при вызове процедуры число ее параметров должно быть точно таким же, как и при ее объявлении. Ключевое слово* ***ParamArray*** *"разрешает" процедуре принимать произвольное число параметров.*

```
Dim intSum As Integer
Sub Sum(ParamArray intNums())
Dim x As Variant, y As Integer
For Each x In intNums
y = y + x
Next x
intSum = y
End Sub
Private Sub Exec()
Sum 1, 3, 5, 7, 8
MsgBox "intsum=" & intSum
End Sub
```

***Рекурсия.*** *Под рекурсией понимают способность программы вызывать саму себя. Процедуры выделяют для хранения переменных ограниченный объем памяти в стеке. Стек - это рабочая область памяти, которая заполняется и освобождается динамически в соответствии с потребностями выполняемой программы. При каждом вызове процедурой самой себя выделяется дополнительный объем этой памяти. Процедура, вызывающая сама себя, называется рекурсивной. Рекурсивная процедура, которая бесконечно вызывает саму себя, приводит к ошибке. Например:*

```
Function RunOut(Maximum)
        RunOut = RunOut(Maximum)
End Function
```

*Эта ошибка менее очевидна, если две процедуры бесконечное число раз вызывают друг друга, или некоторое условие, ограничивающее рекурсию, никогда не выполняется. Рекурсия имеет свои области применения.*

*Например, следующая процедура использует рекурсию для вычисления факториалов:*

```
Function Factorial (N)
    If N <= 1 Then  ' Достигнут конец рекурсивных вызовов.
Factorial = 1       ' (N = 0) завершение вызовов.
    Else            ' Повторный вызов функции, если N > 0.
    Factorial = Factorial(N - 1) * N
    End If
End Function
```

*Необходимо проверить, что рекурсивная функция не вызывает себя столько раз, что начинает сказываться нехватка памяти. При возникновении ошибки, убедитесь, что процедура не вызывает себя бесконечное число раз. После этого попробуйте сэкономить память с помощью:*

- *устранения ненужных переменных;*
- *использования типов данных, отличных от Variant;*
- *переосмысления логики процедуры; часто вместо рекурсии можно воспользоваться вложенными циклами.*

***Создание пользовательской функции на VBA в среде EXCEL.***

*Любая процедура и функция, содержащиеся в общем модуле (но не в модуле формы и не в модуле класса), могут быть использована на рабочем листе Excel.*

*Для выполнения процедур используется команда меню* ***Сервис/Макрос/Макросы****, а для вставки функций применяется команда меню* ***Вставка/Функция*** *или кнопка стандартной панели инструментов* ***Вставка функции****.*

*При вставке функции запускается мастер функций, и если в проекте VBA имеется хотя бы одна функция, то в правом окне мастера функций появляется категория* ***Определенные пользователем****.*

*При выборе этой категории в левом окне мастера функций отображается перечень функций проекта VBA. После выбора необходимой функции из списка, мастер функций предлагает ввести ее параметры, количество которых определено в ее заголовке.*

*Если функция имеет необязательные параметры, или их число не определено, то мастер функций будет открывать окна для их ввода последовательно, по мере заполнения ранее открытых, как это происходит, например, при вставке функции рабочего листа СУММ.*

*Значения параметров можно ввести с клавиатуры, либо можно ввести адрес ячейки, содержащей значение, либо можно указать ячейку с параметром, щелкнув по ней правой кнопкой мыши. Заметим, что ввод в качестве параметра диапазона ячеек допустим лишь в том случае, когда функция имеет неопределенное количество параметров, либо вводимый параметр представляет собой массив, либо он имеет объектовый тип данных Range.*

*Поскольку функция не может быть выполнена непосредственно, а должна быть вызвана, то для ее отладки (например, в пошаговом режиме) необходимо написать процедуру, обеспечивающую ввод параметров функции, ее вызов и вывод результата работы функции. Если функция возвращает более одного значения (используя список формальных параметров), то эти значения не могут быть выведены на рабочий лист с помощью мастера функций.*

*В качестве примера перепишем в виде функции процедуру замены букв «А», «Б», «В» на цифры 1, 2, 3 соответственно*:

```
Function пример12(s As String)
Dim sn As String, t As String
Dim l As Integer, i As Integer
l = Len(s)
For i = 1 To l
t = Mid(s, i, 1)
Select Case t
Case "А": sn = sn + "1"
Case "Б": sn = sn + "2"
Case "В": sn = sn + "3"
Case Else: sn = sn + t
End Select
Next i
пример12 = sn
End Function
```

# *12. Использование файлов в VBA*

***Типы доступа к файлам.***

*Сам по себе файл не представляет ничего более, чем ряд связанных байтов, расположенных на диске. Когда приложение обращается к файлу, оно должно знать, что содержат эти байты (символы, целые числа, строки и т.д.) Тип данных, содержащихся в файле, определяет тип доступа к файлу. VBA три типа доступа к файлу:*

*Последовательный доступ, обычно используемый* для записи текстовых файлов, например протоколов ошибок или отчетов.

*Произвольный доступ*, используемый при необходимости считать и записать данные в файл без его закрытия. Файлы произвольного доступа содержат данные в виде записей, которые упрощают и ускоряют поиск нужных сведений.

*Двоичный доступ*, используется, когда требуется считать или записать байт в любую позицию в файле, например при сохранении или отображении точечных изображений.

*Последовательный доступ предполагает, что каждый символ в файле либо является текстовым символом, либо входит в форматирующую последовательность текста, как, например, символ новой строки. Данные хранятся, как символы в коде ANSI.*

*Предполагается, что файл, открытый для произвольного доступа, составлен из множества записей одинаковой длины. Можно применять определенные пользователем типы для создания записей, составленных из нескольких полей, каждое из которых может иметь разный тип данных. Данные хранятся как двоичная информация.*

*Двоичный доступ позволяет использовать файлы для хранения любых данных. Такой доступ похож на произвольный, за исключением того, что не существует никаких предположений о типе данных или длине записи. Однако необходимо точно знать, как данные были записаны в файл, чтобы можно было корректно извлекать их.*

***Функции и операторы доступа к файлам.***

*Ряд функций и операторов может быть применен для файлов всех трех типов, а некоторые работают только с файлами одного или двух типов. Со всеми типами доступа к файлам используют следующие :*

***Для работы с папками и дисками:***

*Функция **Dir** возвращает значение типа String, представляющее имя файла, каталога или папки, которое удовлетворяет указанному шаблону имени файла, набору атрибутов файла или метке тома на диске.*

***Dir[(путь[, атрибуты])]***

*Здесь **путь** (Необязательный) - строковое выражение, указывающее имя файла; может содержать имя каталога или папки и диска. Если путь не найден, возвращается пустая строка ("").*

***Атрибуты** (Необязательный) - константа или числовое выражение, описывающее атрибуты файла. Если этот аргумент опущен, возвращаются все файлы, имена которых удовлетворяют аргументу путь.*

*Функция **Dir** поддерживает использование подстановочных знаков для нескольких символов (*) и для одиночного символа (?) для указания нескольких файлов.*

*При первом вызове функции **Dir** необходимо указать путь, в противном случае возникает ошибка. Если указаны атрибуты файла, наличие аргумента путь является обязательным.*

*Функция **Dir** возвращает первое имя файла, имя которого удовлетворяет аргументу путь. Для получения остальных файлов, имена которых удовлетворяют указанному пути, следует повторно вызвать функцию **Dir** без аргументов. Если файлов, имена которых удовлетворяют указанному пути, не осталось, возвращается пустая строка (""). При следующем после возврата пустой строки вызове функции необходимо указать аргумент **путь**; в противном случае возникает ошибка. Изменить значение аргумента путь можно в любой момент, не дожидаясь, пока закончатся файлы, имена которых удовлетворяют текущему пути. Рекурсивный вызов функции **Dir** запрещен.*

*Вызов функции **Dir** с атрибутом vbDirectory не приводит к последовательному возврату подкаталогов.*

*Поскольку имена файлов возвращаются в произвольном порядке, их можно сохранить в массиве, а затем отсортировать этот массив.*

*Пример возврата списка каталогов на диске C:.*

```
MyPath = "C:\"          ' Указывает путь.
MyName = Dir(MyPath, vbDirectory)   ' Возвращает первый элемент.
Do While MyName <> ""       ' Начинает цикл.
' Игнорирует текущий каталог и каталог предыдущего уровня.
If MyName <> "." And MyName <> ".." Then
' Проверяет является ли MyName каталогом.
If (GetAttr(MyPath & MyName) And vbDirectory) = vbDirectory Then
Debug.Print MyName ' Выводит элемент если это каталог.
End If
End If
MyName = Dir          ' Возвращает следующий элемент.
Loop
```

*Инструкция* ***ChDir*** *- Изменяет текущий каталог или папку.*

***ChDir путь***

*Обязательный аргумент* ***путь*** *является строковым выражением, определяющим какой каталог (или папка) станет текущим. Аргумент* ***путь*** *может содержать имя диска. Если диск не указан, инструкция* ***ChDir*** *изменяет текущий каталог или папку на текущем диске.*

*Инструкция* ***ChDir*** *изменяет текущий каталог, но не изменяет стандартный диск. Например, если стандартным является диск C, то после выполнения следующей инструкции стандартным каталогом станет каталог на диске D, но стандартным диском останется диск C:*

***ChDir "D:\TMP"***

*Функция* ***CurDir*** *- Возвращает значение типа Variant (String), представляющее текущий путь.*

***CurDir[(диск)]***

*Необязательный аргумент* ***диск*** *является строковым выражением, указывающим существующий диск. Если диск не указан или аргумент диск является пустой строкой (""), функция* ***CurDir*** *возвращает путь к текущему диску.*

*Инструкция* ***MkDir*** *- Создает новый каталог или папку.*

***MkDir путь***

*Обязательный аргумент **путь** является строковым выражением, определяющим создаваемый каталог или папку. Аргумент путь может содержать имя диска. Если диск не указан, инструкция **MkDir** создает новый каталог или папку на текущем диске.*

*Инструкция **RmDir** - Удаляет существующий каталог или папку.*

***RmDir путь***

*Обязательный аргумент **путь** является строковым выражением, определяющим каталог или папку, которую следует удалить. Аргумент путь может содержать имя диска. Если диск не указан, инструкция **RmDir** удаляет каталог или папку с текущего диска.*

*При попытке удалить с помощью инструкции **RmDir** каталог (или папку), который содержит файлы, возникает ошибка. Для предварительного удаления всех файлов из каталога или папки следует использовать инструкцию **Kill**.*

*Инструкция **ChDrive** - Изменяет текущий диск.*

***ChDrive диск***

*Обязательный аргумент **диск** является строковым выражением, указывающим существующий диск. Если этот аргумент задается пустой строкой (""), текущий диск не изменится. Если диск является строкой, состоящей из нескольких символов, инструкция **ChDrive** использует только первый символ.*

***Для работы со свойствами файлов:***

*Функция **FileDateTime** - Возвращает значение типа Variant (Date), содержащее дату и время создания или последнего изменения файла.*

***FileDateTime(путь)***

*Обязательный аргумент **путь** является строковым выражением, указывающим имя файла. Аргумент путь может содержать имя каталога или папки и диска.*

*Функция FileLen - Возвращает значение типа Long, содержащее размер файла в байтах.*

***FileLen(путь)***

*Обязательный аргумент **путь** является строковым выражением, определяющим файл. Аргумент **путь** может содержать имя каталога или папки и диска.*

*Если в момент вызова функции **FileLen** указанный файл открыт, возвращается размер этого файла до его открытия. Для определения размера открытого файла следует использовать функцию **LOF**.*

*Функция **GetAttr** - возвращает значение типа Integer, содержащее атрибуты файла, каталога или папки.*

***GetAttr(путь)***

*Обязательный аргумент **путь** является строковым выражением, указывающим имя файла. Аргумент **путь** может содержать имя каталога или папки и диска.*

*Значение, возвращаемое функцией **GetAttr**, являются суммой следующих значений:*

| | | |
|---|---|---|
| *vbNormal* | *0* | *Обычный.* |
| *vbReadOnly* | *1* | *Только чтение.* |
| *vbHidden* | *2* | *Скрытый.* |
| *vbSystem* | *4* | *Системный.* |
| *vbDirectory* | *16* | *Каталог или папка.* |
| *vbArchive* | *32* | *Файл был изменен после последнего резервирования.* |

*Для определения установленных атрибутов следует с помощью оператора **And** выполнить поразрядное сравнение значения, возвращенного функцией **GetAttr** , и значения, соответствующего нужному атрибуту файла. Ненулевой результат означает, что данный атрибут установлен. Например, значение следующего выражения будет нулевым, если архивный атрибут не установлен:*

***Result = GetAttr(FName) And vbArchive***

*Если архивный атрибут установлен, будет возвращено ненулевое значение.*

*Инструкция **SetAttr** - задает атрибуты файла.*

***SetAttr pathname, attributes***

***pathname** (обязательный) - строковое выражение, указывающее имя файла; может содержать имя каталога или папки и диска.*

***attributes** (обязательный) - константа или числовое выражение, задающее атрибуты файла.*

*Ниже приведены допустимые значения аргумента **attributes**:*

| | | |
|---|---|---|
| *vbNormal* | *0* | *Обычный (по умолчанию).* |
| *vbReadOnly* | *1* | *Только чтение.* |
| *vbHidden* | *2* | *Скрытый.* |
| *VbSystem* | *4* | *Системный.* |
| *vbArchive* | *32* | *Файл был изменен после последнего резервирования.* |

*При попытке изменения атрибутов открытого файла возникает ошибка выполнения.*

***Для манипулирования файлами:***

*Инструкция **FileCopy** - копирует файл.*

***FileCopy source, destination***

***source** (обязательный) - строковое выражение, указывающее имя файла, подлежащего копированию. Аргумент source может содержать имя каталога или папки и диска.*

***destination** (обязательный) - строковое выражение, указывающее имя результирующего файла. Аргумент destination может содержать имя каталога или папки и диска.*

*При попытке скопировать открытый файл с помощью инструкции FileCopy возникает ошибка.*

*Инструкция **Name** - Изменяет имя файла, каталога или папки.*

***Name староеИмя As новоеИмя***

***староеИмя** (обязательный) - строковое выражение, указывающее имя и положение существующего файла; может содержать имя каталога или папки и диска.*

***новоеИмя** (обязательный) - строковое выражение, указывающее новое имя и положение файла; может содержать имя каталога или папки и диска. Файл с таким именем не должен существовать.*

*Оба аргумента, **новоеИмя** и **староеИмя**, должны указывать на один и тот же диск. Если путь, указанный с помощью аргумента **новоеИмя** существует и отличен от указанного с помощью аргумента **староеИмя**, инструкция **Name** переместит файл в новый каталог или папку и переименует его (если требуется). Если пути, указанные с помощью аргументов **новоеИмя** и **староеИмя**, разные, а имена файлов совпадают, инструкция **Name** переместит файл в новый каталог или папку без изменения его имени. С помощью инструкции **Name**, можно переместить файл из одного каталога (или папки) в другой, однако нельзя переместить каталог или папку.*

*При попытке переименовать открытый файл с помощью инструкции **Name** возникает ошибка. Прежде чем приступить к изменению имени файла, необходимо его закрыть.*

*В аргументах инструкции **Name** не допускается использование подстановочных знаков для нескольких символов (*) и оного символа (?).*

*Инструкция **Kill** - удаляет файлы с диска.*

***Kill путь***

*путь (обязательный) является строковым выражением, указывающим имена одного или нескольких файлов, подлежащих удалению. Аргумент путь может содержать имя каталога или папки и диска.*

*Инструкция **Kill** поддерживает использование подстановочных знаков для нескольких символов (*) и для одиночного символа (?) для указания нескольких файлов.*

*При попытке удалить открытый файл с помощью инструкции **Kill** возникает ошибка. Для удаления каталогов следует использовать инструкцию **RmDir.***

***Для управления файлами:***

*Инструкция **Open** - разрешает выполнение с файлом операций ввода/вывода.*

***Open путь For режим [Access доступ] [блокировка] As [#]номерФайла [Len=длина]***

***путь** (обязательный) - строковое выражение, указывающее имя файла; может содержать имя каталога или папки и имя диска.*

***режим** (обязательный) - ключевое слово, указывающее режим файла: **Append, Binary, Input, Output** или **Random**. По умолчанию, файл открывается в режиме Random.*

***доступ** (необязательный) - ключевое слово, указывающее операции, разрешенные с открытым файлом: **Read**, **Write** или **Read Write**.*

***блокировка** (необязательный) - ключевое слово, указывающее операции, разрешенные с открытым файлом другим процессам: **Shared**, **Lock Read**, **Lock Write** и **Lock Read Write**.*

***номерФайла** (обязательный) - допустимый номер файла в интервале от 1 до 511 включительно. Для определения следующего свободного номера файла следует использовать функцию **FreeFile**.*

***длина** (необязательный) - Число, меньшее либо равное 32 767 (байт). Для файлов, открытых в режиме Random, это значение является длиной записи. Для файлов с последовательным доступом это значение является числом буферизуемых символов.*

*Чтобы получить возможность выполнить любую операцию ввода/вывода, файл необходимо открыть.*

*Инструкция **Open** резервирует буфер ввода/вывода для файла и определяет режим использования этого буфера.*

*Если аргумент путь описывает несуществующий файл, такой файл будет создан при открытии в режиме **Append, Binary, Output** или **Random**.*

*Если файл уже открыт другим процессом и указанный режим доступа не разрешен, инструкция **Open** не будет выполнена и возникнет ошибка.*

*Если аргумент режим имеет значение **Binary**, предложение **Len** игнорируется.*

*В режимах **Binary, Input** и **Random** можно еще раз открыть уже открытый файл под другим номером, не закрывая его. В режиме **Append** и **Output** необходимо закрыть файл, чтобы получить возможность открыть его еще раз под другим номером.*

*Функция **FreeFile** - Возвращает значение типа Integer, представляющее следующий номер файла, доступный для использования с инструкцией **Open**.*

***FreeFile[(диапазонНомеров)]***

*Необязательный аргумент **диапазонНомеров** является выражением типа Variant, указывающим диапазон, из которого возвращается следующий свободный номер файла. Значение 0 (используется по умолчанию) задает возвращение номера файла из диапазона 1 – 255 включительно. Значение 1 задает возвращение номера файла из диапазона 256 – 511.*

*Функцию **FreeFile** используют для возвращения незанятого номера файла.*

*Инструкция **Close** - завершает операции ввода/вывода с файлом, открытым с помощью инструкции **Open**.*

*Инструкция **Close** - завершает операции ввода/вывода с файлом, открытым с помощью инструкции **Open**.*

***Close [списокНомеровФайлов]***

***списокНомеровФайлов** (необязательный) представляет один или несколько номеров файлов. При этом используется следующий синтаксис, где **номерФайла** -любой допустимый номер файла:*

***[[#]номерФайла] [, [#]номерФайла] . . .***

*Если **списокНомеровФайлов** опущен, закрываются все активные файлы, открытые с помощью инструкции **Open**.*

*При закрытии файла, открытого в режиме **Output** или **Append**, в него добавляется содержимое последнего буфера вывода. Все буферы, связанные с закрытым файлом, освобождаются.*

*Инструкция **Close** разрывает связь между файлом и соответствовавшим ему номером файла.*

*Инструкция **Reset** - закрывает все файлы, открытые с помощью инструкции **Open** и записывает содержимое всех буферов файлов на диск.*

***Reset***

*Инструкция **Reset** закрывает все активные файлы, открытые с помощью инструкции **Open**.*

***Для позиционирования в открытом файле:***

*Функция **Seek** - возвращает значение типа Long, определяющее текущее положение указателя чтения/записи внутри файла, открытого с помощью инструкции **Open**.*

***Seek(номерФайла)***

*Обязательный аргумент **номерФайла** является выражением типа Integer, содержащим допустимый номер файла.*

*Функция **Seek** возвращает значение в интервале от 1 до 2 147 483 647 (т.е. 2^31 – 1) включительно.*

*Ниже приведено описание значений, возвращаемых для каждого режима доступа к файлу.*

***Random** - номер записи, которая будет считана или записана следующей.*

***Binary, Output, Append, Input** -номер байта, с которого начинается выполнение следующей операции ввода/вывода. Первому байту файла соответствует 1, второму 2 и т.п.*

*Функция **EOF** - возвращает значение типа Integer содержащее логическое значение True при достижении конца файла.*

***EOF(номерФайла)***

*Обязательный аргумент **номерФайла** является выражением типа Integer, содержащим любой допустимый номер файла.*

*С помощью функции **EOF** можно избежать ошибок, возникающих при попытках чтения или записи после достижения конца файла.*

*Функция **EOF** возвращает значение **False** до тех пор, пока не будет достигнут конец файла. При использовании с файлами, открытыми в режиме **Random** или **Binary**, функция **EOF** возвращает значение **True**, если последней выполненной инструкции **Get** не удалось считать целую запись; в противном случае возвращается значение **False**.*

*Для файлов, открытых для доступа в режиме **Binary**, попытка чтения файла с помощью функции **Input** до возвращения функцией **EOF** значения **True** приводит к ошибке.*

*При чтении двоичных файлов с помощью функции **Input** следует вместо функции **EOF** использовать функции **LOF** и **LOC** или использовать с функцией **EOF** инструкцию **Get**.*

***Ввод/вывод в файлы различных типов доступа.***

***<u>Последовательный доступ к файлам:</u>***

*Последовательный доступ лучше применять к файлам, состоящим только из текста, созданного в текстовом редакторе. В них невыгодно хранить числа, так как в строковом виде они занимают больше места, чем в числовом.*

*Файлы, открытые для последовательного доступа, допускают следующие операции:*

*ввод символов в файл (**Input**);*

*вывод символов из файла (**Output**);*

*добавление символов в файл (**Append**).*

*Чтобы открыть файл для последовательного доступа, следует использовать оператор **Open**:*

***Open pathname For [Input | Output | Append] As filenum [Len = Buffersize]***

*Когда последовательный файл открывается для ввода (**Input**), файл должен существовать, иначе возникнет ошибка.*

*Однако при попытке открыть несуществующий файл для операций вывода (**Output**) или добавления (**Append**) оператор **Open** сначала создает файл, а затем открывает его.*

*Номер файла **filenum** является обязательным и может быть получен с помощью функции **FreeFile**. Необязательный параметр **Len** задает число символов в буфере для операций копирования данных из файла в приложение. Чем больше буфер, тем реже производятся обращения к диску.*

*После открытия файла для операций **Input, Output или Append** его надо закрыть с помощью оператора **Close**, прежде чем открывать для операции другого типа.*

*Для редактирования файла сначала надо прочитать его содержимое в программные переменные, затем изменить переменные и записать их содержимое обратно в файл. Чтобы получить содержимое текстового файла, следует открыть его для операции последовательного ввода (**Input**). Затем используют операторы: **Line Input #, Input ()** или **Input #**, чтобы скопировать файл в программные переменные. Существуют операторы и функции, которые для последовательных файлов читают и записывают один символ или одну строку за одну операцию.*

*Следующий фрагмент читает файл строка за строкой:*

```
Dim Lines As String, NextLine As String
Do Until EOF (FileNum)
Line Input # FileNum, NextLine
Lines = Lines + NextLine + Chr(13) + Chr(10)
Loop
```

*Оператор **Line Input #** распознает последовательность символов **CR/LF**, но не включает их в переменную. О сохранении этих символов должен позаботиться программист.*

*Можно также использовать оператор **Input #,** который читает числа и/или строковые выражения, записанные в файл. Например, для чтения строки адреса можно использовать следующий оператор:*

***Input # FileNum, name, street, city, zip, country***

*Можно использовать функцию **Input ()** для копирования любого числа символов из файла в переменную при условии, что они могут поместиться в этой переменной:*

***Lines = Input (n, FileNum)***

*Чтобы скопировать целый файл в переменную, следует использовать функцию **InputB**. Так как она возвращает строку в коде ANSI, для преобразования ее в строку UNICODE надо применять функцию **StrConv**:*

***Lines = StrConv(InputB(LOF(FileNum), FileNum), vbUnicode)***

*Для записи строк в файл следует открыть его для последовательного вывода (**Output**) или добавления (**Append**), а затем использовать оператор **Print #**, например:*

***Print #FileNum, Text***

*Существует также оператор **Write #**, который записывает список чисел и/или строковых выражений в файл. Он автоматически отделяет каждое выражение запятой и заключает строковые выражения в кавычки:*

***AnyString = "AnyChars" : AnyNum = 12345***

***Write #FileNim, AnyString, AnyNum***

*Результат будет выглядеть так:*

***"AnyChars",12345***

```
‘ Из одного текстового файла переписать в другой строки,
‘ не содержащие букв «а».
Sub пример13()
Dim str$, i%
Open "c:\data.txt" For Input As #1
Open "c:\dataout.txt" For Output As #2
Do Until EOF(1)
Line Input #1, str
p = 0
For i = 1 To Len(str)
If Mid(str, i, 1) = "à" Then p = p + 1
Next
If p = 0 Then Print #2, str
Loop
Close
End Sub
```

***Произвольный доступ к файлам:***

*Байты в файлах произвольного доступа формируют записи, каждая из которых содержит одно или более полей. Запись с одним полем соответствует любому стандартному типу. Запись с несколькими полями соответствует определяемому пользователем типу. Например, тип Worker, определяемый ниже, создает 19 байтные записи, которые состоят из трех полей:*

```
Type Worker
LastName As String * 10
Title As String * 7
Rank As String * 2
End Type
```

*После определения типа, соответствующего записи, следует объявить другие переменные, которые необходимы в процессе открытия файла для произвольного доступа, например:*

```
'Переменная записей
Public Employee As Worker
'Текущая запись
Public Position As Long
'Номер последней записи в файле
Public LastRecord As Long
```

*Открыть файл для производного доступа можно с помощью следующего синтаксиса оператора* ***Open****:*

***Open pathname [ For Random] As filenumber Len = reclength***

*Так как типом доступа по умолчанию является произвольный (Random), ключевые слова* ***For Random*** *не обязательны.*

*Выражение* ***Len = reclength*** *задает размер каждой записи в байтах. Заметим, что каждая строковая переменная в VBA хранится в формате Unicode, и надо задать длину этой строки.*

*Если* ***reclength*** *меньше, чем действительная длина записываемой в файл записи, генерируется ошибка.*

*Если* ***reclength*** *больше действительной длины записи, то запись сохраняется, но дисковое пространство при этом будет расходоваться нерационально.*

*Для открытия файла можно использовать следующий код:*

```
Dim FileNum As Integer, RecLength As Long, Employee As Worker
' Вычисляем длину каждой записи.
RecLength = LenB (Employee)
' Получаем следующий доступный номер файла.
FileNum = FreeFile
' Открываем новый файл оператором Open.
Open "MYFILE.FILE" For Random As FileNum Len = RecLength
```

*Чтобы отредактировать файл произвольного доступа, сначала следует считать записи из файла в программные переменные, затем изменить значения переменных и записать переменные обратно в файл. В следующих разделах обсуждается, как редактировать файлы, открытые для произвольного доступа.*

*Для копирования записей в переменные следует использовать оператор **Get**. Например, чтобы скопировать запись из файла Employee Records в переменную **Employee**, можно использовать следующий код:*

***Get #FileNum, Position, Employee***

*В этой строке **FileNum** содержит номер, который оператор **Open** использовал для открытия файла; **Position** содержит номер записи, которая копируется; переменная **Employee**, объявленная с определенным пользователем типом **Worker**, получает содержимое записи.*

*Добавлять или изменять записи в файлах, открытых для произвольного доступа, можно с помощью оператора **Put**. Чтобы заменить запись, следует использовать оператор **Put**, задавая положение записи в файле, например:*

***Put #FileNum, Position, Employee***

*Этот код заменит запись с номером, заданным переменной* ***Position****, данными из переменной* ***Employee****.*

*Чтобы добавить новые записи в конец файла, открытого для произвольного доступа, следует использовать оператор* ***Put****, показанный в предыдущем фрагменте кода. Для этого надо установить значение переменной ,* ***Position*** *равным на единицу больше номера последней записи в файле. Например, чтобы добавить новую запись в файл, содержащий пять записей, следует установить переменную* ***Position*** *равной 6.*

*Следующий оператор добавляет запись в конец файла:*

```
LastRecord = LastRecord + 1
Put #FileNum, LastRecord, Employee
```

*Можно удалить запись, очистив ее поля, но тогда запись все еще останется в файле. Не стоит просто очищать записи в файле, так как, оставаясь в нем, они расходуют пространство и служат помехой при выполнении операций последовательного доступа. Лучше скопировать остающиеся в файле записи в новый файл, и затем удалить старый.*

*Чтобы удалить запись в файле произвольного доступа, следует:*

*Создать новый файл.*

*Скопировать все необходимые из исходного файла в этот новый файл.*

*Закрыть исходный файл и с помощью оператора **Kill** удалить его.*

*С помощью оператора **Name** переименовать новый файл, дав ему имя исходного файла.*

***Доступ к двоичным файлам:***

*Двоичный доступ дает возможность полного управления файлом, так как байты в файле могут представлять собой все что угодно. Например, можно сократить дисковое пространство, построив записи переменной длины. Двоичный доступ следует использовать, если важно сохранить небольшой размер файла.*

*Чтобы открыть файл для двоичного доступа, используют следующий синтаксис оператора* ***Open:***

***Open pathname For Binary As filename***

*Отличие от оператора для произвольного доступа заключается в отсутствии указания длины записи с помощью предложения* ***Len = RecLength.*** *Будучи включено в оператор открытия двоичного файла, оно просто игнорируется, поскольку предполагается, что записи имеют произвольную длину. Т.о невозможно организовать доступ к записям в произвольном порядке, а только последовательно, определяя длину каждой записи. В то же время можно непосредственно обращаться к каждому байту в файле.*