Коммерциялық емес акционерлік қоғам

АЛМАТЫ ЭНЕРГЕТИКА ЖӘНЕ БАЙЛАНЫС УНИВЕРСИТЕТІ

«Ғарыш техникасы және технологиялары» кафедрасы

АЛГОРИТМДЕУ ЖӘНЕ ПРОГРАММАЛАУ

5В074600- Ғарыш техникасы және технологиялары мамандығының студенттеріне арналған дәрістер конспектілері

Алматы 2017 ж.

Құрастырушы: М.Қ. Шимырбаев. Алгоритмдеу және программалау. 5В074600- Ғарыш техникасы және технологиялары мамандығының студенттеріне арналған дәрістер конспектілері - Алматы: АЭжБУ, 2017.- б.

Дәрістер конспектілерінде «Алгоритммдеу және программалау» пәнінің негізгі теориялық материалы қамтылған. Конспектілерде әртүрлі алгоритмдерді құрудың негізгі ұстамдары, және Visual Basic программалау тілі туралы базалық мәліметтер келтірілген. Бұл тілдің қосымша мүмкіндіктері программалаудың арнайы құралдары көрсетілген.

Ил.-86. Табл. – 9. Библиограф. – 16 назв.

Рецензент: т.ғ.к. Құсайынов Б.Қ.

«Алматы энергетика және байланыс университеті» коммерциялық емес акционерлік қоғамының 2017 ж. басылым жоспары бойынша басылып отыр.

© КАҚ «Алматы энергетика және байланыс университеті», 2017.

Мазмұны:

Дәріс 1. Алгоритмизацияның негіздері. Алгоритмы и величины. Этапы решения задачи на ЭВМ…….. 4
Дәріс 2. Данные и величины……. 6
Дәріс 3. Линейные вычислительные алгоритмы……. 9
Дәріс 4. Составление алгоритмов……. 14
Дәріс 5. Составление алгоритмов с предусловием……. 17
Дәріс 6. Алгоритм цикла с вложенным ветвлением. Вспомогательные алгоритмы и процедуры……. 21
Дәріс 7. История и классификация языков программирования…… 24
Дәріс 8. Алфавит и основные операторы языка бейсик……. 28
Дәріс 9. Операторы языка Бейсик…….. 33
Дәріс 10. Программирование повторений…… 36
Дәріс 11. Работа в системе Visual Basic…… 42
Дәріс 12. Массивы……. 58
Дәріс 13. Процедуры……. 66
Дәріс 14. Рекурсия……. 71
Дәріс 15. Файлы……. 76
Дәріс 16 . Меню…… 85
Дәріс 17. База данных…… 90
Дәріс 18. Анимация. Способы создания анимации…….. 94
Дәріс 19. Дополнительные средства проектирования приложений …. 96
Дәріс 20. Дополнительные средства отображения информации……... 101
Дәріс 21. Объекты и программирование……. 104
Дәріс 22. Обработка событий мыши. Событие Click…….. 105
Дәріс 23. Обработка событий клавиатуры…… 110
Список литературы…… 118

Лекция 1. Основы алгоритмизации. Алгоритмы и величины Этапы решения задачи на ЭВМ

Содержание лекции: алгоритмы и величины. Этапы решения задачи на ЭВМ.

Цель лекции: дать основные понятия дисциплины и ознакомить с основными этапами решения задач.

Работа по решению любой задачи с использованием компьютера делится на следующие этапы:

1) Постановка задачи.
2) Формализация задачи.
3) Построение алгоритма.
4) Составление программы на языке программирования.
5) Отладка и тестирование программы.
6) Проведение расчетов и анализ полученных результатов.

Часто эту последовательность называют технологической цепочкой решения задачи на ЭВМ. Непосредственно к программированию в этом списке относятся пункты 3, 4, 5. На этапе постановки задачи должно быть четко сформулировано, что дано и что требуется найти. Здесь очень важно определить полный набор исходных данных, необходимых для получения решения.

Второй этап — формализация задачи. Здесь чаще всего задача переводится на язык математических формул, уравнений, отношений. Если решение требует математического описания какого-то реального объекта, явления или процесса, то формализация равносильна получению соответствующей математической модели.

Третий этап — построение алгоритма. Опытные программисты часто сразу пишут программы на языках, не прибегая к каким-либо специальным способам описания алгоритмов (блок-схемам, псевдокодам). Однако в учебных целях полезно использовать эти средства, а затем переводить полученный алгоритм на язык программирования.

Первые три этапа предусматривают работу без компьютера. Дальше следует собственно программирование на определенном языке, в определенной системе программирования. Последний (шестой) этап — это использование уже разработанной программы в практических целях.

Таким образом, программист должен обладать следующими знаниями и навыками:

- уметь строить алгоритмы;
- знать языки программирования;
- уметь работать в соответствующей системе программирования.

Понятие алгоритма.

Одним из фундаментальных понятий в информатике является понятие алгоритма. Происхождение самого термина «алгоритм» связано с математикой. Это слово происходит от Algorithm! — латинского написания имени Мухаммеда аль-Хорезми (787—850 гг), выдающегося математика средневекового Востока. В XII в. был выполнен латинский перевод его математического трактата, из которого европейцы узнали о десятичной позиционной системе счисления и правилах арифметики многозначных чисел.

Именно эти правила в то время называли алгоритмами. Сложение, вычитание, умножение столбиком, деление уголком многозначных чисел — вот первые алгоритмы в математике. Правила алгебраических преобразований, способы вычислений корней уравнений также можно отнести к математическим алгоритмам.

В наше время понятие алгоритма трактуется шире. *Алгоритм* — это последовательность команд управления каким-либо исполнителем. В школьном курсе информатики с понятием алгоритма, с методами построения алгоритмов ученики знакомятся на примерах учебных исполнителей: Робота, Черепахи, Чертежника и т.д.

Эти исполнители ничего не вычисляют. Они создают рисунки на экране, перемещаются в лабиринтах, перетаскивают предметы с места на место. Таких исполнителей принято называть исполнителями, работающими в обстановке.

В разделе информатики под названием «Программирование» изучаются методы программного управления работой ЭВМ.

Следовательно, в качестве исполнителя выступает компьютер. Компьютер работает с величинами — различными информационными объектами: числами, символами, кодами и т. п. Поэтому алгоритмы, предназначенные для управления компьютером, принято называть алгоритмами работы с величинами.

Рисунок 1.1- Схема алгоритма перехода улицы

Если предписание должно быть выполнено машиной, то необходимо точное, «формальное» описание для соответствующего алгоритма.

Для алгоритмов важно различать следующие аспекты:

1) Постановку задачи, которая должна быть решена с помощью алгоритма.

2) Специфичный способ, каким решается задача, при этом для алгоритма различают:

- элементарные шаги обработки;
- описание выбора отдельных подлежащих выполнению шагов.

Свойства алгоритмов.

Массовость - способность получать результат для исходных данных из некоторого множества.

Результативность - способность получать требуемый результат за конечное число шагов.

Детерминированность - способность получать один и тот же результат на каждом шаге для одного и того же набора исходных данных.

Язык программирования.

Чтобы алгоритм можно было описать в форме, выполнимой на одной из машин, применяют формальные языки.

Описание алгоритма на формально описанном языке называется *программой*, а формальный язык - *языком программирования*.

Формальные языки описываются с помощью специальных нотаций, которые задают правила построения слов. Одна из распространенных нотаций БНФ- нотация (форма Бэкуса- Наура), которая позволяет писать выражения, определяющие множество последовательностей символов, т.е. формальных языков.

Лекция 2. Данные и величины

Содержание лекции: определение величин и данных. Типы величин и данных. Составление простых алгоритмов.

Цель лекции: дать понятия о величинах, типах величин, данных и типах данных.

Совокупность величин, с которыми работает компьютер, принято называть данными. По отношению к программе данные делятся на исходные, результаты (окончательные данные) и промежуточные (рисунок 2.1), которые получаются в процессе вычислений.

Рисунок 2.1- Классификация данных

Например, при решении квадратного уравнения:

$$ax^2 + bx + c = 0$$

исходными данными являются коэффициенты a, b, c, результатами — корни уравнения x_1, x_2, промежуточным данным — дискриминант уравнения:

$$D = b^2 — 4ac.$$

Для успешного освоения программирования необходимо усвоить следующее правило: всякая величина занимает свое определенное место в памяти ЭВМ (иногда говорят — ячейку памяти). Хотя термин «ячейка» с точки зрения архитектуры современных ЭВМ несколько устарел, однако в учебных целях его удобно использовать.

У всякой величины имеются три основных свойства: имя, значение и тип. На уровне команд процессора величина идентифицируется при помощи адреса ячейки памяти, в которой она хранится. В алгоритмах и языках программирования величины делятся на константы и переменные.

Константа — неизменная величина, и в алгоритме она представляется собственным значением, например: 15, 34.7, k, true и т.д.

Переменные величины могут изменять свои значения в ходе выполнения программы и представляются символическими именами — идентификаторами, например: X, S2, codl5. Любая константа, как и переменная, занимает ячейку памяти, а значение этих величин определяется двоичным кодом в этой ячейке.

Типы величин.

Теперь о типах величин — типах данных. С понятием типа данных вы уже, возможно, встречались, изучая в курсе информатики базы данных и электронные таблицы. Это понятие является фундаментальным для программирования.

В каждом языке программирования существует своя концепция типов данных, своя система типов.

Тем не менее, в любой язык входит минимально необходимый набор основных типов данных, к которому относятся: целый, вещественный, логический и символьный типы.

С типом величины связаны три ее характеристики:

- множество допустимых значений,
- множество допустимых операций,
- форма внутреннего представления.

В таблице 2.1 представлены эти характеристики основных типов данных.

Типы констант.

Типы констант определяются по контексту (т. е. по форме записи в тексте), а типы переменных устанавливаются в описаниях переменных. Есть еще один вариант классификации данных — по структуре. Данные делятся на

простые и структурированные. Для простых величин (их еще называют скалярными) справедливо утверждение: одна величина — одно значение, для структурированных: одна величина — множество значений.

К структурированным величинам относятся массивы, строки, множества и т.д.

Таблица 2.1 - Типы данных и величин

Тип	Значения	Операции	Внутреннее представление
Целый	Целые положительные и отрицательные числа в некотором диапазоне. Примеры 23,-12, 387	Арифметические операции с целыми числами: +, -, ·, целое деление и остаток от деления. Операции отношений (<, >, = и др.)	Формат с фиксированной точкой
Вещественный	Любые (целые и дробные) числа в некотором диапазоне. Примеры: 2,5, -0,01, 45,0, $3{,}6 \cdot 10^9$	Арифметические операции: +, -, ·, /. Операции отношений	Формат плавающей точки
Логический	True (истина) False (лож)	Логические операции: И and, ИЛИ or, НЕ not. Операции отношений	1 бит: 1- true 0- false
Символьный	Любые цифры компьютерного алфавита. Примеры: ‘a’, ‘5’, ‘+’, ‘S’	Операции отношений	Коды таблицы символьной кодировки. 1 символ – 1 байт

ЭВМ — исполнитель алгоритмов. Как известно, всякий алгоритм (программа) составляется для конкретного исполнителя в рамках его системы команд. О каком же исполнителе идет речь при обсуждении вопроса о программировании для ЭВМ?

Ответ очевиден: исполнителем является компьютер. Точнее говоря, исполнителем является комплекс ЭВМ + Система программирования (СП). Программист составляет программу на том языке, на который ориентирована СП. Иногда в литературе такой комплекс называют виртуальной ЭВМ.

Например, компьютер с работающей системой программирования на Бэйсике называют Бэйсик-машиной; компьютер с работающей системой программирования на Паскале называют Паскаль-машиной и т.п. Схематически это изображено на рисунке 2.2.

Независимо от того, на каком языке программирования будет написана программа, алгоритм решения любой задачи на ЭВМ может быть составлен из команд:

- присваивания;
- ввода;
- вывода;
- обращения к вспомогательному алгоритму;
- цикла;
- ветвления.

Рисунок 2.2 - Схема Паскаль-машины

Для описания алгоритмов в дальнейшем мы будем использовать блок-схемы и учебный алгоритмический язык.

Лекция 3. Линейные вычислительные алгоритмы

Содержание лекции: виды вычислительных алгоритмов. Линейные вычислительные алгоритмы. Нелинейные вычислительные алгоритмы. Арифметические и неарифметические выражения.

Цель лекции: научить составлять линейные и нелинейные алгоритмы. Дать понятия арифметическим и неарифметическим выражениям.

Главным простым воздействием в вычислительных методах считается присваивание смысла переменной величине. В случае, если смысл константы определен обликом ее записи, то переменная получает определенный смысл лишь только в итоге присваивания. Присваивание имеет возможность реализоваться 2-мя методами: с поддержкой команды присваивания и с поддержкой команды ввода. Рассмотрим образец: в учебнике арифметики критерии деления обычных дробей описаны так:

1) Числитель 1 дроби умножить на знаменатель 2 дроби.

2) Знаменатель 1 дроби умножить на числитель 2 дроби.

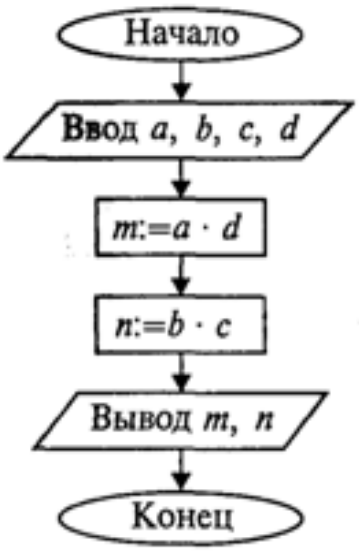
3) Записать дробь, числитель которой есть итог выполнения пункта 1, а знаменатель — итог выполнения пункта 2. В алгебраической форме это смотрится грядущим образом:

$$\frac{a}{b} : \frac{c}{d} = \frac{a \cdot d}{b \cdot c} = \frac{m}{n}.$$

Воспроизведем метод деления дробей для ЭВМ.

В данном методе сохраним те же обозначения для переменных, которые применены в записанной выше формуле. Начальными данными считаются целочисленные переменные a, b, c, d. Итогом - тип целых величин.

Блок-схема и слово метода на учебном алгоритмическом языке приведены на рисунке 3.1 (в последующем для краткости станем называть учебный алгоритмический язык АЯ).




```
алг Деление дробей
нач
  цел a,b,c,d,m,n
  ввод a,b,c,d
  m:=a*d
  n:=b*c
  вывод m,n
кон
```

Рисунок 3.1- Схема алгоритма и блок-схема вычисления деления дробей

Формат команды присваивания следующий:

переменная:=выражение

Знак «:=» нужно читать как «присвоить».

Команда присваивания обозначает следующие действия, выполняемые компьютером:

1) Вычисляется выражение.

2) Полученное значение присваивается переменной.

В приведенном выше алгоритме присутствуют две команды присваивания. В блок-схемах команда присваивания записывается в прямоугольнике. Такой блок называется вычислительным блоком.

В описаниях алгоритмов необязательно соблюдать строгие правила в записи выражений. Их можно писать в обычной математической форме. Это еще не язык программирования со строгим синтаксисом.

В приведенном алгоритме присутствует команда ввода:

ввод *a,b,c,d*

В блок-схеме команда ввода записывается в параллелограмме — блоке ввода-вывода. При выполнении данной команды процессор прерывает работу и ожидает действий пользователя. Пользователь должен набрать на

устройстве ввода (клавиатуре) значения вводимых переменных и нажать на клавишу ввода Enter.

Значения следует вводить в том же порядке, в каком соответствующие переменные расположены в списке ввода

Обычно с помощью команды ввода присваиваются значения исходных данных, а команда присваивания используется для получения промежуточных и конечных величин.

Полученные компьютером результаты решения задачи должны быть сообщены пользователю. Для этих целей предназначена команда вывода:

вывод *m,n*

С помощью этой команды результаты выводятся на экран или на устройство печати на бумагу.

Пример линейного алгоритма

Задача о нахождении площади стен комнаты

1) Ввести длину комнаты, присвоить переменной A введенное значение.

2) Ввести ширину комнаты, присвоить переменной B введенное значение.

3) Ввести высоту комнаты, присвоить переменной H введенное значение.

4) Произвести вычисление по формуле: S=2·(A+B)·H.

5) Вывести (напечатать) результат - значение переменной S - площадь стен комнаты.

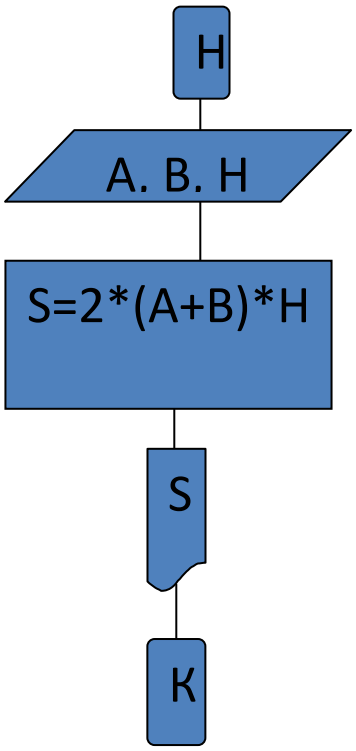


Рисунок 3.2 - Блок-схема линейного алгоритма вычисления

Метод, в котором очередность выполнения кое-каких предписаний находится в зависимости от выполнения проверяемых исполнителем критериев, именуется нелинейным (рисунок 3.3).

Вычислительный процесс исполняется или по одним, или по иным формулам, в зависимости от выполнения кое-какого закономерного обстоятельства.

Образец нелинейного (разветвляющегося) метода.

Упорядочить a,b,c по возрастанию (от наименьшего к большему), применяя дополнительную функцию h.

Перестановка осуществляется в следующей последовательности:

1) h=a.

2) a=b.

3) b=h.

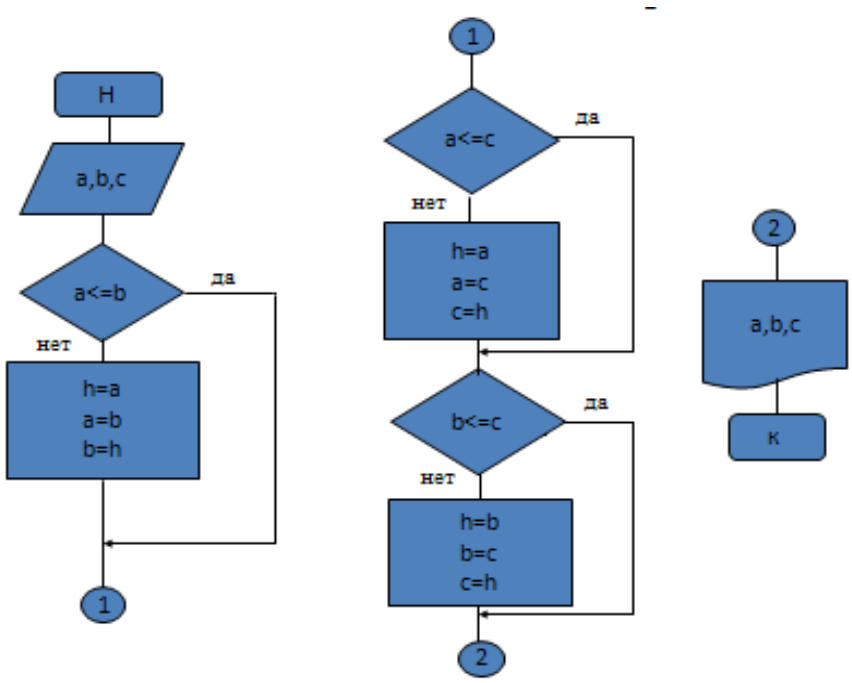


Рисунок 3.3 - Блок-схема нелинейного алгоритма

Арифметические и неарифметические выражения.

Арифметическое выражение - очередность количеств, констант, переменных, функций и арифметических выражений, заключенных в круглые скобки, которые объединены меж собой символами арифметических операций.

Логическое (условное) выражение.

Логическое выражение с обычным условием.

Несложное условие - это 2 выражения, меж которыми помещается сигнал сопоставления.

Операция	Описание операции
>	Больше чем
>=	Больше или равно
<	Меньше чем
<=	Меньше или равно
=	Равно
<>	Не равно

Простое условие имеет значение:
True - истина
False - ложь

Пример:
2.9994 < 2.9995 True
3.1416 < = 3.1416 True
1.3E-18 < 2.3E-19 False
-Y^2 > Abs(Y)

Логическое выражение со сложным условием.

Сложное условие – это очередность несложных критериев или же иных относительных выражений, заключенных в круглые скобки, которые объединены меж собой символами закономерных операций.

And - логическое умножение.

Or - логическое телосложение.

Not - логическое отрицание.

Смысл закономерного выражения рассчитывается по правилам вычисления закономерных выражений:

Логическое произведение подлинно лишь только тогда, когда оба закономерных сомножителя истинны.

A	B	A And B
True	True	True
True	False	False
False	True	False
False	False	False

Логическая сумма ложна только тогда, когда оба логических слагаемых ложны.

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Логическое отрицание выражения ложно, если само это выражение истинно, и наоборот.

A	Not A
True	False
False	True

Лекция 4. Составление алгоритмов

Содержание лекции: дать практические навыки по составлению алгоритмов.

Цель лекции: научить студентов составлять алгоритмы различных видов.

Так как операция присваивания считается важной операцией в вычислительных методах, рассмотрим ее более детально.

Рассмотрим последовательное выполнение 4 команд присваивания, в коих принимают участие 2 переменные величины a и b.

В приведенной ниже таблице визави всякой команды присваивания указываются смысл переменных, которые уточняются впоследствии выполнения команды.

Этот пример иллюстрирует три основных свойства команды присваивания:

- пока переменной не присвоено значение, она остается неопределенной;
- значение, присвоенное переменной, сохраняется в ней вплоть до выполнения следующей команды присваивания этой переменной;
- новое значение, присваиваемое переменной, заменяет ее предыдущее значение.

Таблица 4.1 - Схема работы команды присваивания

Команда	*a*	*b*
a: =1	1	-
b: =2 · a	1	2
a: = b	2	2
b: = a + b	2	4

Рассмотрим один очень полезный алгоритм, который приходится часто использовать при программировании. Даны две величины: X и Y. Требуется произвести между ними обмен значениями. Например, если первоначально было X = 1, Y = 2, то после обмена должно стать: X = 2, Y = 1.

Хорошей моделью для решения этой задачи является следующая ситуация: имеются два стакана — один с молоком, другой с водой. Требуется произвести обмен их содержимым. Всякому ясно, что в этом случае нужен дополнительный третий пустой стакан. Последовательность действий будет следующей:

1) Перелить из первого стакана в третий.

2) Перелить из второго в первый.

3) Перелить из третьего во второй. Цель достигнута!

По аналогии для обмена значениями двух переменных нужна третья дополнительная переменная. Назовем ее Z. Тогда задача обмена решается последовательным выполнением трех команд присваивания.

Таблица 4.2 - Схема работы команды присваивания

Команда	*X*	*Y*	*Z*
ввод X, Y	1	2	-
Z : = X	1	2	1
X: = Y	2	2	1
Y : = Z	2	1	1

Аналогия со стаканами не совсем точна в том смысле, что при переливании из одного стакана в другой первый становится пустым. В результате же присваивания (X: = Y) переменная, стоящая справа (Y), сохраняет свое значение.

Алгоритм для деления дробей имеет линейную структуру. В нем все команды выполняются в строго однозначной последовательности, каждая по одному разу. Линейный алгоритм составляется из команд присваивания, ввода, вывода и обращения к вспомогательным алгоритмам (об этом позже).

При описании алгоритмов в блок-схемах типы, как правило, не указываются (но подразумеваются). В алгоритмах на АЯ для всех переменных типы указываются явно. Описание типов переменных производится сразу после заголовка алгоритма. В них используются следующие обозначения типов: цел — целый тип, вещ — вещественный тип, лит — символьный (литерный) тип, лог — логический тип. В алгоритме для деления дробей для всех переменных указан целый тип.

Ветвления и циклы в вычислительных алгоритмах.

Составим алгоритм решения квадратного уравнения

$$ax^2 + bx + c = 0.$$

Задача хорошо знакома из математики. Исходными данными здесь являются коэффициенты a, b, c. Решением в общем случае будут два корня x_1 и x_2, которые вычисляются по формуле:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Все используемые в этой программе величины вещественного типа.

```
алг корни квадратного уравнения
вещ a,b,c,x1,x2,d
нач ввод a,b,c
  d:=b^2 - 4ac
  x1:=(-b+√d)/(2a)
  x2:=(-b-√d)/(2a)
  вывод x1,x2
кон
```

Слабость такого алгоритма видна невооруженным глазом. Он не обладает важнейшим свойством, предъявляемым к качественным алгоритмам, — универсальностью по отношению к исходным данным. Какими бы ни были значения исходных данных, алгоритм должен приводить к определенному результату и завершать работу.

Результатом может быть число, но может быть и сообщение о том, что при определенных данных задача решения не имеет. Недопустимы остановки в середине алгоритма из-за невозможности выполнить какую-то операцию. Упомянутое свойство в литературе по программированию называют результативностью алгоритма (в любом случае должен быть получен какой-то результат).

Чтобы построить универсальный алгоритм, сначала требуется тщательно проанализировать математическое содержание задачи.

Решение уравнения зависит от значений коэффициентов a, b, c. Вот анализ рассмотренной выше задачи (ограничиваемся только поиском вещественных корней):

- если a = 0, b = 0, c = 0, то любое x — решение уравнения;
- если a = 0,b = 0, c ≠ 0, то уравнение действительных решений не имеет;
- если a = 0, b ≠ 0, то это линейное уравнение, которое имеет одно решение x = -c/b;
- если a ≠ 0 и d = b2- 4ac ≥ 0, то уравнение имеет два вещественных корня (формулы приведены выше);
- если a ≠ 0 и d < 0, то уравнение не имеет вещественных корней.

Блок-схема алгоритма приведена на рисунке 4.2.

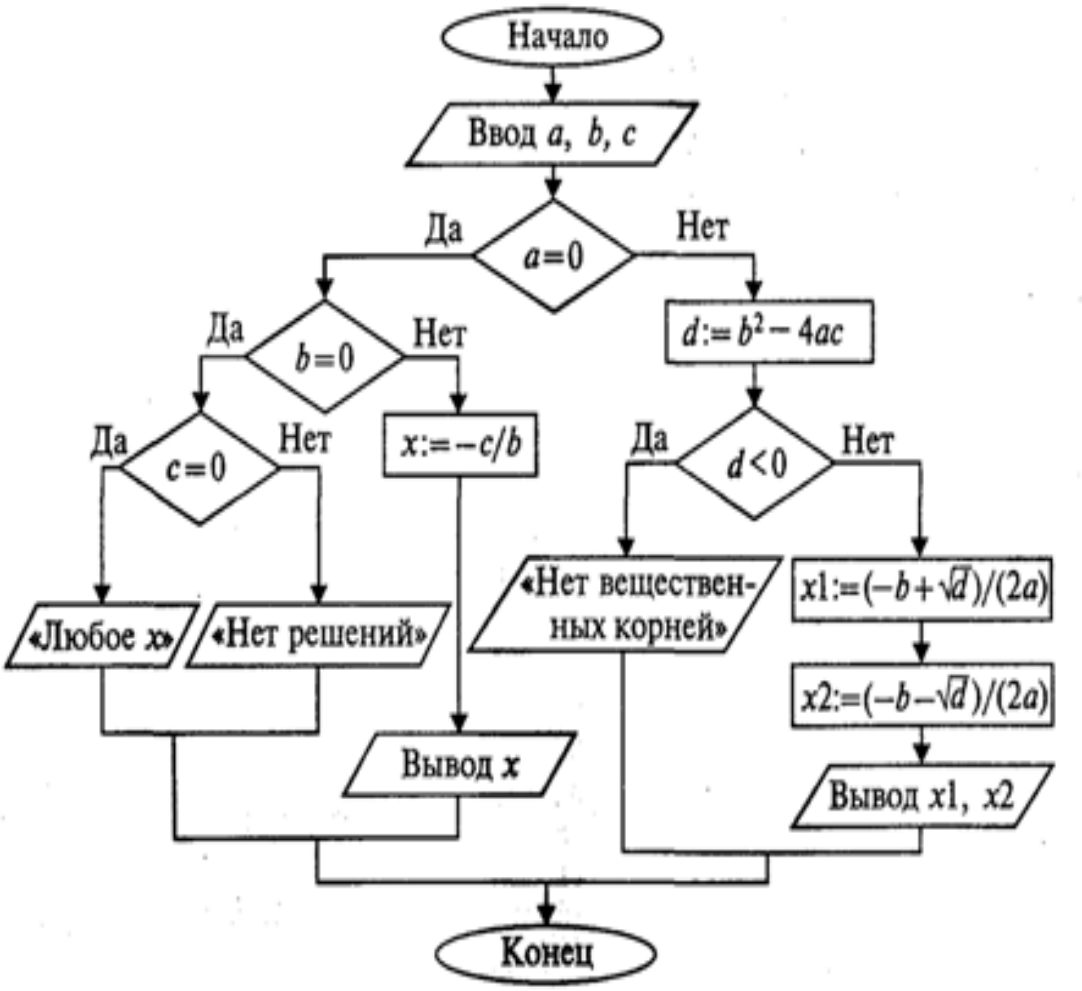


Рисунок 4.2 - Блок-схема алгоритма

Лекция 5. Составление алгоритмов с предусловием

Содержание лекции: практические методы составления алгоритмов с предусловием.

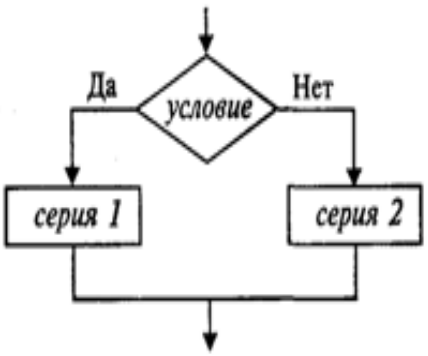
Цель лекции: научить составлять алгоритмы с предусловием.

Блок-схема алгоритма, показанного на рисунке 4.2, на алгоритмическом языке:

```
алг корни квадратного уравнения
вещ a,b,c,d,x1,x2
нач ввод a,b,c
  если a=0
  то   если b=0
       то   если c=0
            то вывод «Любое x – решение»
            иначе вывод «Нет решений»
            кв
       иначе  x:=-c/b
            вывод x
    кв
  иначе d:=b²-4ac
  если d<0
  то вывод «Нет вещественных корней»
  иначе x1:=(-b+√d)/(2a);x2:=(-b-√d)/(2a)
            вывод «x1=»,x1,«x2=»,x2
  кв
  кв
кон
```

Рисунок 5.1- Алгоритм, записанный на алгоритмическом языке

В этом алгоритме многократно использована структурная команда ветвления. Общий вид команды ветвления в блок-схемах и на алгоритмическом языке следующий:




```
если условие
то серия 1
иначе серия 2
кв
```

Рисунок 5.2 - Общий вид команды ветвления

Вначале проверяется условие (вычисляется отношение, логическое выражение). Если условие истинно, то выполняется серия 1 — последовательность команд, на которую указывает стрелка с надписью «да» (положительная ветвь). В противном случае выполняется серия 2 (отрицательная ветвь).

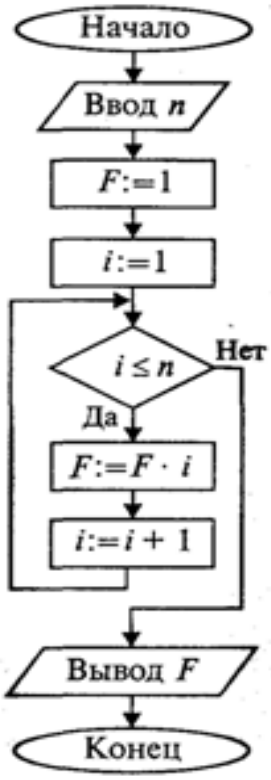
В АЯ условие записывается после служебного слова, если положительная ветвь — после слова *то*, отрицательная — после слова *иначе*. Буквы *кв* обозначают конец ветвления.

Если на ветвях одного ветвления содержатся другие ветвления, то такой алгоритм имеет структуру вложенных ветвлений. Именно такую структуру имеет алгоритм «Корни квадратного уравнения».

Рассмотрим следующую задачу: дано целое положительное число *n*. Требуется вычислить *n!* (n-факториал). Вспомним определение факториала:

$$n! = \begin{cases} 1, & \text{если } n = 0; \\ 1 \cdot 2 \ldots n, & \text{если } n \geq 1. \end{cases}$$

На рисунке 5.3 приведена блок-схема алгоритма. В нем используются три переменные целого типа: *n* — аргумент; *i* — промежуточная переменная; *F* — результат. Для проверки правильности алгоритма построена трассировочная таблица. В такой таблице для конкретных значений исходных данных по шагам прослеживается изменение переменных, входящих в алгоритм. Данная таблица составлена для случая *n = 3*.



Шаг	*n*	*F*	*i*	Условие
1	3			
2		1		
3			1	
4				1≤3, да
5		1		
6			2	
7				2≤3, да
8		2		
9			3	
10				3≤3, да
11		6		
12			4	
13				4≤3, нет
14		вывод		

Рисунок 5.3 - Блок-схема алгоритма ветвления с табличным пояснением

Трассировка доказывает правильность алгоритма. Теперь запишем этот алгоритм на алгоритмическом языке.

```
алг Факториал
цел n,i,F
нач ввод n
    F:=1;i:=1
    пока i≤n, повторять
    нц F:=F*i
       i:=i+1
    кц
    вывод F
кон
```

Рисунок 5.4 - Алгоритм ветвления на алгоритмическом языке

Этот алгоритм имеет циклическую структуру. В алгоритме использована структурная команда цикл-пока, или цикл с предусловием. Общий вид команды цикл-пока в блок-схемах и в АЯ следующий:

- *пока условие, повторять;*
- *нц;*
- *серия;*
- *кц.*

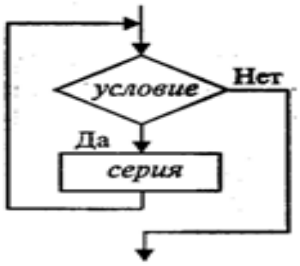
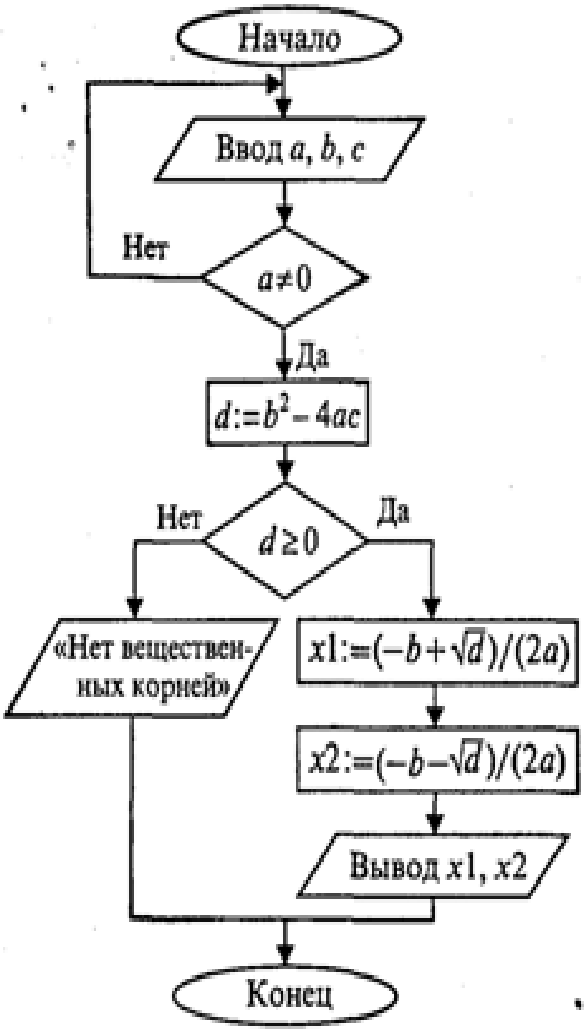


Рисунок 5.5 - Блок-схема «цикл-пока»

Выполнение серии команд (тела цикла) повторяется, пока условие цикла истинно. Когда условие становится ложным, цикл заканчивает выполнение. Служебные слова *нц* и *кц* обозначают «начало цикла» и «конец цикла» соответственно.

Цикл с предусловием — это основная, но не единственная форма организации циклических алгоритмов. Другим вариантом является цикл с постусловием. Вернемся к алгоритму решения квадратного уравнения. К нему можно подойти с такой позиции: если $a = 0$, то это уже не квадратное уравнение и его можно не рассматривать.

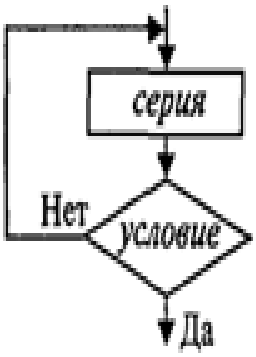
В таком случае будем считать, что пользователь ошибся при вводе данных, и следует предложить ему повторить ввод. Иначе говоря, в алгоритме будет предусмотрен контроль достоверности исходных данных с предоставлением пользователю возможности исправить ошибку. Наличие такого контроля — еще один признак хорошего качества программы.




```
алг квадратное
уравнение
вещ a,b,c,d,x1,x2
нач
  повторять
     ввод a,b,c
  до a≠0
  d:=b²-4ac
  если d≥0
  то x1:=(-b+√d)/(2a)
     x2:=(-b-√d)/(2a)
     вывод x1,x2
  иначе
     вывод «Нет
     вещественных
      корней»
  кв
кон
```

Рисунок 5.6 - Блок-схема цикла с предусловием

В общем виде структурная команда цикл с постусловием или цикл—до представляется так:




```
повторять
   серия
до условие
```

Рисунок 5.7 - Блок-схема цикла с постусловием

Здесь используется условие окончания цикла. Когда оно становится истинным, цикл заканчивает работу.

Лекция 6. Алгоритм цикла с вложенным ветвлением. Вспомогательные алгоритмы и процедуры

Содержание лекции: основные понятия об алгоритмах с вложенным ветвлением.

Цель лекции: научить составлять алгоритмы с вложенным циклом.

Составим алгоритм решения следующей задачи: даны два натуральных числа M и N. Требуется вычислить их наибольший общий делитель — НОД (M, N).

Эта задача решается с помощью метода, известного под названием алгоритма Евклида.

Его идея основана на том свойстве, что если M > N, то НОД(M, N) = НОД(M — N,N). Другой факт, лежащий в основе алгоритма, тривиален — НОД(M, M) = M. Для «ручного» выполнения этот алгоритм можно описать в форме следующей инструкции:

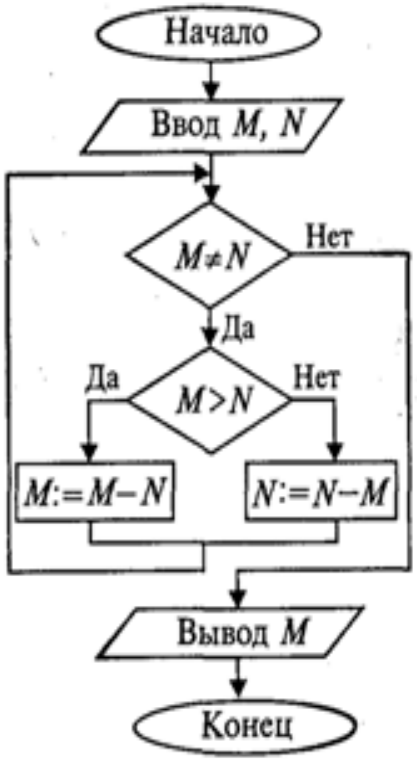
1) Если числа равны, то взять их общее значение в качестве ответа; в противном случае продолжить выполнение алгоритма.

2) Определить большее из чисел.

3) Заменить большее число разностью большего и меньшего значений.

4) Вернуться к выполнению пункта 1.

Блок-схема и алгоритм на АЯ будут следующими:




```
алг Евклид
цел M,N
нач ввод M,N
пока M≠N, повторять
нц если  M>N
   то M:=M-N
   иначе N:=N-M
   кв
кц
кон
```

Рисунок 6.1 - Алгоритм с вложенным циклом

Алгоритм имеет структуру цикла с вложенным ветвлением. Проделайте самостоятельно трассировку этого алгоритма для случая M = 18, N = 12. В результате получится НОД = 6, что, очевидно, верно.

Вспомогательные алгоритмы и процедуры.

В теории алгоритмов известно понятие вспомогательного алгоритма. Вспомогательным называется алгоритм решения некоторой подзадачи из основной решаемой задачи. В таком случае алгоритм решения исходной задачи называется основным алгоритмом.

В качестве примера рассмотрим следующую задачу: требуется составить алгоритм вычисления степенной функции с целым показателем $y = x^k$, где k — целое число, $x \neq 0$. В алгебре такая функция определена следующим образом:

$$x^n = \begin{cases} 1 & n = 0, \\ \dfrac{1}{x^{-n}} & n < 0, \\ x^n & n > 0. \end{cases}$$

Для данной задачи в качестве подзадачи можно рассматривать возведение числа в целую положительную степень.

Учитывая, что , запишем основной алгоритм решения этой задачи.

```
алг Степенная функция
цел n; вещ x,y
нач ввод x,n
  если n=0
  то  y:=1
  иначе  если   n>0
         то  СТЕПЕНЬ(x,n,y)
         иначе  СТЕПЕНЬ(1/x,-n,y)
         кв
кв
вывод y
кон
```

Рисунок 6.2 - Алгоритм решения задачи по возведению в степень

Здесь дважды присутствует команда обращения к вспомогательному алгоритму с именем СТЕПЕНЬ. Это алгоритм возведения вещественного основания в целую положительную степень путем его многократного

перемножения. Величины, стоящие в скобках в команде обращения к вспомогательному алгоритму, называются фактическими параметрами.

В учебном алгоритмическом языке вспомогательные алгоритмы оформляются в виде процедур. Запишем на АЯ процедуру СТЕПЕНЬ.

```
процедура СТЕПЕНЬ(вещ a,цел k,вещ z)
цел i
нач z:=1; i:=1
   пока i≤k, повторять
   нц z:=z*a
      i:=i+1
   кц
кон
```

Рисунок 6.3 - Функция «Уровень» на алгоритмическом языке

Заголовок вспомогательного алгоритма начинается со слова «процедура», после которого следует имя процедуры и в скобках — список формальных параметров. В этом списке перечисляются переменные-аргументы и переменные-результаты с указанием их типов.

Здесь a и *k* — формальные параметры-аргументы, z — параметр-результат. Следовательно, процедура «степень» производит вычисления по формуле $z = a^k$. В основном алгоритме «Степенная функция» обращение к процедуре производится путем указания ее имени с последующим в скобках списком фактических параметров.

Между формальными и фактическими параметрами процедуры должны выполняться следующие правила соответствия:

- по количеству (сколько формальных, столько и фактических параметров);
- по последовательности (первому формальному соответствует первый фактический параметр, второму — второй и т.д.);
- по типам (типы соответствующих формальных и фактических параметров должны совпадать).

Фактические параметры-аргументы могут быть выражениями соответствующего типа.

Обращение к процедуре инициирует следующие действия:

1) Значения параметров-аргументов присваиваются соответствующим формальным параметрам.

2) Выполняется тело процедуры (команды внутри процедуры).

3) Значение результата передается соответствующему фактическому параметру, и происходит переход к выполнению следующей команды основного алгоритма.

В процедуре «степень» нет команд ввода исходных данных и вывода результатов. Здесь присваивание начальных значений аргументам (a, n) производится через передачу параметров-аргументов. А присваивание результата переменной (y) происходит через передачу параметра-результата (z).

Таким образом, передача значений параметров процедур — это третий способ присваивания (наряду с командой присваивания и командой ввода).

Использование процедур позволяет строить сложные алгоритмы методом последовательной детализации.

Лекция 7. История и классификация языков программирования

Содержание лекции: краткие исторические сведения об алгоритмических языках.

Цель лекции: дать сведения о классификации языков программирования.

Язык программирования — это способ записи программ решения различных задач на ЭВМ в понятной для компьютера форме. Процессор компьютера непосредственно понимает язык машинных команд (ЯМК).

Программы на ЯМК программисты писали лишь для самых первых ламповых машин — ЭВМ первого поколения. Программирование на ЯМК — дело непростое. Программист должен знать числовые коды всех машинных команд, должен сам распределять память под команды программы и данные.

В 1950-х гг. появляются первые средства автоматизации программирования — языки Автокоды. Позднее для языков этого уровня стало применяться название «Ассемблеры». Появление языков типа Автокод-Ассемблер облегчило участь программистов.

Переменные величины стали изображаться символическими именами. Числовые коды операций заменились на мнемонические (словесные) обозначения, которые легче запомнить. Язык программирования стал понятнее для человека, но при этом удалился от языка машинных команд. Чтобы компьютер мог исполнять программы на Автокоде, потребовался специальный переводчик — транслятор. Транслятор — это системная программа, переводящая текст программы на Автокоде в текст эквивалентной программы на ЯМК.

Компьютер, оснащенный транслятором с Автокода, понимает Автокод. В этом случае можно говорить о псевдо-ЭВМ (аппаратура плюс транслятор с Автокода), языком которой является Автокод. Языки типа Автокод-Ассемблер являются машинно-ориентированными, т.е. они настроены на структуру машинных команд конкретного компьютера. Разные компьютеры с разными типами процессоров имеют разный Ассемблер.

Языки программирования высокого уровня (ЯПВУ) являются машинно-независимыми языками. Одна и та же программа на таком языке может быть

выполнена на ЭВМ разных типов, оснащенных соответствующим транслятором. Форма записи программ на ЯПВУ по сравнению с Автокодом еще ближе к традиционной математической форме, к естественному языку.

Очень скоро вы увидите, что, например, на языке Паскаль она почти такая же, как на школьном Алгоритмическом языке. ЯПВУ легко изучаются, хорошо поддерживают структурную методику программирования.

Первыми популярными языками высокого уровня, появившимися в 1950-х гг., были Фортран, Кобол (в США) и Алгол (в Европе).

Языки Фортран и Алгол были ориентированы на научно-технические расчеты математического характера. Кобол — язык для программирования экономических задач. В Коболе по сравнению с двумя другими названными языками слабее развиты математические средства, но зато хорошо развиты средства обработки текстов, организация вывода данных в форме требуемого документа.

Для первых ЯПВУ предметная ориентация языков была характерной чертой. Большое количество языков программирования появилось в 1960—1970-х гг. А за всю историю ЭВМ их было создано более тысячи. Но распространились, выдержали испытание временем немногие. В 1965 г. в Дартмутском университете был разработан язык Бейсик.

По замыслу авторов это простой язык, легко изучаемый, предназначенный для программирования несложных расчетных задач. Наибольшее распространение Бейсик получил на микроЭВМ и персональных компьютерах. На некоторых моделях школьных компьютеров программировать можно только на Бейсике.

В эпоху ЭВМ третьего поколения получил большое распространение язык PL/I (Program Language One), разработанный фирмой IBM. Это был первый язык, претендовавший на универсальность, т. е. на возможность решать любые задачи: вычислительные, обработки текстов, накопления и поиска информации. Однако PL/I оказался слишком сложным языком.

Для машин типа IBM 360/370 транслятор с него не мог считаться оптимальным, содержал ряд невыявленных ошибок. На ЭВМ класса мини и микро он вообще не получил распространения. Однако тенденция к универсализации языков оказалась перспективной. Старые языки были модернизированы в универсальные варианты — Алгол-68, Фортран-77.

Значительным событием в истории языков программирования стало создание в 1971 г. языка Паскаль. Его автор — швейцарский профессор Н.Вирт — разрабатывал Паскаль как учебный язык структурного программирования. Наибольший успех в распространении этого языка обеспечили персональные компьютеры. Фирма Borland International, Inc (США) разработала систему программирования Турбо Паскаль для ПК. Турбо Паскаль — это не только язык и транслятор с него, но еще и операционная оболочка, обеспечивающая пользователю удобство работы.

Турбо Паскаль вышел за рамки учебного предназначения и стал языком профессионального программирования с универсальными возможностями.

Транслятор с Турбо Паскаля по оптимальности создаваемых им программ близок наиболее удачному в этом отношении транслятору — транслятору с Фортрана.

В силу названных достоинств, Паскаль стал основой многих современных языков программирования, например, таких, как Ада, Модула-2 и др.

Язык программирования Си (английское название — C) создавался как инструментальный язык для разработки операционных систем, трансляторов, баз данных и других системных и прикладных программ.

Так же, как и Паскаль, Си — это язык структурного программирования, но, в отличие от Паскаля, в нем заложены возможности непосредственного обращения к некоторым машинным командам, к определенным участкам памяти компьютера. Дальнейшее развитие Си привело к созданию языка объектно-ориентированного программирования Си++.

Модула-2 — это еще один язык, предложенный Н.Виртом, основанный на языке Паскаль и содержащий средства для создания больших программ.

ЭВМ будущего, пятого поколения называют машинами «искусственного интеллекта». Но прототипы языков для этих машин были созданы существенно раньше их физического появления. Это языки ЛИСП и Пролог.

ЛИСП появился в 1965 г. Язык ЛИСП основан на понятии рекурсивно определенных функций. А поскольку доказано, что любой алгоритм может быть описан с помощью некоторого набора рекурсивных функций, то ЛИСП, по сути, является универсальным языком.

С его помощью на ЭВМ можно моделировать достаточно сложные процессы, в частности интеллектуальную деятельность людей.

Язык Пролог разработан во Франции в 1972 г. также для решения проблемы «искусственного интеллекта». Пролог позволяет в формальном виде описывать различные утверждения, логику рассуждений и заставляет ЭВМ давать ответы на заданные вопросы.

Реализовать тот или иной язык программирования на ЭВМ — это значит создать транслятор с этого языка для данной ЭВМ. Существуют два различных метода трансляции. Они называются соответственно «компиляция» и «интерпретация». Для объяснения их различия можно предложить следующую аналогию: лектор должен выступить перед аудиторией на незнакомом ей языке. Перевод можно организовать двумя способами:

- полный предварительный перевод — лектор заранее передает текст выступления переводчику, тот записывает перевод, размножает его и раздает слушателям (после чего лектор может и не выступать);
- синхронный перевод — лектор читает доклад, переводчик одновременно с ним слово в слово переводит выступление.

Компиляция является аналогом полного предварительного перевода; интерпретация — аналогом синхронного перевода. Транслятор, работающий

по принципу компиляции, называется компилятором; транслятор, работающий методом интерпретации, — интерпретатором.

При компиляции в память ЭВМ загружается программа-компилятор. Она воспринимает текст программы на ЯПВУ как исходную информацию. После завершения компиляции получается программа на языке машинных команд. Затем в памяти остается только программа на ЯМК, которая выполняется, и получаются требуемые результаты.

Интерпретатор в течение всего времени работы программы находится во внутренней памяти.

В ОЗУ помещается и программа на ЯПВУ. Интерпретатор в последовательности выполнения алгоритма «читает» очередной оператор программы, переводит его в команды и тут же выполняет эти команды. Затем переходит к переводу и выполнению следующего оператора. При этом результаты предыдущих переводов в памяти не сохраняются.

Поэтому использование компиляторов удобнее для больших программ, требующих быстрого счета. Программы на Паскале, Си, Фортране всегда компилируются. Бейсик чаще всего реализован через интерпретатор.

Visual Basic for Applications (VBA) – встроенный язык программирования Excel. VBA положен в основу системы программирования *Visual Basic*. Как и Visual Basic, VBA реализует концепцию визуального программирования, управляемого событиями.

VBA позволяет существенно расширить вычислительные возможности Excel. С помощью VBA можно легко и быстро создавать различные приложения.

VBA имеет графическую инструментальную среду, позволяющую создавать экранные формы, управляющие элементы, собственные функции, работать с которыми на рабочем листе можно при помощи мастера функций точно так же, как и с любой встроенной функцией.

В VBA широко используются элементы объектно-ориентированного программирования (ООП) и связанные с ним понятия. В основе ООП, управляемого событиями, лежат понятия *класс, инкапсуляция, объект, событие.* В качестве объектов могут рассматриваться конкретные предметы, а также абстрактные или реальные сущности.

В частном случае, в VBA *объектом* являются элементы пользовательского интерфейса, которые создаются на Форме пользователя (UserForm) или на рабочем листе, а также рабочая книга и ее элементы. Доступ к объекту при ООП возможен только через его методы и свойства. Каждый объект является представителем некоторого *класса* однотипных объектов, т.е. объект является экземпляром класса. Класс определяет общие для всех его объектов методы и свойства.

Методы — это программные процедуры, реализующие некоторый алгоритм, который определяет взаимодействие объектов класса с внешней средой.

Свойства представляют собой характеристики (атрибуты), присущие объектам (например, размер шрифта, название и др.).

Инкапсуляция. Объект, с одной стороны обладает определенными свойствами, которые характеризуют его состояние в данный момент времени, а, с другой стороны, над ним возможны операции, которые приводят к изменению этих свойств. Таким образом, объект объединяет в себе как описывающие его данные (свойства), так и средства обработки этих данных (методы).

Инкапсуляцией называется объединение в объекте его свойств и возможных над ним операций (методов). В ООП используется определенная форма записи при работе с объектами. Вызов метода выполняющего действия над тем или другим объектом, записывается следующим образом: ОБЪЕКТ. МЕТОД.

Лекция 8. Алфавит и основные операторы языка бейсик

Содержание лекции: общие сведения об алфавите и основных операторах языка бейсик.

Цель лекции: изучить алфавит и основные операторы языка бейсик.

Назначение языка бейсик (BASIC). Наименование BASIC является сокращением английских слов «Beginner All-purpose Symbolic Instruction Code», что в переводе означает «многоцелевой язык символических команд для начинающих». BASIC был задуман разработчиками как язык, доступный для усвоения студентами гуманитарных вузов. Однако простота грамматики и синтаксиса, его схожесть с фортраном привели к широкому распространению бейсика в практике математических и научно-технических расчетов.

Бейсик — язык интерпретирующий. Работа на бейсике организуется с помощью специальной программы-интерпретатора, которая записана в ПЗУ или загружается в ОЗУ с внешнего накопителя — магнитной ленты или диска. Эта программа переводит каждый оператор бейсик-программы на машинно-ориентированный язык конкретной ЭВМ.

Бейсик — язык диалоговый. Это означает, что программными средствами на бейсике можно организовать диалоговый режим работы ПЭВМ с пользователем. При этом ПЭВМ сообщает пользователю о том, какими возможностями она обладает, как ею пользоваться, в каком порядке должна выполняться данная конкретная программа и т. д. Диалог задается пользователем и в его программах.

Алфавит бейсика содержит 26 прописных латинских букв, цифры от 0 до 9, точку, запятую, кавычки, знаки арифметических операций (+ , —, /, * (умножение) и ~\ (возведение в степень)), знаки отношения двух величин (> , <, =), скобки и другие специальные знаки. Версии бейсика для отечественных микроЭВМ имеют в алфавите и русские буквы. Переменные бейсика делятся на 4 типа.

1) Простые переменные, обозначаемые латинскими буквами, например, A, B, C, . . ., X, Y, Z.

2) Простые переменные, обозначаемые латинскими буквами с одной цифрой после них, например, Al, D2, . . ., Z9.

3) Индексированные переменные одномерных массивов вида A (a), B (a).....Z (a), где а — число, простая переменная или арифметическое выражение.

4) Индексированные переменные двумерных массивов вида A(a, p), B(a,P), ..., Z (a, p), где а и р — числа, простые переменные или арифметические выражения.

Максимальные значения а и р зависят от объема ОЗУ (обычно 256).

Примеры. X(8) — переменная одномерного массива X с номером 8, Y(2,3) — переменная двумерного массива, стоящая в 3-й строке и 4-м столбце (нумерация строк и столбцов начинается с 0), Z (I, J·K) — переменная двумерного массива, номер строки которой задается значением целой части переменной I, а номер столбца — значением целой части произведения J·K.

Числа могут вводиться в программу в качестве вещественных констант либо вводиться с пульта для задания числовых значе-ний переменных. Числа задаются в естественном виде с разделительной точкой, а не запятой. Нуль перечеркивается наклонной чертой, что позволяет отличить его от буквы О. Порядок числа указывается после буквы Е, например, число 1,234 можно ввести в виде 1.234; .1.234E1; 123.4E —2 и т. д. Перед мантиссой или порядком могут стоять знаки+ и —, обычно знак + опускается. Например, число —0,000097 записывается в виде — 0.000097; —.000097; — 97E—6; — 9.7E — 05 и т. д. Первый нуль мантиссы обычно можно не вводить. Максимальный порядок чисел зависит от типа ЭВМ и лежит в пределах от 30 до 252. Операторы бейсика — допустимые предписания, предназначенные для задания ЭВМ операций обработки информации. Операторы записываются в виде слов или предложений большими латинскими буквами без пробелов в словах (у некоторых ПЭВМ пробелы игнорируются). Существуют также служебные операторы (директивы), исполняемые сразу после их ввода с пульта (некоторые из них могут выполняться и по программе). Программа на бейсике есть последовательность операторов, записанных по определенным правилам и обеспечивающих решение заданной задачи на ЭВМ. В бейсике операторы записываются построчно. В одной строке может быть один оператор или несколько, разделенных специальным знаком (двоеточием : или наклонной чертой \). Строки должны иметь последовательную нумерацию. Рекомендуется задавать номера строк с пропусками, например, 5, 10, 15, 20 и т. д. или 10, 20, 30 и т. д. Это позволит при необходимости вставить между ними новые строки. Максимальный номер строки зависит от типа ПЭВМ.

Операторы задания массивов DIM и COM.

Одномерные и двумерные массивы занимают в ОЗУ ЭВМ определенную область памяти, обозначаемую как DIM или COM. Чтобы в эту область не

попали другие данные, перед вводом и использованием массивов нужно зарезервировать для них место. Это делается с помощью операторов DIM и COM, после которых указываются переменная и скобки с указанием размеров массива. Так, полный оператор DIM A (9) резервирует в памяти место для одномерного массива A из 10 переменных от A(0) до A(9). Оператор DIM C(5, 6) резервирует место под двумерный массив C, содержащий шесть строк и семь столбцов (включая нулевые).

Операция	Обозначение	Пример	Результат
Сложение	+	2+5	7
Вычитание	-	10-8	2
Умножение	*	3*4	12
Деление	/	15/3 15/4	5 3.75
Целочисленное деление	\	15\4	3
Возведение в степень	^	2^3	8
Остаток от деления	MOD	13 MOD 5	3

Рисунок 8.1 - Символы арифметических операций Бейсика

Для обозначения размерности массивов можно использовать арифметические выражения (причем их дробная часть отбрасывается), а также числовые значения переменных. Например, оператор DIM R (I·N, K) резервирует область памяти DIM под двумерный массив R с максимальным номером строки, равным целой части произведения I·N и максимальным номером столбца, равным целой части числового значения переменной K.

Операторы выполнения математических функций на бейсике приведены на рисунке 8.2. Аргумент функции должен заключаться в скобки. Им могут быть числа, переменные и арифметические выражения, например:

SIN(30) - Синус угла в 30 градусов.

SIN(Z) - синус угла, заданного аргументом в виде числового значения аргумента Z.

EXP(SQR(2)) – значение числа e: $e=\exp^{\sqrt{2}}$

Аргумент и значение функций не должны выходить за пределы определения функции и за пределы разрядной сетки ЭВМ, иначе будет диагностироваться ошибка. Вычисление функции имеет приоритет перед арифметическими операциями.

Исполняемые по программе основные операторы версии бейсика для микроЭВМ Электроника-60, Электроника-ДЗ-28 даны на в таблице 8.2. Рассмотрим их более детально.

Оператор LET (от слова letter — буква) служит для присвоения буквенным переменным, указанным слева от знака равенства, числового значения константы, переменной или арифметического выражения, записанных справа от знака равенства.

Оператор DATA (от слова data — данные) служит для ввода в память ЭВМ определенной последовательности чисел. Они даются списком с разделительными запятыми.

Основные операторы приведены в таблице 8.3.

Таблица 8. 2 - Операторы вычисления математических функций

Запись	Запись	Наименование	Запись	Запись	Наименование
оператора	функции	функции	оператора	функции.	функции
SIN(X)	sin x	Синус X	RND (x)	rnci x	Генерация случайных
COS(X)	cos x	Косинус X			чисел со значением от С
AT\(X)	arctg*	Арктангенс X			до 1
SQR(X)	"V*	Корень квадратный из X	SGN(X)	sign x	Сигнум-функция (4-1
EXP(X)	e^x= exp x	Экспонента X			и — 1 при лг<0)
LOG(X)	Ln x	Натуральный логарифм X	FN a(X)	—	Функция, заданная
ABS(X)	abs x	Абсолютное значение X	EXF(X)	—	Внешняя функция
INT(X)	int x	Целая часть X			

Таблица 8.3 -Основные операторы языка бейсик

Операторы	Примеры записи	Пояснения
LET	LET A = 2 LET X = Y LET Z = a	Переменной A присваивается значение 2. Переменной X присваивается значение переменной Y. Переменной Z присваивается значение арифметического выражения *a*.

DIM COM	DIM A(5) COM A(I,J) DIM A(a,L)	Задание одномерного массива A из шести переменных A(0), A(1), ..., A(5). Задание двумерного массива с максимальным номером строки I и столбца J. Задание двумерного массива с максимальным номером строки, равным целой части арифметического выражения *а* и столбца с максимальным номером, определяемым численным значением переменной L.
DATA	DATA 1, 2, 3, 4	Ввод чисел 1, 2, 3, 4
READ	READ A, B, C,D	Присвоение переменным A, B, C и D численных значений, заданных оператором DATA.
RESTORE	RESTORE	Поворот стека данных.
INPUT	INPUT A INPUT A = A INPUT A, B, C	Ввод значения A с пульта после появления знака ?. Ввод значения A после печати пояснения A= ... Ввод значений A, B и C, разделенных запятыми.
PRINT	PRINT 2 PRINT A, B, C PRINT A = A PRINT F = a	Печать числа 2 Печать численных значений переменных A, B и C. Печать численного значения A после комментария A =. Печать численного значения арифметического выражения a после комментария F =.
GOTO	GOTO я	Безусловный переход к строке с номером п.
IF ... THEN IF...GOTO	IF a f (5 THEN n IF a r p GOTO n IF a r p THEN 7	Условный переход к строке *n*, если справедливо отношение *г* Выполнение операций 7, если справедливо отношение *г*.
FOR ... TO ... STEP ... NEXT	FOR a = p TO 7 STEP 6 NEXT a FOR a = p TO 7 NEXT a	Организация цикла с управляющей переменной a, меняющейся от начального значения p до конечного 7 с шагом 6 (P, 7 и 6 — числа, значения переменных или арифметических выражений). Организация цикла с шагом 6= + 1.
GOSUB	GOSUB n	Обращение к подпрограмме, начинающейся со строки *n*.
.RETURN	RETURN	Возврат из подпрограммы.
Операторы	Примеры записи	Пояснения.
REM (REMARK)	REM КОММЕНТАРИЙ	Печать комментария при выводе листинга программы.
STOP	STOP	Остановка вычислений в любом месте программы.
END	END	Конец программы и останов в конце программы.
DEFFN<V> (параметр) =a	DEF FN Y(X)=SIN (X)/X	Задание функции a пользователя; в примере Y (X) = sin x/x.

Лекция 9. Операторы языка Бейсик

Содержание лекции: дать сведения об операторах языка Бейсик.
Цель лекции: изучить операторы языка Бейсик.

Оператор READ (read — читать) служит для чтения чисел из блока данных DATA и присвоения переменным своего списка численных значений.

Оператор RESTORE (от слова restore — возвращать) служит для поворота стека данных, который формируется в ОЗУ ЭВМ оператором DATA (стек можно уподобить обойме револьвера, если числа отождествлять с патронами). Этот оператор необходим, если, например, одно из чисел в списке DATA нужно присвоить разным переменным. В приведенной программе вначале переменным стека оператора READ в строке 20 присваиваются отмеченные выше численные значения. Затем оператор RESTORE в строке 30 обеспечивает поворот (возврат) стека. В результате переменной F присваивается значение 1, а переменной L — сначала значение 2, а затем 3. Оператор INPUT (input — ввод) служит для ввода чисел с пульта по запросу ЭВМ. Он сопровождается печатью знака вопроса (?) или комментария, заключенного в апострофы (или кавычки), после слова INPUT, а также для присвоения числовых значений списку переменных.

Примеры.

1. 10 INPUT A.

Печатается знак вопроса, и ЭВМ ждет число, значение которого (после ввода) будет присвоено переменной A.

2. 20 INPUT A, B, C, D.

Печатается знак вопроса, после чего надо ввести четыре числа, разделяя их запятыми.

3. 30 INPUT ВВЕДИТЕ ЧИСЛО УРАВНЕНИЙ N = N.

Печатается запрос ВВЕДИТЕ ЧИСЛО УРАВНЕНИЙ N = , после которого надо ввести значение N.

4. 10 INPUT ВВЕДИТЕ A, B, A, B.

Печатается указание ВВЕДИТЕ A, B, после чего следует ввести - значения A и B, разделив их запятой.

У некоторых ПЭВМ, например, Электроника-60, встречаются версии бейсика, у которых применение комментария после оператора INPUT не предусмотрено. В этом случае для вывода комментария используется оператор PRINT.

Оператор PRINT (от слова print — печать) служит для задания вида печати (специальным знаком после него), печати комментариев, заключенных в апострофы (или кавычки), печати числовых значений констант, переменных и арифметических выражений, входящих в список этого оператора.

Примеры.

1. 50 PRINT 125, 98, 36.

Печатаются числа 125 98 36.

2. 50 PRINT A, B, C.

Печатаются числовые значения переменных A, B, C. Так, если A= 125, B = 98 и C = 36, то печать имеет вид, аналогичный приведенному выше.

3. 50 PRINT ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ.

Печатается комментарий ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

4. 100 PRINT A = A2.

Печатаются комментарии и значение переменной A2. Если A2= 125, печатается A= 125.

5. 60 PRINT F = SQR (A*B+C).

Печатается F=a, где a — числовое значение арифметического выражения VAB~+CT.

6. 70 PRINT.

Печатается пустая строка (т. е. пробелы).

Если в списке оператора PRINT в качестве разделителя используется запятая, то каждое число (см. примеры 1 и 2) печатается в соответствующей зоне одной строки. Последняя разделяется обычно на пять зон печати. Если чисел больше пяти, то последующие печатаются с новой строки, и т. д. Для более плотной печати в качестве разделителя можно использовать знак ; (точка с запятой). Тогда очередное число будет следовать за предыдущим с минимальным пробелом.

Тип печатающего устройства указывается специальными знаками после оператора PRINT: например, для системы подготовки программ на базе ПЭВМ Электроника-ДЗ-28 оператор PRINT после включения системы обеспечивает вывод комментариев и чисел только на дисплей. Для печати принтером после оператора PRINT следует поставить знак #1, а для обратного переключения индикации на дисплей — знак #0.

Пример.

1. 10 PRINT#0ВЫЧИСЛЕНИЕ ЭКСПОНЕНТЫ 20 INPUT X

2. 30 PR I NT# 1 X= К, EXP < X > = EXP < X > 40 END

В этом примере на экран дисплея будет выведено сообщение ВЫЧИСЛЕНИЕ ЭКСПОНЕНТЫ, а значения X и EXP(X) будут напечатаны принтером. Таким образом, первоначально введенный знак =#=0 или =#=1 сохраняется до замены его другим, повторять этот знак в последующих операторах PRINT не нужно.

Оператор GOTO n (от слов go to — идти к) обеспечивает безусловный переход к строке с номером *n*. В программе:

10 INPUTX=X 20 LETV=SIN<X) 25 PRINTV=V 30 GOTO 10

после ввода X и вычисления Y = sinX оператор GOTO 10 в строке 30 обеспечивает переход к строке 10, т. е. к вводу нового значения X по оператору INPUT. Таким образом, программа зацикливается.

Операторы IF и THEN (if — если, then — тогда) служат для организации условных переходов.

Таблица 9.1 - Символы отношения г для операторов IF...THEN

Символ бейсика	Математический символ	Пример записи оператора	Значение символа
=	=	A = B	Равно
<	<	A<B	Меньше
< =	<	A< = B	Меньше или равно
>	>	A> B	Больше
= >	>	A=> B	Больше или равно

Пример.

```
1. 10  INPUTX=X
2. 20  IF X=0 THEN 50
3. 30 LETF=SIN<X)/X
4. 40 PRINT1F=FsGOTO 10
5. 50 LETF=l:60T0 49
```

В приведенной программе вычисляется функция F = siriA;/x. При x = 0 получаем неопределенность вида 0/0, а поскольку деление на 0 машина не производит, нужно этот случай исключить. Для этого в строке 20 значение x сравнивается с нулем. Если x = Q, происходит переход к строке 50, где переменной F присваивается известное значение 1. Операторы организации циклов FOR ... TO и STEP образуют цикл по схеме:

FOR x = p TO v STEP fi (тело цикла)

NEXT a

Эта запись означает следующее: для (for) переменной a, имеющей начальное значение p и меняющейся до (to) конечного значения v с шагом (step) fi, обеспечивается выполнение цикла, завершаемого оператором NEXT a (следующий). Если a достигает значения v, происходит выполнение следующего за циклом оператора. Переменная a должна быть простой, a, p, y и v могут быть числами, переменными или значениями арифметических выражений.

Лекция 10. Программирование повторений

Содержание лекции: дать сведения о программировании повторений.

Цель лекции: научить писать программы, используя операторы повторений.

При решении задач приходится многократно вычислять значения по одним и тем же зависимостям для различных значений входящих в них величин.

Такие многократно повторяемые участки вычислительного процесса называют циклами.

В языке Visual Basic 6 циклы - это многострочные операторы двух видов:

- циклы со счетчиком (рисунок 10.1);
- циклы с условием.

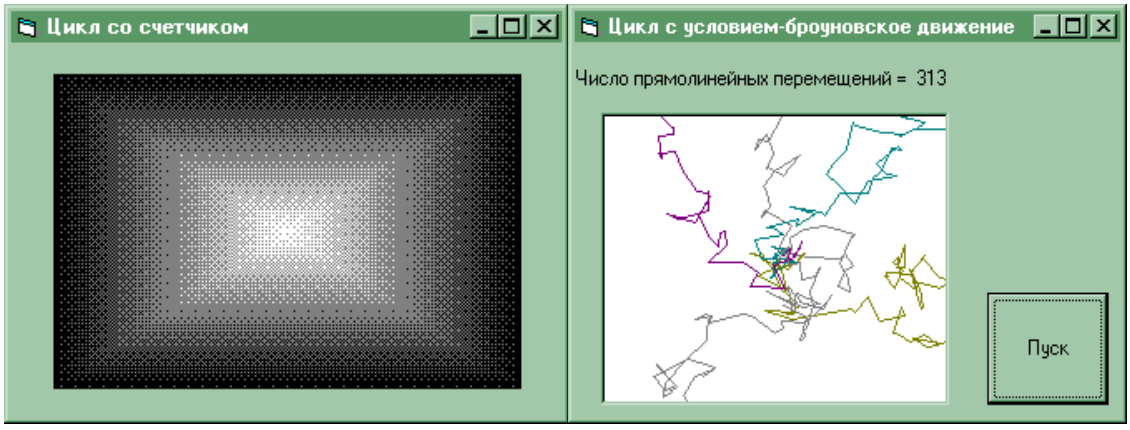


Рисунок 10. 1- Окно цикла со счетчиком

Программирование повторений с помощью цикла со счетчиком.

Для того чтобы организовать повторения с заданным их числом, применяют оператор цикла For…Next. Этот оператор называется циклом со счетчиком.

Это многострочный оператор, его синтаксис таков:

```
For Имя = Значение1 To Значение2 [Step Значение3 ]
    ПовторяющиесяОператоры
Next [Имя]
```

где Имя - это имя переменной, которую называют счетчиком;

Значение1 - начальное значение счетчика;

Значение2 - конечное значение счетчика;

Значение3 - величина, на которую изменяется значение счетчика при одном повторении. Она называется шагом цикла;

Повторяющиеся Операторы - это часть программы, которая должна повторяться один или несколько раз.

Пример 1. Процедура для щелчка командной кнопки с именем Command1.

```
Private Sub Command1_Click()
 For i=0 To 100 Step 20 ‘ Первая строка цикла
   Print i
 Next i                    ‘Последняя строка цикла
End Sub
```

Здесь i - имя счетчика, 0 и 100 - начальное и конечное значения счетчика, 20 - шаг цикла, а последовательность операторов состоит из одного оператора Print i - печати значения счетчика на экранной форме.

Результат работы программы: 0, 20, 40, 60, 80, 100.

Что произойдет, если начальное значение счетчика будет равно 1?

Циклы со счетчиком удобно применять для построения графиков функций. График функции строится на каком-либо отрезке оси абсцисс «по точкам», когда для каждого значения аргумента x на этом отрезке можно легко найти значение y этой функции и поместить на плоскость чертежа точку с координатами (x,y). Точек можно брать много - при этом они будут сливаться в одну сплошную линию - график функции.

Пример 2. Построить график функции

$y=ax^2+bx+c$

(a=1, b= -2, c= -3, x от -2 до 4 с шагом 0,01).

На рисунке показана экранная форма. На нем с помощью цикла со счетчиком и метода PSet построен график. (рисунок 10.2).

Кроме самого графика, с помощью метода Line изображены оси координат, рядом с которыми напечатаны числа (также с помощью циклов со счетчиком и метода Print).

Программный код, содержащий процедуру построения данного графика следующий.

```
Private Sub Command1_Click()
Picture1.Scale (-3, 5)-(5, -5)
        'Построение графика:
For x = -2 To 4 Step 0.01
y = x ^ 2 - 2 * x - 3
Picture1.PSet (x, y), vbGreen
Next x
        'Ось абсцисс:
Picture1.Line (-3, 0)-(5, 0), vbCyan
For i = -3 To 4
Picture1.PSet (i, 0), vbCyan
Picture1.Print i
Next i
        'Ось ординат:
Picture1.Line (0, -5)-(0, 5), vbCyan
For i = -4 To 5
Picture1.PSet (0, i), vbCyan
Picture1.Print i
Next i
End Sub.
```

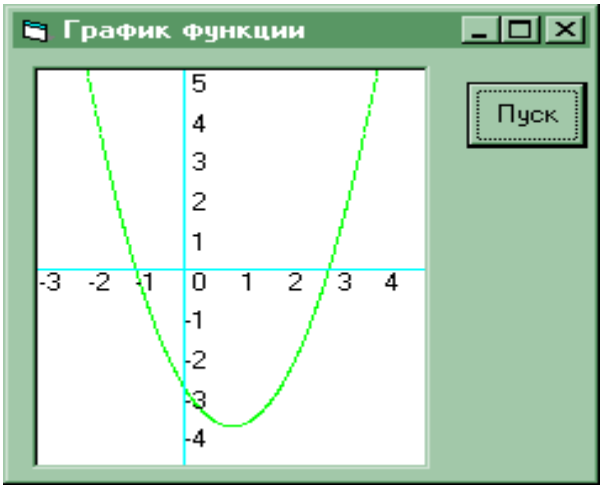


Рисунок 10.2 - Функция «График»

Вложенные циклы.

В цикл, называемый внешним, могут входить один или несколько вложенных циклов, называемых внутренними. Организации как внешнего, так и внутреннего цикла осуществляются по тем же правилам, что и простого цикла. Параметры внешнего и внутреннего циклов разные и изменяются не одновременно, т.е. при одном значении параметра внешнего цикла параметр внутреннего цикла принимает поочередно все значения. Для программирования вложенных циклов используются конструкции типа:

```
For ИмяА = ЗначениеА1 To ЗначениеА2 [Step ЗначениеА3]
  For ИмяБ = ЗначениеБ1 To ЗначениеБ2 [Step ЗначениеБ3]
    ПовторяющиесяОператоры
  Next [ИмяБ]
Next [ИмяА]
```

где ИмяА и ИмяБ - это индексы двух циклов (внешнего цикла).

Работают эти циклы следующим образом: при каждом значении счетчика внешнего цикла А счетчик внутреннего цикла Б «пробегает» все свои значения. И при каждом значении счетчика внутреннего цикла Б выполняются Повторяющиеся Операторы.

Пример 3. Написать программу, которая печатает таблицу чисел.

1	2	3	4	5
1	4	9	16	25
1	8	27	64	125
1	16	81	256	625

В первой строке - натуральные числа. Во второй строке их квадраты, в третьей - их кубы и т.д.

Код программы:

```
Private Sub Command1_Click()
Dim i As Integer, j As Integer
ni = 4: nj = 5: Cls
```

```
For i=1 To ni     ‘ Первая строка внешнего цикла
  For j=1 To nj   ‘ Первая строка внутреннего цикла
      Print j^i,
  Next j          ‘ Последняя строка внутреннего цикла
  Print
Next i            ‘ Последняя строка внешнего цикла
End Sub.
```

Здесь i - счетчик строк таблицы, а j - счетчик столбцов, j^i - возведение номера столбца в степень, равную номеру строки, ni и nj - число строк и столбцов таблицы.

Cls - метод, который применяется к экранной форме и служит для ее очистки от того, что на ней было напечатано ранее.

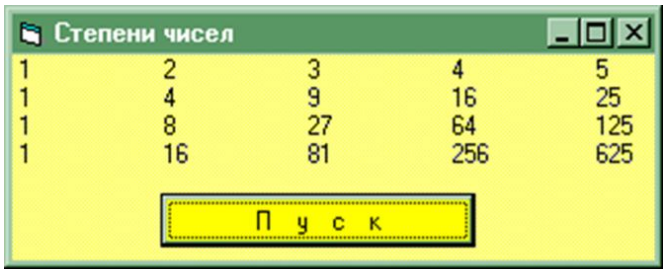


Рисунок 10.3 - Введенная таблица чисел

Программирование повторений с помощью цикла с условием.

Цикл с условием - это многострочный оператор, первая строка которого начинается с ключевого слова Do (Выполнить), а последняя строка начинается с ключевого слова Loop (Петля).

Синтаксис цикла с условием имеет две формы.

Форма 1:

Do Условие

Повторяющиеся Операторы

Loop

Форма 2:

Do

Повторяющиеся Операторы

Loop Условие

Условие бывает двух типов:

- с ключевым словом While - в этом случае оно называется условием продолжения цикла (условием типа While);

- с ключевым словом Until - в этом случае оно называется условием завершения цикла (условием типа Until).

После ключевого слова While (или Until) записывается Условное Выражение.

Возможны 4 варианта:

1	2	3	4
Do While…	Do Until…	Do	Do
…	…	… .	.....
Loop	Loop	Loop While	Loop Until

Рассмотрим логику работы цикла с условием:

Случай 1. Использование условия типа While - продолжения цикла. Повторяющиеся Операторы выполняются, если значение Условного Выражения равно True. В противном случае цикл завершается.

Случай 2. Использование условия типа Until - завершения цикла. Повторяющиеся Операторы выполняются, если значение Условного Выражения равно False. В противном случае цикл завершается.

Пример 4. Вы хотите, чтобы цикл завершился при выполнении следующего условия:

X > 100 Or Time > 600 .

Это равносильно тому, что цикл должен продолжаться, пока выполняется условие

Not (X > 100 Or Time > 600),

что равносильно условному выражению

X <= 100 And Time <= 600.

Какое из условий использовать

Until X > 100 Or Time > 600 или

While X <= 100 And Time <= 600

Вы выбираете по желанию.

Рисунок 10.4 - Схема ввода чисел

Об использовании оператора GoTo.

Использование безусловного перехода при программировании повторений в наше время считается «дурным тоном». Объясняется это тем, что эти операторы запутывают программу, делают ее трудночитаемой. Такая программа напоминает тарелку с итальянскими макаронами - найти оба конца одной макаронины очень трудно!

В 60-е годы были провозглашены принципы структурного программирования, которые запрещают программистам использовать

оператор GoTo. Эти принципы можно сформулировать так: надежность программы, быстрота ее разработки, а также простота ее модификации (изменения) напрямую зависят от ее логичности и понятности. А чтобы программа была логичней и понятней, нужно отказаться от широкого использования глобальных переменных, оператора GoTo и разбивать ее на отдельные не слишком большие части - модули, каждый из которых независим от других модулей и взаимодействует с ними по строго определенным правилам.

Почему же оператор GoTo все же есть в VB ?

Иногда он удобен для отработки программы сбоев. Для того, чтобы программа не «зависла» в случае ошибки, а продолжала работать, используется особый оператор языка VB, который называется оператором перехвата ошибки:

On Error GoTo Метка

Метка находится там, куда необходимо перейти в случае ошибки. Например, после нее может стоять оператор, выдающий сообщение об ошибке пользователю. Получив такое сообщение, пользователь может исправить ошибку без остановки программы.

В остальных случаях использовать оператор GoTo не следует (можно, но не нужно).

Использование цикла с условием таит в себе одну опасность - попасть в бесконечный цикл.

Бесконечный цикл - это повторяющийся процесс, в котором число повторений ничем не ограничено.

Если в бесконечный цикл попала работающая программа, то говорят она «зависла». Для выхода из этой ситуации можно нажать клавиши Ctrl+Break.

Если в бесконечный цикл попала исполняемая программа, то для выхода иногда требуется перезагрузка компьютера.

Функция Окно сообщения MsgBox.

Синтаксис функции:

MsgBox (Текст [, Опция] [, Заголовок])

или

MsgBox Текст [, Опция] [, Заголовок],

где Текст - это строка сообщения, которая может содержать

1024 символов.

16 - критическое сообщение,

32 - вопрос,

48 - предупреждение,

64 - информация.

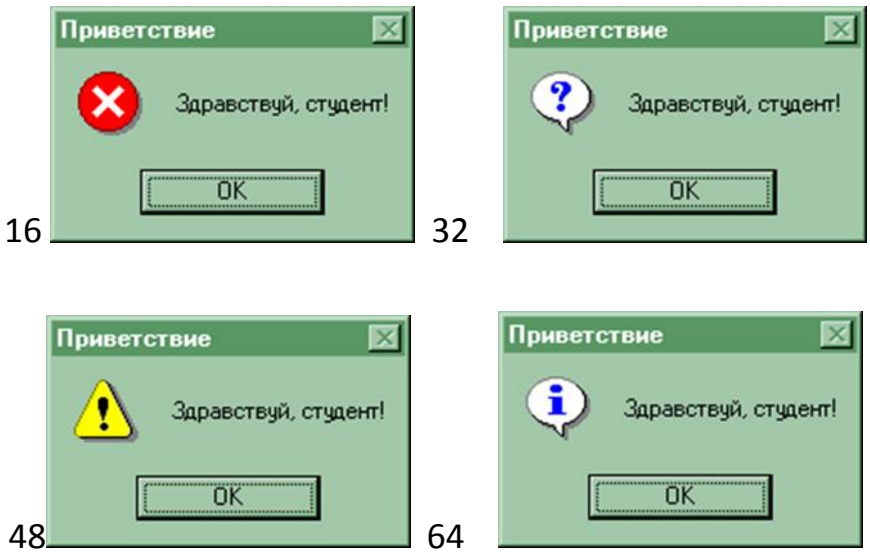


Рисунок 10.5 - Виды сообщений

Лекция 11. Работа в системе Visual Basic

Содержание лекции: знакомство с системой Visual Basic.
Цель лекции: дать первые навыки в работе с Visual Basic.

Причины появления Visual Basic.

В начале 60-х годов в Dartmouth College был разработан язык программирования BASIC – *Beginner's All purpose Symbolic Instruction Code* – универсальный символьный программный код для начинающих.

Самые первые языки программирования (50-е годы) предназначались, главным образом, для решения сложных математических задач и разобраться в них было достаточно сложно по причине их сильной приближенности к аппаратным ресурсам компьютера. Расширение сферы применения ЭВМ и круга людей, работающих с ними, привело к упрощению языков программирования. Эти же условия потребовали перехода на иной, качественно новый, уровень взаимодействия человека с ЭВМ. В частности, широкое распространение получили графические методы взаимодействия. Операционная система Microsoft Windows имеет графический, интуитивно понятный для пользователя интерфейс. Однако это значительно усложняет процесс написания программ. В связи с этим развитие получили визуальные языки программирования, которые значительно упрощают создание приложений.

На современном этапе развития общества от программиста требуется быстрота и качество написания программного кода с сохранением требований стандартизации. Среди *основных требований к языкам программирования* можно выделить следующие:

- необходимость поддержки общепринятого стандарта взаимодействия с пользователем (графического интерфейса);
- наличие средств поддержки ускоренной разработки приложений;
- малый результирующий объем программного (исполняемого) кода;
- переносимость на другие платформы.

Описание языка Visual Basic.

Событийно-управляемое программирование.

В 1991 г. фирмой Microsoft был разработан и выпущен Visual Basic**.** Система программирования Microsoft Visual Basic for Windows, обладая простыми в обращении средствами визуального проектирования, позволяет в полной мере использовать преимущества графической среды Windows и быстро конструировать эффективные приложения. Существуют и другие визуальные языки программирования с подобным интерфейсом (Borland Delphi, Borland C++ Builder).

Visual Basic – один из первых языков, поддерживающих событийно-управляемое программирование (*event-driven programming*).Этот стиль хорошо согласуется со стандартом GUI (*Graphical User Interface*)**.** Традиционно программирование ориентировалось на поэтапное описание конкретного процесса. Однако современные компьютерные приложения слишком сложны и данный стиль программирования для них не подходит.

Смысл событийно-управляемого программирования заключается в том, что вместо скрупулезного описания каждого шага необходимо лишь указать, как реагировать на различные события (действия пользователя): выбор команды, щелчок или перемещение мыши и пр. В результате программист создает не одну большую программу, а приложение, состоящее из набора взаимодействующих процедур, управляемых пользователем.

Экранные элементы.

После запуска Visual Basic на экране появляются окно, показанное на рисунке 11.1. О*сновное окн состоит* из трех частей:

Заголовок (Microsoft Visual Basic).

Строка меню (File, Edit, View, Run, Debug, Window, Help)**.**

Панель инструментов (Toolbar), на которой расположены кнопки быстрого вызова часто используемых команд.

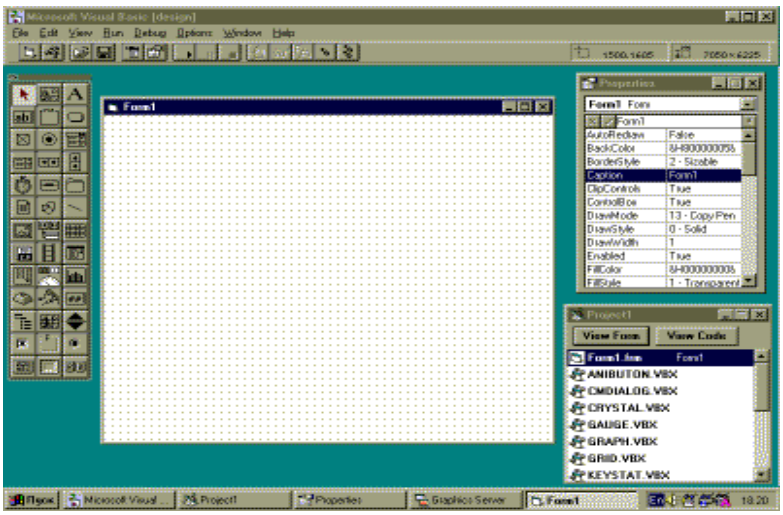


Рисунок 11.1 - Основное окно

В центре экрана – большое пустое окно формы с заголовком *Form1*. Слева от него – окно шаблонов или окно инструментария (Toolbox). Справа – окно свойств (Properties), а под ним – окно проекта (Project).

1. *Основное окно. Панель инструментов (Toolbar).*

Кнопки панели инструментов предназначены для быстрого вызова часто используемых команд.

New Form. Эта кнопка используется для создания новой формы. Приложение может содержать более одной формы. Определение формы и весь связанный с ней код хранятся в отдельном файле с расширением *.FRM. Если приложение состоит из нескольких форм, то соответственно имеются несколько файлов с этим расширением.

New Module. Создание модуля кода. Код размещается в файлах с расширением *.BAS. На отдельные модули кода программу разбивают, главным образом, чтобы упорядочить ее внутреннюю структуру: блоки кода, отвечающие за работу определенных компонентов программы, обычно помещают в отдельные модули. Глобальные объявления в любом модуле кода распространяются на всю программу. А переменные, объявленные в форме, доступны лишь внутри модуля формы.

Open Project. Кнопка открытия проекта. В файле проекта содержатся имена всех файлов форм и некоторые дополнительные сведения о проекте. Файл проекта имеет расширение *.MAK.

Save Project. В диалоговом окне *Save File As* надо задать имя файла и каталог, в котором будут храниться код и компоновка формы. Далее Visual Basic откроет диалоговое окно *Save Project As*, позволяющее сохранить файл проекта. По умолчанию, файлу проекта предлагается присвоить имя Project1.MAK.

Menu Design Window. Окно конструктора меню.

Properties Window. Окно задания свойств объектов.

Start. Кнопка запуска приложения на выполнение.

Break. Кнопка прерывает выполнение программы и активизирует окно отладки *Debug*.

End. Кнопка остановки выполнения приложения.

Toggle BreakPoint. Кнопка простановки (убирания) точек прерывания в целях отладки программы. Строка выделяется инверсной подсветкой, указывающей на наличие точки прерывания.

Instant Watch. Просмотр значения выражения (переменной) в целях отладки.

Calls. Показ списка вызванных и активных в данный момент процедур.

Single Step. Пошаговое выполнение программы при нажатии на кнопку. Вложенные процедуры также выполняются в пошаговом режиме. Аналогичного эффекта можно достичь, нажимая клавишу F8.

Procedure Step. Выполнение программы по процедурам (или Shift-F8).

2. *Окно проекта (Project).*

Окно проекта содержит список всех файлов, необходимых для выполнения создаваемого приложения. При запуске Visual Basic в этом окне уже присутствуют несколько компонентов. Первому компоненту присвоено имя Form1.frm, т. е. открыт файл Form1.frm, связанный с формой.

Остальные файлы с расширением .VBX – расширение файлов системы программирования Visual Basic. В них содержатся дополнительные инструментальные средства. При загрузке файла они включаются в окно инструментария (Toolbox). Каждый файл типа *.VBX в окне проекта позволяет ввести в окно инструментария один или несколько новых элементов.

В окне проекта имеются еще две кнопки: View Form и View Code. Щелчок кнопок открывает окно с просмотром формы и кода соответственно.

Создание программы в Visual Basic состоит из двух взаимосвязанных процессов: конструирование формы (создание интерфейса программы), с которой пользователь будет вести диалог, и кодирования, определяющего характер работы программы.

3. *Окно формы и окно шаблонов.*

Форма – это окно будущего приложения. На первом рисунке она расположена в центре экрана. При создании нового проекта Visual Basic открывает пустую форму и присваивает ей заголовок Form1. Форма служит своеобразным холстом, на котором размещаются различные части программы – объекты или элементы управления (и саму форму Visual Basic рассматривает как объект).

Элементы управления создаются на основе шаблонов, кфждый из которых представлен в окне инструментария (Toolbox) своим значком

(иконкой). Большая часть инструментов встроена в Visual Basic. Кроме того, окно инструментария можно расширить путем подключения дополнительных элементов: каждый файл типа *.VBX в окне проекта позволяет ввести в окно Toolbox один или несколько новых элементов.

Список элементов Toolbox (стандартный).

Pointer (указатель). Активизация этого элемента означает, что можно редактировать форму: добавлять новые объекты, перетаскивать их на другие места или изменять размеры объектов, используя один из маркеров.

Picture Box (окно рисунка).

image Box (окно изображения). Использование этих инструментов позволяет размещать графическую информацию в определенных участках формы. Окна рисунков лучше подходят для динамической среды (например, анимационные эффекты). Окна изображений удобнее использовать в статике, т. е. когда созданную картинку не предполагается изменять.

Label (метка). Метки – это поля, заполняемые программистом текстовой информацией и недоступные пользователю для редактирования. Содержимое метки определяется ее свойством *Caption*. Прямой ввод текста или рисование на метке во время выполнения программы не допускаются.

Text Box (текстовое окно). Текстовые окна – это экранные области, в которые пользователь может вводить текст. Распознать введенный текст можно с помощью свойства *Text*.

Command Button (командная кнопка),

Check Box (флажок), Option Button (переключатель). Эти три объекта с точки зрения программиста практически одинаковы, но для пользователя они отличаются по внешнему виду и по назначению. Щелчок командной кнопки обычно активизирует какую-то операцию, а флажки и переключатели, сообщая текущее состояние того или иного параметра, дают возможность изменять его. Щелчок флажка устанавливает или сбрасывает определенный параметр: если на квадратике флажка значок X, параметр активен (включен). Любой флажок функционально независим от других флажков. Переключатели, напротив, всегда работают в группе и позволяют выбрать какой-нибудь вариант из нескольких возможных. Когда один переключатель активен, остальные отключены.

Frame (рамка). Служит для разделения в окне различных групп объектов. Применительно к переключателям рамки влияют на поведение кнопок. Для остальных объектов рамки выступают в роли визуального разделителя и функции, регулирующей доступ к группе объектов.

List Box (список) – перечень вариантов, выбираемых щелчком мыши. В слишком длинный список, который не выводится на экран целиком,

Visual Basic добавляет линейки прокрутки. Для корректной работы с этим объектом необходимо, чтобы его высота была минимум три строки. Содержимое списка нельзя задать на этапе проектирования. Чтобы ввести в него элемент, необходимо использовать метод *Additem.* Инициализацию списка лучше проводить в процедуре обработки события Load – она принадлежит форме, содержащей этот список.

Combo Box (комбинированный список). Назван так потому, что, объединяя текстовое окно (здесь оно называется полем ввода) с обычным списком, образует единый элемент управления. Однако, в отличие от обычного, комбинированный список не позволяет размещать элементы в несколько колонок.

Horizontal Scroll Bar (горизонтальная линейка прокрутки), Vertical Scroll Bar (вертикальная линейка прокрутки). Действуют совершенно одинаково, только в разных направлениях. Эти объекты позволяют узнавать о позиции движка на линейке и контролировать диапазон ее действия и дискретность перемещения движка.

Timer (таймер). Объект, способный инициировать события через регулярные промежутки времени.

Drive List Box (список дисков), Directory List Box (список каталогов), File List Box (список файлов). Кнопки позволяют создавать и настраивать диалоговые окна, предназначенные для взаимодействия с файловой системой. Каждый из них управляет отдельным компонентом файловой системы. Объект Drive List Box – это раскрывающийся список, содержащий имена всех дисков в системе. Объект Directory List Box выводит на экран список всех подкаталогов в текущем каталоге. Объект File List Box отображает все (или с заданным расширением) файлы в текущем каталоге.

Shape (фигура), Line (линия). Так же, как и окна изображений служат, главным образом, для создания фона в окне формы. Инструмент Line позволяет создавать простые прямые линии. Модифицируя свойства линий, можно изменять их размер, цвет и стиль. Инструмент Shape, по умолчанию, позволяет рисовать только прямоугольники, но, изменив свойство *Shape*, можно вычерчивать окружности, эллипсы и прямоугольники со скругленными углами. Кроме того, допускается подбор цвета и закрашивание замкнутых фигур.

4. *Окно свойств. Свойства объектов.*

Окно свойств имеет вид, показанный на рисунке. Активизировать окно можно щелчком кнопки мыши где-нибудь в окне или выбором из меню Window команды Properties, клавишей F4 или щелчком кнопки *Properties Window* на панели инструментов Toolbar.

Раскрывающийся список в верхней части окна Properties, отражающий имена и типы объектов, включенных в приложение и расположенных на форме, называется *списком объектов.* Изначально здесь содержится лишь информация о форме. Но по мере того как объекты добавляются, Visual Basic регистрирует их в этом списке.

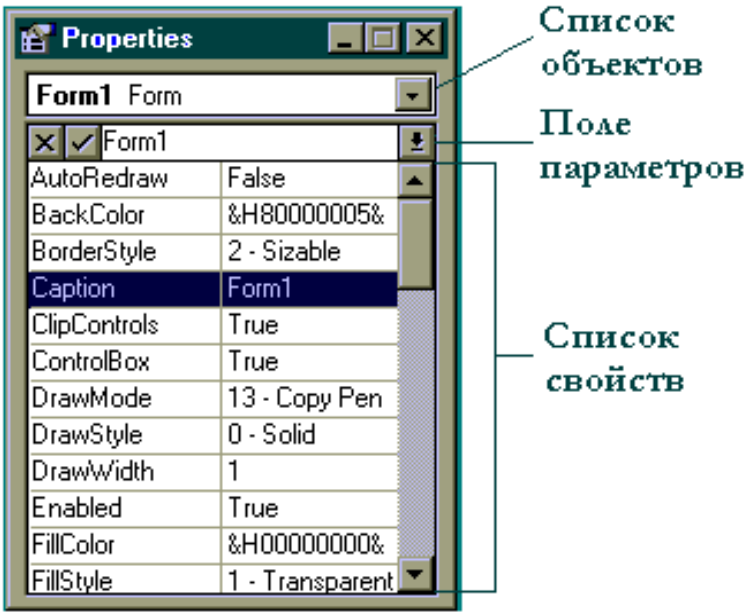


Рисунок 11.2 - Инструменты свойств

Ниже списка объектов следует *поле параметров*. Здесь можно определить конкретное значение для выбранного свойства. На рисунке для свойства Caption формы задано значение Form1. Кнопка с крестиком означает отказ от изменения. Кнопка с галочкой – принятие сделанных исправлений.

Далее следует таблица, называемая *списком свойств*. В левой колонке перечислены свойства выбранного объекта (например, кнопки или списка). В правой колонке следует список его текущих значений.

Понятие свойства (properties) в Visual Basic связано с механизмом формального описания атрибутов объекта. Каждый объект обладает специфическими свойствами, которые определяют внешний вид и поведение объекта в приложении. Некоторые свойства имеют фиксированный набор значений (их можно узнать, нажав первую из кнопок в поле параметров). Если же кнопка справа в этом поле имеет вид с тремя точками, то это означает, что нажатие на кнопку инициирует вывод диалогового окна с выбором необходимого значения. Примерами могут служить выбор цветов, а также выбор файлов для загрузки картинок. Другие свойства могут иметь практически неограниченный набор значений. Это, например, задание имени или названия формы.

Нет необходимости описывать все свойства каждого объекта: значения всех свойств устанавливаются по умолчанию. Окно Properties активно только на стадии проектирования.

Некоторые объекты и их свойства.

На внешний вид и поведение объектов могут оказывать влияние очень многие свойства. Здесь в качестве примера будут перечислены и описаны наиболее характерные.

Форма.

AutoRedraw (автоматическая перерисовка). Это свойство управляет тем, каким именно образом осуществляется повторное создание (восстановление) формы, например, после сворачивания (и последующего развертывания окна) или перекрытия изображения другими окнами. Если свойству присвоено значение True, то происходит автоматическое восстановление изображения. Если False – вызывается процедура обработки события перерисовки и выполняется код, написанный программистом.

BackColor (цвет фона). Значения цвета определяются шестнадцатеричной константой. Щелкнув кнопку с тремя точками в поле параметров окна свойств, можно из палитры цветов выбрать нужный цвет.

Border Style (тип границ окна). Границы окна могут отсутствовать (None), очерчиваться одинарными или двойными линиями, а окно иметь постоянный размер (Fixed Single, Fixed Double), также окно может быть масштабируемо (Sizeable). Выбрать необходимое значение можно из раскрывающегося списка. Изменение этого свойства допускается лишь на этапе разработки приложения.

Caption (название). Это свойство содержит текст, выводимый в строку заголовка окна формы.

Enabled (доступ). Любой объект обладает свойством Enabled. Если значение True, то объект активизирован. Если False – объект неактивен (блокируется воздействие на объект (форму) любых событий, связанных с мышью или клавиатурой).

FontBold, FontItalic, FontStrikethru (полужирный, курсив, зачеркнутый). Эти свойства принимают только булевы значения. Допускается любая комбинация атрибутов шрифта. Модификация свойств влияет лишь на выводимый вновь текст.

FontName (название шрифта). Смена шрифта действует лишь на новый текст, не влияя на уже имеющуюся информацию.

FontSize (размер шрифта). Измеряется в пунктах. Пункт – стандартная типографская единица измерения, равная 0.035 см.

ForeColor (основной цвет). Это цвет текста или графических изображений. Устанавливается с помощью палитры цветов аналогично свойству BackColor.

Height, Width (высота и ширина формы). Определяется в twips. 1 twip=0.0018 см.

Icon (значок). Значок, в который превращается свернутая форма.

Name (имя). Свойство задает имя формы в программном коде, изменять которое в период выполнения программы нельзя. Сравните со свойством *Caption*.

Picture (картинка). Это свойство позволяет сделать так, чтобы в окне всегда отображалась какая-нибудь растровая картинка (bitmap picture). Для изменения свойства в период выполнения программы используется функция LoadPicture.

Top, Left (верхняя и левая координаты окна). Свойства определяют позиционирование формы на экране. Координаты измеряются в twips.

Visible (видимость). Этим свойством обладают многие инструменты. Если оно истинно, то объект (в данном случае форма) видим, если ложно – не видим.

WindowState (состояние окна). Свойство определяет, в каком виде окно отображается на экране: нормальном, свернутом в значок или развернутом.

Для взаимодействия с операционной системой и между собой объекты реагируют на определенные события, происходящие в момент выполнения программы. Полный перечень событий, на которые реагирует конкретный объект, можно узнать в окне кода.

Большинство объектов реагируют на события, связанные с мышью, – Click (щелчок) и DblClick (двойной щелчок). К таким объектам относится и форма. Помимо этого, она реагирует на событие Load (загрузка), которое происходит при загрузке формы (например, в момент запуска приложения). Процедура обработки события Load удобна для инициализации свойств и переменных при запуске программы.

Чтобы отобразить информацию на экране, можно использовать следующие методы.

Cls. Этот метод позволяет очистить форму ото всех изображений и текста. Синтаксис

[имя формы].Cls

LoadPicture. Функция для установки у формы свойства *Picture*. Ее синтаксис

[имя формы].LoadPicture [имя файла]

Print. Этот метод выводит текст на форму:

[имя формы]. Print [[выражение][{;|,}]]…

В конце каждого выражения должна быть запятая или точка с запятой. Если поставлена точка с запятой, то за последним символом предыдущего выражения сразу выводится следующее. Если стоит запятая, – вывод осуществляется в зонном формате (каждая зона имеет размер 14 знакомест). Если Символы «**,**» и «;» отсутствуют, то позиция вывода переводится в начало следующей строки.

Окна рисунков и изображений.

Эти объекты обладают свойствами *Enabled, Height, Left, Name, Picture, Top, Visible, Width* и ряд описанных ранее свойств, поведение которых аналогично поведению свойств формы. Однако, координаты объекта в случае свойств*Height, Left, Top, Width* относительны, а не абсолютны, т. е. они измеряются как смещение от начала координат формы, а не экрана (это касается всех элементов, расположенных на форме). Свойство *Picture* позволяет выводить на экран либо растровую картинку (bitmap), либо значок (icon).

Stretch (подгонка). Это свойство имеют окна изображений. По умолчанию оно False, что означает, что элемент управления (окно) подгоняется под размеры загружаемого в него изображения. Значение True означает, что изображение подгоняется под элемент управления.

Метки.

Alignment (выравнивание). Способ выравнивания текста на метке: по левой границе, по правой и по центру.

AutoSize (подгонка размера). True – размер метки автоматически подгоняется под размер текста. False – метка сохраняет заданный размер, а лишние символы отсекаются.

BorderStyle (тип границ). Метка может либо иметь контур, либо нет.

Caption (название). Свойство содержит текст, отображаемый в поле метки.

Текстовые окна.

MaxLenth (максимальная длина). По умолчанию значение равно нулю, что означает, что в текстовое окно можно вводить любое количество символов. Если значение свойства не равно нулю, то Visual Basic ограничивает количество символов заданной величиной.

MultiLine. Значение True разрешает ввод нескольких строк текста, нажимая Enter. False запрещает ввод более одной строки текста.

Text. С помощью этого свойства программа распознает, какой именно текст введен пользователем. Кроме того, оно позволяет изменять отображаемый на экране текст.

Текстовое окно не реагирует на события, связанные с мышью, но позволяет отслеживать нажатие любой клавиши.

Командные кнопки, флажки, переключатели.

Cancel (отмена). Это свойство связано только с командными кнопками. True – щелчок кнопки эквивалентен нажатию клавиши Escape. Значение True у свойства может иметь только одна командная кнопка на форме.

Default (по умолчанию). Поддерживается лишь командными кнопками. True – щелчок кнопки эквивалентен нажатию клавиши Enter. Значение True у свойства может иметь только одна командная кнопка на форме.

Value (состояние). Значение этого свойства отражает состояние флажка или переключателя. Для переключателя: True – переключатель активен (в центре кружочка жирная точка); False – переключатель неактивен (точка отсутствует). Для флажка: 0 – флажок не помечен, 1 – флажок установлен (в центре крестик), 2 – флажок затенен (имеет блеклый вид).

Списки.

Columns (колонки). Свойство задает возможность размещения элементов списка в несколько колонок. Все элементы списка располагаются в одну колонку, если значение свойства равно нулю. Если *Columns>=1*, то элементы списка размещаются в соответствующем числе колонок. При необходимости в окно списка автоматически добавляется линейка прокрутки.

List (список). Это массив, состоящий из элементов списка. Свойство нельзя модифицировать при проектировании. Доступ к отдельным элементам списка: объект.List(индекс). Индекс первого элемента равен нулю.

Sorted (сортировка). Для списков предусмотрена возможность сортировки (не для русского текста), если свойство принимает значение True.

Text. Свойство содержит текст последнего выделенного элемента списка.

Наиболее распространены следующие методы.

AddItem. Метод включает элемент в список: объект. AddItem текст [, индекс]

Clear. Метод удаляет из списка все элементы: объект. Clear

RemoveItem. Метод удаляет отдельный элемент из списка: объект. RemoveItem индекс.

Линейки прокрутки

Max, Min. Эти свойства задают максимальное и минимальное значение движка на линейке прокрутки. Свойства могут принимать значения целого типа, поэтому находятся в интервале [-32768; 32767]. Когда движок попадает в крайнюю правую или нижнюю позицию, это число присваивается свойству *Value*.

Value (текущая позиция). Свойство отражает текущую позицию движка на линейке. При изменении *Value*программным путем Visual Basic перемещает движок в соответствующую позицию.

Создание меню.

Для разработки меню следует открыть диалоговое окно Menu Design Window (из меню Window команда Menu Design или кнопка Menu Design Window на панели инструментов). В окне для каждого элемента меню следует определить следующие свойства.

Caption (название) – текст, который появится в строке меню. Если перед буквой в названии стоит знак амперсенда, то это означает, что данная буква будет подчеркнута и доступ к пункту меню можно осуществить, используя комбинацию клавиш Alt + Б, где Б – подчеркнутая буква.

Name (имя) – идентификатор элемента в программном коде. В целях избежания совпадения с зарезервированными словами рекомендуется использовать префикс mnu перед именем пункта меню.

Щелкните поле Caption в раскрывшемся окне Menu Design и наберите название пункта меню (например, &Файл). Знак амперсенда & указывает на то, что следующая за ним буква (Ф) будет подчеркнута, и нажатие этой буквы в сочетании с клавишей Alt (т. е. Alt-Ф) обеспечивает быстрый доступ к данному пункту. Затем нажмите клавишу Tab для перехода в поле Name. Введите в него имя пункта меню для обеспечения доступа к нему в программном коде (например, mnuFile) и нажмите клавишу Enter. Название (но не имя) появится в списке, занимающем нижнюю часть окна.

Структура формируемого меню отображается в нижней части окна. Стрелки в списке позволяют перемещаться по пунктам меню. Если щелкнуть кнопку «стрелка вправо», то под &Файл появятся четыре точки, которые сигнализируют, что следующий элемент меню будет размещен на втором уровне. Чтобы вернуться на первый уровень, необходимо нажать кнопку «стрелка влево».

Каждый пункт меню реагирует на событие Click.

Программирование графики.

При различных операциях с окном приложения (масштабирование, перекрытие и сдвиг окон) изображение в окне необходимо обновлять. При подобных операциях Windows генерирует событие Paint. Поэтому для восстановления изображения надо написать процедуру обработки данного события (Form_Paint). В первый раз эту процедуру надо вызвать самостоятельно, а в дальнейшем изображение будет восстанавливаться автоматически. Более простой способ перерисовки экрана – установка свойства формы *AutoRedraw*=True. Однако это связано с определенными затратами памяти, необходимыми для хранения образа экрана в памяти ЭВМ.

Экранные координаты.

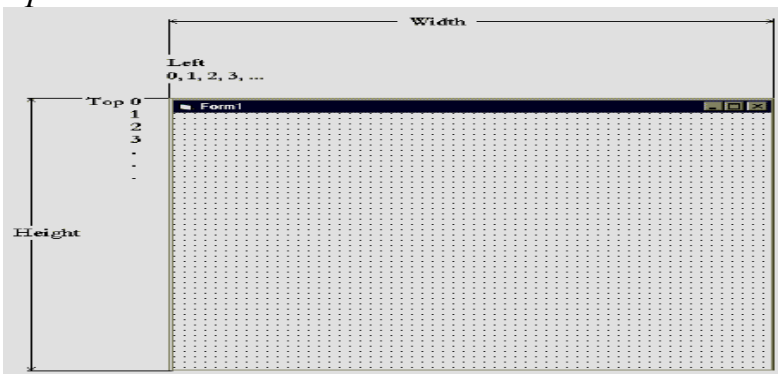


Рисунок 11.3 - Экранные координаты

Для указания размеров формы надо задать ее свойства: Width, Height, Left и Top, показанные на рисунке. Значения этих величин измеряются в twips. Если требуется работать с другими единицами измерения, то нужно изменить свойство формы ScaleMode согласно приведенной таблице:

Таблица 11.1 - Единицы измерения шкалы

ScaleMode	Единицы измерения
0	Определяются пользователем
1	twips. По умолчанию (1/20 пункта или 1/1440 дюйма или ~0.002 см)
2	Пункты (1/72 дюйма или 0,035 см.)
3	Пиксели
4	Символы (1/6 дюйма по вертикали и 1/12 дюйма по горизонтали)
5	Дюймы
6	Миллиметры
7	Сантиметры

Если ScaleMode=0, то необходимо установить свойства ScaleHeight, ScaleWidth, ScaleLeft и ScaleTop, которые аналогичны описанным выше свойствам формы. Масштаб формы можно изменять программно: либо устанавливая значения нужных свойств, либо методом Scale, который имеет следующий синтаксис:

[объект.] Scale [(left,top) – (right,bottom)],

где в качестве параметров метода указаны координаты левого верхнего и правого нижнего углов формы.

Для того, чтобы установить Декартову систему координат с центром (точка (0,0)) в центральной точке формы с одинаковой шириной и длиной окна (ScaleWidth=2a,ScaleHeight=2a), необходимо записать метод Scale в следующем виде

Scale (-a,a) – (a,-a).

Вызов метода Scale без параметров восстанавливает исходный масштаб окна формы (в twips) и возвращает начало координат в левый верхний угол.

Программирование на Visual Basic.

Типы данных.

В языке *Visual Basic* введены семь основных типов данных, которые представлены в таблице.

Таблица 11.2 - Типы данных

Тип	Описание	Занимаемая память (байт)
Integer	Целый тип	2
Long	Длинный целый тип	4
String	Текстовый тип	1 байт на символ
Currency	Числовой тип с фиксированной десятичной точкой	8
Single	Вещественный тип одинарной точности	4
Double	Вещественный тип двойной точности	8
Variant	Произвольный тип	Зависит от конкретного значения

Тип Currency (валюта) предназначен специально для финансовых расчетов. Число такого типа всегда имеет точность до четвертого знака после точки. Например, число 21,00003 округляется до 21,0000, то есть это тип данных с фиксированной точкой.

Переменная типа Variant (варьируемый) способна принимать любое значение. Данные этого типа в действительности всегда состоят из двух частей: собственно значения (любого типа) и кода, указывающего на истинный тип данных.

Объявление переменных и констант.

Синтаксис объявления переменной следующий.

Dim <переменная> [As тип]

Если опущено ключевое слово As, то Visual Basicприсвоит переменной тип Variant. По умолчанию числовой переменной присваивается нулевое значение.

Оператор Dim, расположенный в процедуре, создает только локальные переменные. Глобальные и действующие на уровне модуля переменные

должны быть явно описаны операторами Dim и Global в разделе общих объявлений (general). Синтаксис оператора Global:

Global <переменная> As тип.

Для хранения постоянных величин используются константы:

Const имя=выражение.

Например, Cjnst One=1.

Операторы управления

Условный оператор.

If выражение Then
[оператор]…
[Else
[оператор]…]
End If

Управление циклом.

А. С проверкой в начале цикла.

Do {While | Until} выражение [оператор]… Loop

Б. С проверкой в конце цикла

Do
[оператор]…
Loop {While | Until} выражение

Оператор For.

For переменная=начальное значение To конечное значение
[Step приращение]
[оператор]…
Next

Графические методы и функции.

В этом разделе будет описан синтаксис необходимых для выполнения лабораторной работы методов и функций программирования графики.

Cls – стирает все изображение и текст в указанном объекте

[объект.] Cls

Line – рисование линии в указанном объекте

[объект.] Line [[Step](x1,y1)] - [Step](x2,y2) [,цвет] [,B[F]]

где (x1,y1)-начальная точка линии. Если она не указана, линия начинается от текущей позиции. Вторую пару координат нужно указывать обязательно -(x2,y2).

Параметр <*цвет*> определяет цвет линии. Если он не задан, то для рисования используется значение свойства формы ForeColor. Для явного задания значения цвета необходимо использовать функцию QBColor(i), где i-константа, определяющая значение цвета. Значения констант можно просмотреть, используя справочную систему Visual Basic (Help).

Если указана буква B (Box-прямоугольник), то вместо линии вычерчивается прямоугольник; пары координат определяют положение его противоположных по диагонали углов. Прямоугольник заполняется цветом,

указанным в FillColor, а тип заполнения определяется свойством FillStyle формы. Если есть буква F, то прямоугольник закрашивается тем же цветом, что и границы. Параметр F используется только вместе с параметром B.

End – завершает работу приложения.

Для нахождения других стандартных функций Visual Basic рекомендуется воспользоваться справочной системой Help.

Функция в Visual Basic имеет следующий синтаксис:

Function <имя> [([параметр[As <тип1>]][,…])][As <тип2>]<операторы>
<имя>=выражение
<операторы>
End Function

Строка <имя>=выражение нужна для того, чтобы функция возвращала значение. <Тип2> указывает на тип возвращаемой величины. В круглых скобках описаны формальные параметры данной функции, фактические значения которых задаются при ее вызове.

Программирование расчета значений координат может быть реализовано с помощью отдельно написанной функции, вызов которой осуществляется в цикле. Программный код функции размещается в разделе общих объявлений (general) списка Object в окне кода.

Для установки курсора в позицию X, Y формы требуется установить значения свойств CurrentX=X, CurrentY=Y. Эти свойства доступны только в период выполнения программы. Последующий вывод информации методом Print будет осуществляться, начиная с точки с координатами (CurrentX, CurrentY).

Окно кода.

Когда создается внешний вид формы и на нее вносятся инструменты, Visual Basic автоматически подготавливает шаблоны процедур для обработки событий. Чтобы открыть окно кода, необходимо дважды щелкнуть на выбранном объекте.

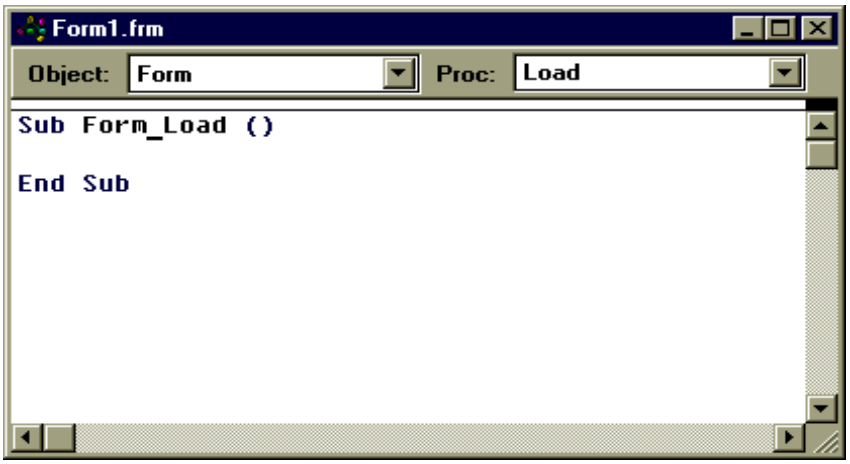


Рисунок 11.4 - Окно кода

В списке Object можно выбрать интересующий объект, а в списке Proc событие, реакцию на которое нужно оформить в виде программы. Сам код помещается между двумя ключевыми словами Sub, End Sub.

Введенный код будет храниться в том же файле, что и экранные объекты формы.

Лекция 12. Массивы

Содержание лекции: дать сведения о массивах.
Цель лекции: привить навыки работы с массивами.

Массив — это пронумерованная последовательность величин одинакового типа, обозначаемая одним именем. Элементы массива располагаются в последовательных ячейках памяти, обозначаются именем массива и индексом. Каждое из значений, составляющих массив, называется его *компонентой* (или *элементом* массива).

Массив данных в программе рассматривается как переменная структурированного типа. Массиву присваивается имя, посредством которого можно ссылаться как на массив данных в целом, так и на любую из его компонент.

Вообще, массив – однородный, упорядоченный структурированный тип данных с прямым доступом к элементам.

Переменные, представляющие компоненты массивов, называются переменными с индексами в отличие от простых переменных, представляющих в программе элементарные данные. Индекс в обозначении компонент массивов может быть константой, переменной или выражением порядкового типа (целочисленный, логический, символьный, перечислимый, диапазон).

Если за каждым элементом массива закреплен только один его порядковый номер, то такой массив называется *линейным*. Вообще количество индексов элементов массива определяет *размерность* массива. По этом признаку массивы делятся на одномерные (линейные), двумерные, трёхмерные и т.д.

Пример: числовая последовательность четных натуральных чисел 2, 4, 6, ..., *N* представляет собой линейный массив, элементы которого можно обозначить A[1]=2, A[2]=4, A[3]=6, ..., A[K]=2·(K+1), где *K*— номер элемента, а 2, 4, 6, ..., *N* — значения.

Индекс (порядковый номер элемента) записывается в квадратных скобках после имени массива.

Например, A[7] — седьмой элемент массива A; D[6] — шестой элемент массива D.

Для размещения массива в памяти ЭВМ отводится поле памяти, размер которого определяется типом, длиной и количеством компонент массива. В языке Pascal эта информация задается в разделе описаний. Массив описывается так:

```
имя массива : Array [тип индекса] Of базовый тип;
```

Чаще всего типом индекса является диапазон. Например,

```
Var B : Array [1..5] Of Real, R : Array [1..34] Of Char;
```

— описывается массив *B*, состоящий из 5 элементов и символьный массив *R*, состоящий из 34 элементов. Для массива *B* будет выделено 5·6=30 байт памяти, для массива *R* — 1·34=34 байта памяти.

Базовый тип элементов массива может быть любым: простым или структурированным, за исключением файлового.

Кроме того, массив можно объявить с использованием собственного типа:

```
Type mas = array[1..100] of integer;
     mas_ = array[1..50] of real;
Var a, b: mas;
    R: mas_;
```

Для объявления массива используется следующая запись.

{Dim | Global} имя(maxSize) [As тип].

Размерность массива (maxSize) является величиной целого типа. Нумерация элементов массива от 0 до maxSize.

Заполнить массив можно следующим образом:

1) С помощью оператора присваивания. Этот способ заполнения элементов массива особенно удобен, когда между элементами существует какая-либо зависимость, например, арифметическая или геометрическая прогрессии, или элементы связаны между собой рекуррентным соотношением.

Задача 1. Заполнить одномерный массив элементами, отвечающими следующему соотношению:

$a_1=1$; $a_2=1$; $a_i=a_{i-2}+a_{i-1}$ (i = 3, 4, ..., n).

```
Read(N); {Ввод количества элементов}
A[1]:= 1;
A[2]:= 1;
FOR I := 3 TO N DO
    A[I] := A[I - 1] + A[I - 2];
```

Другой вариант присваивания значений элементам массива — заполнение значениями, полученными с помощью датчика случайных чисел.

Задача 2. Заполнить одномерный массив с помощью датчика случайных чисел таким образом, чтобы все его элементы были различны.

```
Program Create;
Type Mas = Array[1..100] Of Integer;
```

```
Var A : Mas; I, J, N : Byte; Log : Boolean;
Begin
   Write(''); ReadLn(N);
   randomize; A[1] := -32768 + random(65535);
   For I := 2 To N Do
   Begin
      Log := True;
      Repeat
         A[i] := -32768 + random(65535); J := 1;
         While Log and (j <= i - 1) Do
         begin Log := a[i] <> a[j]; j := j + 1 End
      Until Log
   End;
   For i := 1 to N Do Write(a[i]:7); writeln
End.
```

2) Ввод значений элементов массива с клавиатуры используется обычно тогда, когда между элементами не наблюдается никакой зависимости. Например, последовательность чисел 1, 2, -5, 6, -111, 0 может быть введена в память следующим образом:

```
Program Vvod;
 Var N, I : Integer;
    A : Array [1..20] Of Integer;
Begin
 Write('Введите количество элементов массива ');  ReadLn(N);
 FOR I := 1 TO N DO
 Begin
  Write('Введите A[', I, '] '); ReadLn(A[I])
End.
```

Над элементами массивами чаще всего выполняются такие действия, как

а) поиск значений;

б) сортировка элементов в порядке возрастания или убывания;

в) подсчет элементов в массиве, удовлетворяющих заданному условию.

Сумму элементов массива можно подсчитать по формуле $S=S+A[I]$ первоначально задав $S=0$. Количество элементов массива можно подсчитать по формуле $K=K+1$, первоначально задав $K=0$. Произведение элементов массива можно подсчитать по формуле $P = P \cdot A[I]$, первоначально задав $P = 1$.

Задача 3. Дан линейный массив целых чисел. Подсчитать, сколько в нем различных чисел.

```
{Подсчет количества различных чисел в линейном массиве.
Идея решения: используем вспомогательный массив, элементами
которого являются логические величины (False - если элемент
```

```
уже встречался ранее, True - иначе)}
Program Razlichnye_Elementy;
Var I, N, K, Kol : Integer;
A : Array [1..50] Of Integer;
Lo : Array [1..50] Of Boolean;
Begin
Write('Введите количество элементов массива: '); ReadLn(N);
FOR I := 1 TO N DO
Begin
Write('A[', I, ']='); ReadLn (A[I]);
Lo[I] := True; {Заполняем вспомогательный массив значениями True}
End;
Kol := 0; {переменная, в которой будет храниться количество различных чисел}
FOR I := 1 TO N DO
IF Lo[I] THEN
Begin
Kol := Kol + 1;
FOR K := I TO N DO
{Во вспомогательный массив заносим значение False,
если число уже встречалось ранее или совпадает с текущим элементом A[I]}
Lo[K] := (A[K] <> A[I]) And Lo[K];
End;
WriteLn('Количество различных чисел: ', Kol)
END.
```

Задача 4. Дан линейный массив. Упорядочить его элементы в порядке возрастания.

```
{Сортировка массива выбором (в порядке возрастания).
Идея решения: пусть часть массива (по K-й элемент включительно) отсортирована. Нужно найти в неотсортированной части массива минимальный элемент и поменять местами с (K+1)-м}
Program Sortirovka;
Var N, I, J, K, Pr : Integer; A : Array [1..30] Of Integer;
Begin
Write('Введите количество элементов: '); ReadLn(N);
For I := 1 To N Do
Begin
Write('Введите A[', I, '] '); Readln(A[I]);
End;
WriteLn;
For I := 1 To N - 1 Do
Begin
```

```
K := I;
For J := I + 1 To N Do If A[J] <= A[K] Then K := J;
Pr := A[I]; A[I] := A[K]; A[K] := Pr;
End;
For I := 1 To N Do Write(A[I], ' ');
End.
```

Если два массива являются массивами эквивалентных типов, то возможно присваивание одного массива другому. При этом все компоненты присваиваемого массива копируются в тот массив, которому присваивается значение. Типы массивов будут эквивалентными, если эти массивы описываются совместно или описываются идентификатором одного и того же типа. Например, в описании

```
Type Massiv = Array[1..10] Of Real;
Var A, B : Massiv; C, D : Array[1..10] Of Real; E : Array[1..10] Of Real;
```

типы переменных A, B эквивалентны, и поэтому данные переменные совместимы по присваиванию; тип переменных C, D также один и тот же, и поэтому данные переменные также совместны по присваиванию. Но тип переменных C, D не эквивалентен типам переменных A, B, E, поэтому, например, A и D не совместны по присваиванию. Эти особенности необходимо учитывать при работе с массивами.

При работе с массивами целесообразно использовать процедуры и функции. Вот типовые процедуры:

```
Type mas = array[1..100] of LongInt;
{Заполнение массива с помощью ввода с клавиатуры}
Procedure Vvod_Kl(var n: byte; var a: mas);
Var i: byte;
Begin
Write('Количество элементов?'); Readln(n);
For i := 1 to n do
Begin
write(i, '-й элемент'); readln(a[i])
End
End;
{Заполнение массива случайными данными}
Procedure Vvod_Sl(var n: byte; var a: mas);
Var i: byte;
Begin
Write('Количество элементов?'); Readln(n);
For i := 1 to n do
a[i] := -1000+random(2001)
End;
{Вывод массива}
Procedure Print(n: byte; const a: mas);
```

```
Var i: byte;
Begin
For i := 1 to n do
write(a[i] :8);
writeln
End;
```

При решении практических задач часто приходится иметь дело с различными таблицами данных, математическим эквивалентом которых служат матрицы. Такой способ организации данных, при котором каждый элемент определяется номером строки и номером столбца, на пересечении которых он расположен, называется *двумерным массивом* или *таблицей*.

Например, данные о планетах Солнечной системы представлены следующей таблицей.

Планета	Расст. до Солнца	Относ. обьем	Относ. масса
Меркурий	57.9	0.06	0.05
Венера	108.2	0.92	0.81
Земля	149.6	1.00	1.00
Марс	227.9	0.15	0.11
Юпитер	978.3	1345.00	318.40
Сатурн	1429.3	767.00	95.20

Их можно занести в память компьютера, используя понятие двумерного массива. Положение элемента в массиве определяется двумя индексами. Они показывают номер строки и номер столбца. Индексы разделяются запятой. Например: A[7, 6], D[56, 47].

Заполняется двумерный массив аналогично одномерному: с клавиатуры, с помощью оператора присваивания. Например, в результате выполнения программы:

```
Program Vvod2;
Var I, J : Integer;
A : Array [1..20, 1..20] Of Integer;
Begin
FOR I := 1 TO 3 DO
FOR J := 1 TO 2 DO A[I, J] := 456 + I
End.
```

0 элементы многомерных массивов, важно себе представлять. В большинстве алгоритмических языков реализуется общее правило, устанавливающее порядок хранения в памяти элементов массивов: элементы многомерных массивов хранятся в памяти в последовательности, соответствующей более частому изменению младших индексов.

Задача 5. Заполнить матрицу порядка *n* по следующему образцу.

```
1    2   3  ... n-2 n-1  n
2    1   2  ... n-3 n-2 n-1
3    2   1  ... n-4 n-3 n-2
...  ...  ... ... ... ... ...
n-1  n-2  n-3 ...  2   1   2
n    n-1  n-2 ...  3   2   1
Program Massiv12;
Var I, J, K, N : Integer;
A : Array [1..10, 1..10] Of Integer;
Begin
Write('Введите порядок матрицы: '); ReadLn(N);
For I := 1 To N Do
For J := I To N Do
Begin
A[I, J] := J - I + 1; A[J, I] := A[I, J];
End;
For I := 1 To N Do
Begin
WriteLn;
For J := 1 To N Do Write(A[I, J]:4);
End
End.
```

Задача 6. Дана целочисленная квадратная матрица. Найти в каждой строке наибольший элемент и поменять его местами с элементом главной диагонали.

```
Program Obmen;
Var N, I, J, Max,Ind, Vsp : Integer;A : Array [1..15, 1..15] Of Integer;
Begin
WRITE('Введите количество элементов в массиве: '); READLN(N);
FOR I := 1 TO N DO
FOR J := 1 TO N DO
Begin
WRITE('A[', I, ',', J, '] '); READLN(A[I, J])
End;
FOR I := 1 TO N DO
Begin
Max := A[I, 1]; Ind := 1;
```

```
FOR J := 2 TO N DO
IF A[I, J] > Max THEN
Begin
Max := A[I, J]; Ind := J
End;
Vsp := A[I, I]; A[I, I] := A[I, Ind]; A[I, Ind] := Vsp
End;
FOR I := 1 TO N DO
Begin
WriteLn;
FOR J := 1 TO N Do Write(A[I, J] : 3);
End; WriteLn
End.
```

Задача 7. Задана прямоугольная целочисленная таблица размером $m \times n$. Указать столбец (назвать его номер), где минимальное количество элементов, кратных сумме индексов.

```
type mas2=array[1..20, 1..20] of integer;

{Формирование массива заданного размера с помощью датчика псевдослучайных чисел}
procedure create(var m, n: byte; var a: mas2);
var i, j: byte;
begin
write('Введите размеры таблицы: '); readln(m, n);
for i:=1 to m do
for j:= 1 to n do
a[i, j]:= -20+random(41);
end;

{вывод таблицы на экран}
procedure print(const m, n: byte; const a: mas2);
var i, j: byte;
begin
for i:=1 to m do
begin
writeln;
for j:=1 to n do
write(a[i, j]:5);
end;
writeln
end;
```

```
{Поиск нужного столбца; результат - в переменной k и, далее, - значение функции}
function solution(const m, n: byte; const a: mas2): byte;
var i, j, k, ke, mke: byte;
begin
k:=1; mke:=m;
for j:=1 to n do
begin
ke:=0;
for i:=1 to m do
if abs(a[i, j]) mod (i+j) = 0 then ke:=ke+1;
if ke<mke then begin mke:=ke; k:=j end
end;
solution := k;
end;

var m, n: byte; a: mas2;
begin
randomize;
create(m, n, a);
print(m, n, a);
writeln('Ответ: ', solution(m, n, a))
end.
```

Лекция 13. Процедуры

Содержание лекции: назначение подпрограмм. Механизм подпрограмм, их описание и вызов. Назначение параметров. Формальные и фактические параметры.

Цель лекции: научить работать с подпрограммами, параметрами.

Подпрограмма (процедура) (англ. *subroutine*) — поименованная или иным образом идентифицированная часть компьютерной программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно *вызвана* из разных частей программы. В языках программирования для оформления и использования подпрограмм существуют специальные синтаксические средства.

Назначение подпрограмм.

Подпрограммы изначально появились как средство оптимизации программ по объёму занимаемой памяти — они позволили не повторять в программе идентичные блоки кода, а описывать их однократно и вызывать по мере необходимости. К настоящему времени данная функция подпрограмм

стала вспомогательной, главное их назначение — структуризация программы с целью удобства её понимания и сопровождения.

- выделение набора действий в подпрограмму и вызов её по мере необходимости позволяет логически выделить целостную подзадачу, имеющую типовое решение. Такое действие имеет ещё одно (помимо экономии памяти) преимущество перед повторением однотипных действий: любое изменение (исправление ошибки, оптимизация, расширение функциональности), сделанное в подпрограмме, автоматически отражается на всех её вызовах, в то время как при дублировании каждое изменение необходимо вносить в каждое вхождение изменяемого кода.

Даже в тех случаях, когда в подпрограмму выделяется однократно производимый набор действий, это оправдано, так как позволяет сократить размеры целостных блоков кода, составляющих программу, то есть сделать программу более понятной и обозримой...

Механизм подпрограмм, их описание и вызов.

В простейшем случае (в ассемблерах) подпрограмма представляет собой последовательность команд (операторов), отдельную от основной части программы и имеющую в конце специальную команду выхода из подпрограммы. Обычно подпрограмма также имеет имя, по которому её можно вызвать, хотя ряд языков программирования допускает использование и неименованных подпрограмм. В языках высокого уровня описание подпрограммы обычно состоит по меньшей мере из двух частей: *заголовка* и *тела*. Заголовок подпрограммы описывает её имя и, возможно, параметры, то есть содержит информацию, необходимую для вызова подпрограммы. Тело — набор операторов, который будет выполнен всякий раз, когда подпрограмма будет вызвана.

Вызов подпрограммы выполняется с помощью команды вызова, включающей в себя имя подпрограммы. В большинстве современных языков программирования команда вызова представляет собой просто имя вызываемой подпрограммы, за которым могут следовать фактические параметры

В следующем примере на языке Паскаль подпрограмма subprog вызывается из основной программы трижды:

```
program SubProgExample;
  // Описание подпрограммы subprog
  procedure subprog; // Заголовок, включающий имя подпрограммы
  begin // начало тела подпрограммы
    WriteLn('Bye');
  end; // конец тела подпрограммы
begin
  WriteLn('Hello');
  subprog; // 1-й вызов
```

```
  subprog; // 2-й вызов
  subprog; // 3-й вызов
end.
```

Результатом выполнения такой программы станет вывод строки «Hello» и трёх строк «Bye».

Для сохранения и восстановления контекста выполнения вызывающей процедуры, с целью исключения побочных эффектов, связанных с возможным нежелательным изменением используемых машинных регистров, компилятор формирует для каждой процедуры специальные последовательности команд, называемые прологом и эпилогом процедуры.

Некоторые языки программирования (Паскаль, Ада, Модула-2) допускают описание вложенных подпрограмм, то есть помещение подпрограмм внутрь других подпрограмм. Такие вложенные подпрограммы могут использоваться только в той подпрограмме, в которой они описаны. В иных случаях (например, в языке Си) вложение подпрограмм не допускается. Никаких принципиальных преимуществ вложение подпрограмм не даёт, но может быть удобно для более логичной структуризации программы (если какая-то подпрограмма используется только в некоторой другой подпрограмме, логично поместить первую во вторую).

Параметры подпрограмм

Назначение параметров.

Подпрограммы часто используются для многократного выполнения стереотипных действий над различными данными. Подпрограмма обычно имеет доступ к объектам данных, описанным в основной программе (по крайней мере, к некоторым из них), поэтому для того, чтобы передать в подпрограмму обрабатываемые данные, их достаточно присвоить, например, глобальным переменным. Но такой путь не особенно удобен и чреват ошибками.

Для обеспечения контролируемой передачи параметров в подпрограмму и возврата результатов из неё используется механизм *параметров*. Параметры описываются при описании подпрограммы (в её заголовке) и могут использоваться внутри процедуры аналогично переменным, описанным в ней. При вызове процедуры значения каждого из параметров указываются в команде вызова (обычно после имени вызываемой подпрограммы).

```
program SubProgExample2;
  // Описание подпрограммы subprog
  procedure subprog(Line: String); // Заголовок, включающий имя подпрограммы
  begin // начало тела подпрограммы
    WriteLn(Line);
  end; // конец тела подпрограммы
begin
```

```
  WriteLn('Hello');
  subprog('Good bye,'); // 1-й вызов
  subprog('my love,');  // 2-й вызов
  subprog('good bye!'); // 3-й вызов
end.
```

В приведённом примере параметр *Line* подпрограммы *subprog* в каждом вызове получает различное значение, благодаря чему выводятся не одинаковые строки, а разные.

Формальные и фактические параметры.

Чтобы отличать параметры подпрограммы, описанные в её заголовке и теле, от параметров, указываемых при вызове подпрограммы, используются формальные и фактические параметры. Так, в последнем примере параметр *Line* в заголовке и теле подпрограммы *subprog* — это формальный параметр, а строка *'Good bye'* , использованная в первом вызове этой подпрограммы — фактический параметр. *При вызове подпрограммы фактические параметры, указанные в команде вызова, становятся значениями соответствующих формальных параметров, чем и обеспечивается передача данных в подпрограмму.*

Способ передачи параметров в подпрограмму.

Существует несколько способов передачи параметров в подпрограмму:

1) Передача параметров по значению. Формальному параметру присваивается *значение* фактического параметра. В этом случае формальный параметр будет содержать копию значения, имеющегося в фактическом, и никакое воздействие, производимое внутри подпрограммы на формальные параметры, не отражается на параметрах фактических. Так, если в качестве фактического параметра будет использована переменная, и внутри подпрограммы значение соответствующего формального параметра будет изменено, то фактический параметр останется без изменений.

```
int func1(int x)
{ x=x+1; return x;
}
```

2) Передача параметров по ссылке. В формальный параметр может быть помещён *сам* фактический параметр (обычно это реализуется путём помещения в формальный параметр ссылки на фактический). При этом любое изменение формального параметра в подпрограмме отразится на фактическом параметре — оба параметра во время вызова подпрограммы- суть одно и то же. Параметры, передаваемые по ссылке, дают возможность не только передавать параметры внутрь подпрограммы, но и возвращать вычисленные значения в точку вызова. Для этого параметру внутри подпрограммы просто присваивается нужное значение, и после возврата из подпрограммы

переменная, использованная в качестве фактического параметра, получает это значение.

```
void func2(int &x)
{ x=x+1;
}
```

3) Передача параметров по имени. В формальный параметр может быть помещено произвольное выражение. При этом вычисление этого выражения произойдёт внутри подпрограммы в тот момент, когда потребуется его значение. Если это значение фигурирует несколько раз, то и вычисляться оно будет тоже несколько раз. Параметры, передаваемые по имени, дают возможность писать довольно универсальные подпрограммы. Такой способ передачи параметров используется, к примеру, в языках Алголили Алгол 68.

4) Передача параметров через стек. Это фактически разновидность передачи параметра по значению «с ручным приводом», в данном случае отсутствует понятие формальных и фактических параметров. Все параметры лежат на стеке, причём их типы, количество и порядок не контролируются компилятором. Данный подход реализован в языке Форт.

Язык программирования может предоставлять возможность передавать параметры в подпрограммы либо только по значению, либо по значению и по ссылке, либо по имени и значению. В последних двух случаях для различения способов передачи параметра используются отдельные синтаксические конструкции (в Паскале это ключевое слово *var*при описании параметра). В действительности, если язык содержит понятие ссылки (указателя), то можно обойтись и без передачи параметра по ссылке (её всегда можно смоделировать, описав параметр типа «ссылка»), но эта возможность удобна, так как позволяет работать с формальным параметром-ссылкой без разыменования, а также повышает надёжность и безопасность программы.

На параметры, передаваемые по ссылке, накладываются естественные ограничения: фактический параметр, подставляемый на место такого параметра при вызове, обязан быть переменной (то есть иметь адрес), а в языках со строгой типизацией — ещё и иметь в точности такой же тип данных.

Виды подпрограмм.

В языках программирования высокого уровня используется два типа подпрограмм: *процедуры и функции*.

1) Функция — это подпрограмма специального вида, которая, кроме получения параметров, выполнения действий и передачи результатов работы через параметры имеет ещё одну особенность — она всегда должна возвращать результат. Вызов функции является, с точки зрения языка программирования, выражением, он может использоваться в других выражениях или в качестве правой части присваивания.

2) Процедура — это независимая именованная часть программы, которую после однократного описания можно многократно вызвать по имени из последующих частей программы для выполнения определенных действий.

Подпрограммы, входящие в состав классов в объектных языках программирования, обычно называются методами. Этим термином называют любые подпрограммы-члены класса, как функции, так и процедуры; когда требуется уточнение, говорят о *методах-процедурах* или *методах-функциях*.

Лекция 14. Рекурсия

Содержание лекции: понятие рекурсии. Работа с рекурсией.
Цель лекции: научить работать с рекурсией.

Рекурсия (от латинского слова recursio- возвращение) означает обращение к самому себе.

В частности, рекурсия применяется для решения задачи сортировки одномерного массива-выстраивания его элементов в порядке возрастания (убывания).

Существует много алгоритмов сортировки:

- *«наивная» сортировка*
- *пузырьковая сортировка*
- *сортировка выбором*
- *сортировка включением.*

Задача сравнения нескольких (трех) чисел с целью их сортировки-расстановки в порядке возрастания.

Пусть у нас 3 яблока и рычажные весы без гирь, с помощью которых можно сравнивать вес двух яблок.

Составим алгоритм и программу, которая моделирует процесс попарного сравнения яблок:

Пусть a,b,c-переменные, значения которых - вес каждого яблока. И пусть Max, Mid, Min - это переменные, значениями которых должен стать вес самого большого, среднего и самого маленького яблока.

Рассмотрим перестановки трех чисел:

(a,b,c) (a,c,b) (b,a,c) (b,c,a) (c,a,b) (c,b,a)

По очереди будем брать каждую из них и проверять, не выполняются ли для нее два неравенства:

N1>=N2>=N3.

Как только эти два неравенства будут выполнены, алгоритм завершит свою работу и мы получим результат

Max=N1, Mid=N2, Min=N3.

Этот алгоритм записывается с помощью следующего кода:

```
Dim a As Single, b As Single, c As Single
Dim Max As Single, Mid As Single, Min As Single
'Декларативный стиль программирования.
```

```
'Проверка шести условий в шести вложенных
'друг в друга условных операторах.
If a >= b And b >= c Then
   Max = a: Mid = b: Min = c
Else
  If a >= c And c >= b Then
   Max = a: Mid = c: Min = b
  Else
    If b >= a And a >= c Then
     Max = b: Mid = a: Min = c
    Else
      If b >= c And c >= a Then
       Max = b: Mid = c: Min = a
      Else
        If c >= a And a >= b Then
         Max = c: Mid = a: Min = b
        Else
          If c >= b And b >= a Then
           Max = c: Mid = b: Min = a
          End If
        End If
      End If
    End If
  End If
End If
```

Оценим число сравнений, которые необходимо выполнить для того, чтобы найти максимальное и минимальное значение из трех. В лучшем случае нужно проверить 2 неравенства, а в худшем - 10 неравенств!

Следует поступить так: сначала сравнить 2 яблока. Более тяжелое (первое) отложить, а более легкое (второе) - сравнить с третьим. Если третье окажется легче второго, то третье - самое легкое, а первое - самое тяжелое. Если второе легче третьего, то самое легкое - второе. А для нахождения самого тяжелого яблока сравнить первое яблоко с третьим. В лучшем случае потребуется 2 сравнения, в худшем - всего 3!

Следующий код демонстрирует данный алгоритм:

```
'Процедурный стиль программирования.
If a >= b Then
  If b >= c Then
    Max = a: Mid = b: Min = c
  Else
    If a >= c Then
     Max = a: Mid = c: Min = b
    Else
     Max = c: Mid = a: Min = b
```

```
    End If
  End If
End If
  If a >= c Then
    Max = b: Mid = a: Min = c
  Else
    If b >= c Then
      Max = b: Mid = c: Min = a
    Else
      Max = c: Mid = b: Min = a
    End If
  End If
```

Рекурсивная функция и рекурсивная процедура.

Объект называется рекурсивным объектом, если он целиком или частично определяется через самого себя.

Объекты, которые определяются через себя *целиком,* практического интереса не представляют, так как являются бесконечными.

Но объекты, которые определяются через себя *частично* используются на практике. Примером является определение факториала натурального числа в математике.

Факториалом натурального числа N называется такая функция, которая возвращает произведение всех натуральных чисел от 1 до данного числа. Условно считается, что факториалом 0 является 1.

Обозначим эту функцию Факториал N.

Тогда, Факториал (0)=1; Факториал (3)=6; Факториал (6)=720.

Факториал числа можно определить *через сам факториал*: если N=0, то Факториал (N) равен 1, если N>0, то

Факториал (N) равен числу N умноженному на Факториал (N-1).

Рекурсивная функция (процедура) - это такая функция (процедура), среди *Выполняемых Операторов* которой есть оператор вызова самой этой функции (процедуры).

Если бы при работе рекурсивной функции рекурсивные вызовы выполнялись бы безо всяких условий, ее работа никогда бы не закончилась. Поэтому среди *Выполняемых Операторов* должно быть условие завершения (продолжения) рекурсии.

Последовательность рекурсивных вызовов называется *рекурсивным спуском.*

Номер рекурсивного вызова в этой последовательности называется *глубиной рекурсии*.

Среди *Выполняемых Операторов* в определении функции могут быть такие, которые выполняют некоторое *действие* и которые расположены до оператора рекурсивного вызова. Они выполняются в том порядке, в котором

происходят рекурсивные вызовы. Говорят, что действие выполняется на рекурсивном спуске.

Среди *Выполняемых Операторов* в определении функции могут быть и такие, которые тоже выполняют некоторое *действие,* и которые расположены после оператора рекурсивного вызова. Они выполняются в том порядке, обратном тому, в котором происходят рекурсивные вызовы. Говорят, что действие выполняется на рекурсивном подъеме.

Задача. Определить рекурсивные функции *Факториал1* и *Факториал2*. В первой действие выполняется на *рекурсивном спуске*, во второй - *на рекурсивном подъеме.*

```
Public Function Факториал1(Число1 As Integer, _
Индекс As Integer, Произведение As Long) As Long
  'Действие:
Произведение = Произведение * Индекс
  'Условие:
If Число = 0 Or Индекс = Число Then
  Факториал1 = Произведение
Else
  Факториал1 = Факториал1(Число, Индекс + 1, Произведение)
End If
End Function

Public Function Факториал2(Число As Integer) As Long
Dim Произведение As Long
  'Условие:
If Число = 0 Or Число = 1 Then
   Число = 1: Произведение = 1
Else
   Произведение = Факториал2(Число - 1)
End If
 'Действие:
Факториал2 = Произведение * Число
End Function
```

Сортировка одномерного массива.

Сортировкой одномерного массива называется такая *процедура*, в результате которой его элементы меняют свои порядковые номера так, что каждый последующий в списке элемент не меньше, чем предыдущий.

Например, первоначальный порядок элементов массива был таким:
M(0)=5, M(1)=15, M(2)=12, M(3)=2, M(4)=22, M(5)=5.
После сортировки:
M(0)=2, M(1)=5, M(2)=5, M(3)=12, M(4)=15, M(5)=22.

Сортировка выбором.

Идея алгоритма.

Ищется номер максимального элемента массива. На его место помещается последний элемент массива. А на место последнего элемента помещается максимальный элемент (элементы меняются местами). Затем берется часть массива без последнего элемента. С этим остатком проделывается тоже самое, что и с исходным массивом. И так до тех пор, пока длина последнего остатка не станет равной 1.

```
Public Sub СортировкаВыбором(Массив() As Double, _
n As Long)
Dim A As Double, i As Long, Индекс As Long
  For i = n To 1 Step -1
  Индекс = ИндексМаксЭлемента(Массив(), i)
  A = Массив(Индекс)
  Массив(Индекс) = Массив(i)
  Массив(i) = A
  Next i
End Sub
```

Сортировка вставкой.

Идея алгоритма.

Отделим от массива последний элемент. С помощью этого же алгоритма (рекурсивно) отсортируем укороченный массив (без последнего элемента). В отсортированный массив вставим отделенный вначале последний элемент. Условием окончания рекурсии является то, что последний укороченный массив будет содержать всего один элемент.

Пузырьковая сортировка.

Идея алгоритма.

В массиве последовательно просматриваются элементы от первого до предпоследнего. И для каждого i-го элемента массив просматривается от (i+1)-го элемента до конца.

Если данный i-й элемент окажется больше одного из тех элементов, которые находятся после него, они меняются местами: i-й элемент, словно пузырек воздуха всплывает вверх-по направлению к верхнему краю массива.

```
For  i = 0 To n - 1
 For j = i + 1 To n
  If M(i) > M(j) Then
     A = M(i): M(i) = M(j): M(j) = A
  End If
 Next j
Next i
```

«Наивная» сортировка.

Идея алгоритма.

Создаются всевозможные перестановки элементов массива. И для каждой из них делается проверка-не выполняется ли для нее условие неубывания.

Лекция 15. Файлы

Содержание лекции: виды файлов. Работа с файлами. Изобретение и закрытие файлов.

Цель лекции: обучить трудиться с файлами при составлении программ.

Файлы с последовательным доступом.

Текстовые файлы называют также *файлами с последовательным доступом* или просто *последовательными файлами.*

Операторы языка позволяют:

- *открыть* файл для доступа к его содержимому или для записи в него новых данных, или для чтения записанной в нем информации;
- *внести* в открытый файл данные из программы;
- *извлечь* данные из открытого файла и поместить их в программу;
- *закрыть* файл после того, как работа над ним завершится.

Открытие файла.

Для открытия текстового файла используется оператор Open, имеющий следующий синтаксис:

Open *ИмяФайла* For РежимРаботы As # ДескрипторФайла

ИмяФайла - это либо строка символов, заключенная в кавычки, либо выражение, значением которого является строка символов. Она представляет собой путь к открываемому файлу;

РежимРаботы - это одно из трех ключевых слов:

OutPut - если файл открывается для записи в него данных, начиная с первой позиции (при записи вся имевшаяся информация будет стерта),

Append - если файл открывается для записи в него данных, начиная не с первой, а с конечной позиции, так что вся имевшаяся в нем информация будет сохранена,

Input - если файл открывается для чтения из него текстовых данных.

ДескрипторФайла - это любое целое число 1 до 511. Оно служит идентификатором файла в программе.

Закрытие файла.

Все открытые текстовые файлы закрываются одинаково - с помощью оператора Close.

Close # [*СписокДескрипторов*]

СписокДескрипторов - это записанные через запятую дескрипторы (идентификаторы) файла, которые должны быть закрыты. Если *СписокДескрипторов* отсутствует, то будут закрыты *все* открытые файлы.

Пример:

```
Open “c:\VB\Tests\Test1.txt” For Input As #1
Open “c:\VB\Tests\Test1.txt” For Input As #2
Open “c:\VB\Results\Result1.txt” For Output As #3
 ‘операторы преобразования данных, извлекаемых из файлов для чтения, в данные,  помещаемые в файл для записи
Close #1,2,3
```

Запись в файл

Записать данные в текстовый файл можно двумя способами:

- с помощью оператора Write;
- с помощью оператора Print.

Оба оператора записывают данные в файл *текстовыми строками*.

Текстовая строка - это последовательность символов, которая заканчивается знаком перехода на новую текстовую строку или знаком возврата каретки.

Текстовый файл представляет собой последовательность текстовых строк.

Синтаксис операторов записи в текстовый файл таков:

Write # *ДескрипторФайла*, [*СписокЗначений*]

Print # *ДескрипторФайла*, **[***СписокЗначений***]**

ДескрипторФайла - это целое число, совпадающее с идентификатором открытого для записи файла;

СписокЗначений - это записанные через *Разделитель* значения (или выражения, значения которых вычисляются, в частности переменные). *Разделителем* в *СпискеЗначений* является запятая. Если *СписокЗначений* отсутствует, в файл будет записана пустая строка.

Логика работы операторов Write и Print различна.

Оператор Write. Список Значений просматривается последовательно, и элементы этого списка записываются в одну текстовую строку через запятую. Элементы типа String заключаются в кавычки. После записи последнего элемента записывается символ перехода на новую строку.

Оператор Print. Элементы *Списка Значений* в этом операторе должны быть разделены либо точкой с запятой, либо запятой.

От этого зависит, как они будут записаны в текстовую строку файла:

- *если разделитель* - это точка с запятой (;), значения будут записываться подряд, без промежутков между ними;
- *если разделитель* - это запятая (,), значения будут записываться в 14-символьные зоны вывода.

Кроме того, в *Списке Значений* оператора Print могут присутствовать функции:

- Spc(n) - для вставки n пробелов между значениями в текстовой строке;
- Tab(n) - для указания номера n позиции для записи следующего значения.

Оператор Print удобен для тщательного редактирования текста выходного файла, а оператор Write лучше применять в том случае когда выходной файл будет использоваться в дальнейшем как входной для других программ.

Пример. Программа записи в текстовый файл с именем BirthdayYear1.txt следующих строк:

«Иванов», «Ваня», 1986
«Петров», «Петя», 1984
«Петрова», «Маша», 1986
«Сидоров», «Саша», 1985

```
Dim Family As String, Name As String, _  Year As String
Open "C:\Мои документы\My Documents\Examples\Year2.txt»_ For Output As #1
For i = 1 To 2
   Family = InputBox("Ввод данных", _
   "Введите фамилию")
   Name = InputBox("Ввод данных", _
   "Введите имя")
   Year = Val(InputBox("Ввод данных", _
   "Введите год рождения"))
      'Write #1, Family, Name, Year - Случай 1
    Print #1, Tab(3); Family; Spc(3); Name; Spc(3); Year ' Случай 2
Next i
Close #1
```

Чтение из файла.

Читать данные из текстового файла можно разными способами:

- с помощью операторов Input и Line Input;
- помощью функции Input.

Оператор Input имеет следующий синтаксис:

Input # *Дескриптор Файла Список Переменных*

Дескриптор Файла - это целое число, совпадающее с идентификатором открытого для чтения файла;

Список Переменных - это записанные через запятую переменные любого типа.

Оператор Line Input имеет следующий синтаксис:

Line Input #*Дескриптор Файла Переменная*

Дескриптор Файла - это целое число, совпадающее с идентификатором открытого для чтения файла;

Переменная - переменная типа String или Variant.

Результатом работы оператора Line Input является присвоение *Переменной* значения - всей очередной текстовой строки файла.

Функция Input - это значение двух аргументов. *Первый аргумент* - это количество символов, которое надо прочесть из входного файла. *Второй аргумент* - идентификатор файла, открытого для чтения. *Возвращаемое значение* - прочитанный текст в виде символьной строки.

Чаще всего эту функцию используют для одновременного чтения всего текстового файла и размещения его в текстовом поле экранной формы. Для этого необходимо выяснить размер файла в символах. Делается это с помощью функции LOF (*length of file - длина файла*, значение функции имеет тип Long) , единственным аргументом которой является идентификатор файла открытого для чтения.

Пример:

```
Open "C:\Мои документы\My Documents\Examples\Year2.txt" For Input As #1
Do Until EOF(1)
    Input #1, Family$, Name$, Year%
  List1.AddItem Family
  List2.AddItem Name
  List3.AddItem Year
Loop
Close #1
```

Файлы с произвольным доступом.

Это файлы более высокого уровня, в которых используется пользовательский тип данных.

Главное *отличие* этой структуры в том, что она содержит компоненты, которые могут иметь разные! типы.

Пользовательский тип данных принято называть записью (*record*), а компоненты этой структуры - полями записями (*fields*).

Все записи этих файлов имеют постоянную длину и порядковый номер. Это позволяет находить сразу без предварительного их просмотра нужную запись и что-то с ней сделать.

Пользовательский тип данных объявляется следующим образом:

```
Private Type Имя Типа Данных
  Элемент1 As Тип Элемента1
  …
  ЭлементN As Тип Элемента N
End Type
```

Вместо ключевого слова Private может быть записано ключевое слово Public.

Имя Типа Данных - это имя, которое присваивается определяемому типу данных,

Элемент1, ..., ЭлементN - это названия полей записи, для которой создается пользовательский тип данных,

ТипЭлемента1, ..., Тип Элемента N - это стандартные типы данных, которые будут находиться в полях записях.

После определения пользовательского типа данных необходимо объявить переменные, которые будут иметь этот тип обычным образом:

Dim *Имя Переменной* As *Имя Типа Данных*

Пример. Определить пользовательский тип данных.

```
Private Type Данные О Студенте
 Имя As String * 15
 Фамилия As String * 20
 Дата Рождения As Date
 Класс As String * 4
 Адрес As String * 32
 Телефон As String * 15
End Type
Dim Ученик As Данные О Студенте, x As Integer
x = Len(Ученик)
```

Создание файлов произвольного доступа и работа с ними.

Операторы, которые позволяют создавать файлы произвольного типа и работать с ними следующие:

- операторы *открытия и закрытия* файлов;
- операторы *записи* в файл и *чтения* из файла.

Открытие файла.

Для открытия файла прямого доступа используется оператор Open.

Open *Имя Файла* For Random As #*Дескриптор Файл*_
Len =*Длина Записи*

Имя Файла и *Дескриптор Файл* - определяются так же, как и в операторе Open для текстовых файлов. Но в отличие от текстовых файлов здесь не делается различия между файлами для записи и для чтения : все открываются в одном режиме, который определяется ключевым словом Random.

В данном случае оператор Open содержит ключевое слово Len, вслед за которым указывается *Длина Записи* - целое число.

Закрытие файла.

Для закрытия файла произвольного доступа используется оператор Close. Его синтаксис и работа не отличаются от синтаксиса и логики работы оператора Close для закрытия текстовых файлов.

Close # [*Список Дескрипторов*]

Запись в файл.

Для включения в файл произвольного доступа новой записи применяется оператор Put, который имеет следующий синтаксис.

Put # *Дескриптор Файла, Номер Записи, Имя Переменной*

Дескриптор Файла - это *идентификатор* открытого файла прямого доступа, в который *помещается* запись.

Номер Записи - это номер, который будет иметь запись, помещаемая в файл. Номером может быть только целое положительное число.

Имя Переменной - это имя переменной пользовательского типа, значением которого является запись, помещаемая в файл.

Чтение из в файла.

Для извлечения записи из файла произвольного доступа применяется оператор Get, имеющий следующий синтаксис:

Get # *Дескриптор Файла, Номер Записи, Имя Переменной*

Дескриптор Файла - это *идентификатор* открытого файла произвольного доступа, из которого *извлекается* запись.

Номер Записи - это номер *извлекаемой* из файла записи.

Имя Переменной - это имя переменной пользовательского типа, значением которого является запись, *извлекаемая* из файла.

Пример.

```
Private Sub Command1_Click() ’Внести данные в списки
Dim ListName As String
For i = 1 To 10
 For j = 0 To 5
 Select Case j
```

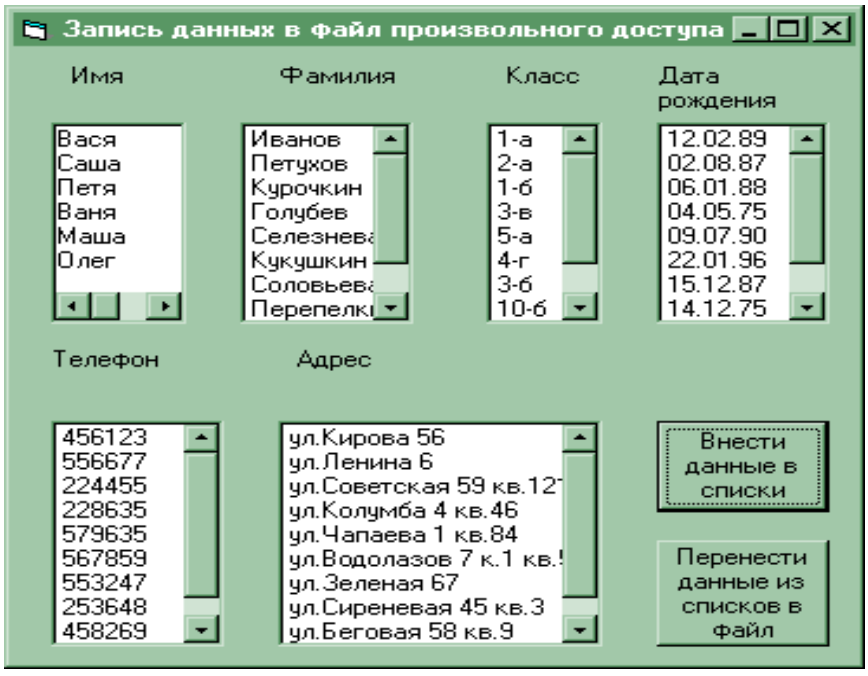


Рисунок 15.1 - Схема записи данных и файла

```
  Case 0: ListName = "имен"
  Case 1: ListName = "фамилий"
  Case 2: ListName = "классов"
  Case 3: ListName = "дат рождений"
  Case 4: ListName = "телефонов"
  Case 5: ListName = "адресов"
 End Select
List(j).AddItem InputBox("Введите очередное значение:", _
          "Ввод данных в список " & ListName)
 Next j
Next i
End Sub
```

Рисунок 15.2 - Схема чтения из файла

```
Private Sub Command1_Click()
Dim n As Integer
Dim i As Integer, Студент As Данные О Студенте
Open "C:\Мои документы\My Documents\Examples\Students.raf" _
  For Random As #1 Len = 94
Print
For i = 1 To 5
Get #1, i, Студент
Print " " & Trim(Студент.Фамилия) & " " & _
Trim(Студент.Имя), Trim(Студент.Класс), _
Trim(Студент.Дата Рождения)
Next i
Close #1
End Sub
```

Операторы переименования, копирования и уничтожения файла.

Оператор переименования файла:

Name *Старое Имя* As *НовоеИмя*

Оператор копирования файла:

FileCopy *Имя Исходного Файла, Имя Файла Копии*

Оператор удаления файла:

Kill *Имя Удаляемого Файла*

Использование индекса для операций над файлом с произвольным доступом.

Для удобства работы с файлами их необходимо *упорядочить*:

- переставить записи файла так, чтобы значения поля были расположены в алфавитном порядке;
- если поле имеет числовое значение, то расположить в порядке его возрастания (убывания).

Такой способ упорядочивания называется сортировкой, а выбранное для этой цели поле - ключом сортировки.

При сортировке в качестве ключа лучше выбрать не одно поле, а *объединение* полей.

Но лучшим решением проблемы является введение еще одного специального поля, которое называется *Идентификационным номером* (IdNumber) или просто *идентификатором записи*.

Идентификатором должно быть *число*, которое *однозначно* определяет запись.

Для представления записи файла в упорядоченном виде, *вместо сортировки всего файла* удобнее поступить следующим образом:

- *создать так называемый индекс* - множество коротких записей. Каждая короткая запись состоит только из двух полей - *Ключа* и *НомераЗаписи*. *Номером Записи* служит номер длинной записи файла (длинная - это полная запись файла, для которого создается индекс);
- затем *отсортировать* этот индекс по возрастанию (или по убыванию) Ключа;
- и выдать длинные записи файла в том порядке, в котором будут расположены их короткие представители в индексе после сортировки.

При работе с файлом произвольного доступа индекс можно использовать для многих целей, например, для *обновления* файла.

Задача обновления заключается в следующем:

- часть сведений в какой-нибудь имеющейся записи нужно заменить новыми данными;
- нужно внести в файл совершенно новую запись;
- нужно удалить какую-нибудь запись из файла.

В отсортированном индексе можно, например, организовать *двоичный поиск.*

Идея алгоритма такого поиска заключается в следующем:

берется номер посредине между максимальным и минимальным номерами записей файлов. Значение ключа искомой записи сравнивается со значением ключа индекса, имеющего данный номер. В зависимости от результата этого сравнения ровно половина номеров отбрасывается, так как

искомая запись заведомо не может иметь номер из этой половины. Затем берется номер посредине оставшихся номеров и все повторяется.

Файлы двоичного доступа.

Кроме текстовых файлов и файлов с произвольным доступом существует еще один тип файлов - *файлы двоичного доступа*.

Открытие файла.

Open *Имя Файла* For Binary As # *Дескриптор Файла*

Закрытие файла.

Close # [*Список Дескрипторов*]

В отличие от файла произвольного доступа, двоичный файл может содержать *записи* различной! *длины*.

Пример.

Пусть файл должен содержать имена и фамилии трех писателей.

```
Private Type ИмяФамилия
  Имя As String
  Фамилия As String
End Type
Dim ИмФам As Имя Фамилия
Private Sub Command1_Click()
Open "Имя Фамилия Bin.bin" For Binary As #1
L% = 1
  For i = 1 To 3
  ИмФам.Имя = InputBox("Введите имя"," Ввод очередного значения_
  в двоичный файл")
  Li% = Len(ИмФам.Имя)
 ИмФам.Фамилия = InputBox ("Введите фамилию"," Ввод очередного
значения        в двоичный файл")
```

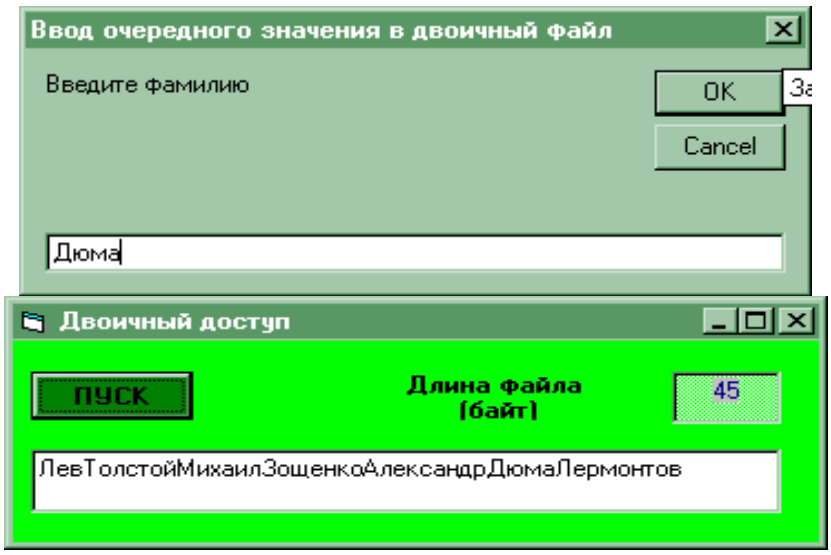


Рисунок 15.3 - Схема ввода и доступа данных

```
 Lf% = Len(ИмФам.Фамилия)
  Put #1, L, ИмФам.Имя
  Put #1, L + Li, ИмФам.Фамилия
  L = L + Li + Lf
  Next i
Text2.Text = LOF(1)
Dim s As String * 100
 Get #1, 1, s
Text1.Text = s
 Close 1
End Sub
```

Лекция 16. Меню

Содержание лекции: редактор меню. Работа с подпрограммой меню.
Цель лекции: научить работать с подпрограммой меню.

Меню представляет собой набор опций (пунктов).

Каждая опция призвана выполнять определенную задачу или совокупность близких по характеру задач, к которым пользователю приходится обращаться довольно часто.

Для того чтобы обратиться к пункту меню, пользователю достаточно щелкнуть мышью этот пункт на линейке меню (рисунок 16.1) или одновременно нажать *Alt+ <подчеркнутая буква>*.

Рисунок 16.1 - Линейка меню

Чаще всего, пункт меню состоит из подпунктов - конкретных задач из данной совокупности. Эти подпункты также называют командами меню.

Для того, чтобы обратиться к той или иной команде меню достаточно щелкнуть мышью этот пункт или нажать указанные справа клавиши, например, *Ctrl+<указанная буква>*.

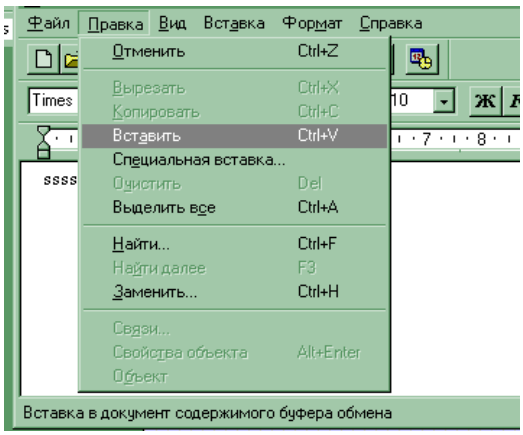


Рисунок 16.2 - Команды пункта меню

Редактор меню.

С помощью Редактора меню (рисунок 16.3) можно создать меню, которое будет выполнять те задачи, которые Вы запрограммируете.

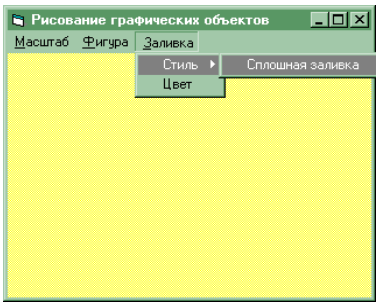


Рисунок 16.3 - Редактор меню

Пример. Создать приложение для рисования различных геометрических фигур.

Меню должно содержать пункты Масштаб (рисунок 16.4), Фигура, Заливка.

Пункт Масштаб-содержать подпункты: *Координаты графического окна* и *Установка масштаба*.

Пункт Фигура содержит - *Квадрат, Прямоугольник, Окружность, Овал.* Каждый из них содержит подпункты: команды для ввода координат и размеров данной фигуры в выбранном масштабе и команду *Нарисовать* для изображения данной фигуры.

Пункт Заливка содержит: *Стиль* (Сплошная заливка, Горизонтальная штриховка, Вертикальная штриховка) и *Цвет заливки фигуры* (Красный, Зеленый, Синий).

Для создания меню необходимо вызвать команду *Menu Editor* из опции Tools Главного меню проекта.

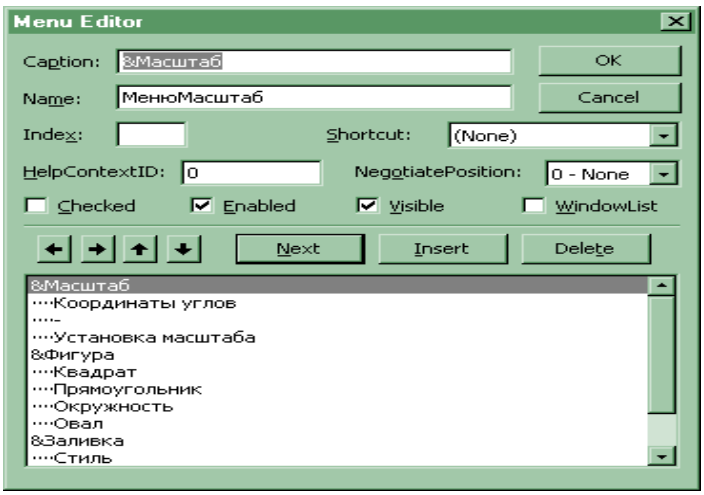


Рисунок 16.4 - Содержание строки меню редактирования

```
Dim xm1 As Integer, xm2 As Integer, _
   ym1 As Integer, ym2 As Integer
Dim R As Integer
Dim x1 As Integer, x2 As Integer, _
   y1 As Integer, y2 As Integer
Private Sub МенюМасштабКоорд_Click()
xm1 = Val(InputBox("Координата x1", "Масштабирование графического окна"))
ym1 = Val(InputBox("Координата y1", "Масштабирование графического окна"))
xm2 = Val(InputBox("Координата x2", "Масштабирование графического окна"))
ym2 = Val(InputBox("Координата y2", "Масштабирование графического окна"))
End Sub
Private Sub МенюМасштабУстМасш_Click()
Form1.Cls
Form1.Scale (xm1, ym1)-(xm2, ym2)
End Sub
Private Sub МенюФигураОкр_Click()
R = Val(InputBox("Введите радиус", "Радиус окружности"))
Form1.Circle (0, 0), R, vbGreen
End Sub
Private Sub МенюФигураПрУг_Click()
x1 = Val(InputBox("Координата x1", "Координаты для прямоугольника"))
y1 = Val(InputBox("Координата y1", "Координаты для прямоугольника"))
x2 = Val(InputBox("Координата x2", "Координаты для прямоугольника"))
```

```
y2 = Val(InputBox("Координата y2", "Координаты для
прямоугольника"))
Form1.Line (x1, y1)-(x2, y2), vbMagenta, B
End Sub
```

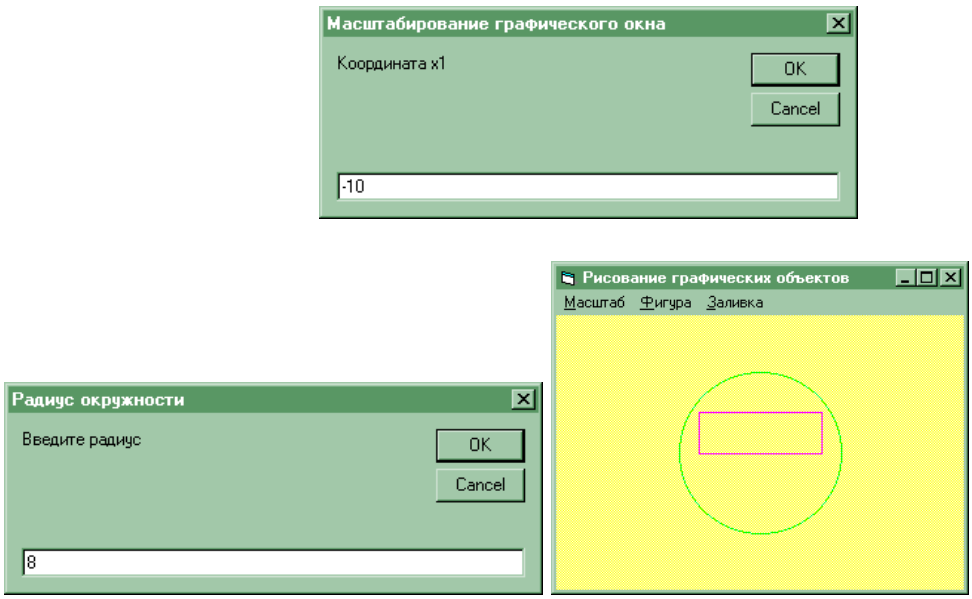


Рисунок 16.5 - Демонстрация работы меню

Меню для работы с файлами (рисунок 16.5).

Создание такого меню осуществляется с использованием Стандартных диалоговых панелей системы Windows.

Каждая из панелей обеспечивает выполнение из 6 следующих функций и выполняется с помощью объектов Общий Диалог:

1) Открытие файла (Open file).

2) Сохранение файла (File Save As).

3) Выбор шрифта (Select Font).

4) Управление печатью (Print).

5) Выбор цвета (Select Color).

6) Получение справочной информации (Help).

Этап 1. Добавление инструмента Общий Диалог (Common Dialog) в набор Tools. Выбрать из опции Project главного меню команду Components. В открывшемся окне выбрать название *Microsoft Common Dialog Control 6.0* и выделить его флажком.

Так выглядит пиктограмма инструмента Common Dialog:

Этап 2. Размещение объекта Общий Диалог на экранной форме. В окне работающего приложения этот объект не виден.

Этап 3. Создание меню на экранной форме. Меню приложения может включать опции - открытие файла, сохранение файла, закрытие файла и т.д.

Этап 4. Создание программного кода.

Пример. Создать приложение, меню которого состоит из 2 опций - Файл и Показать строку. Первая содержит команды: *Создать, Открыть, Закрыть, Сохранить и Сохранить как.*

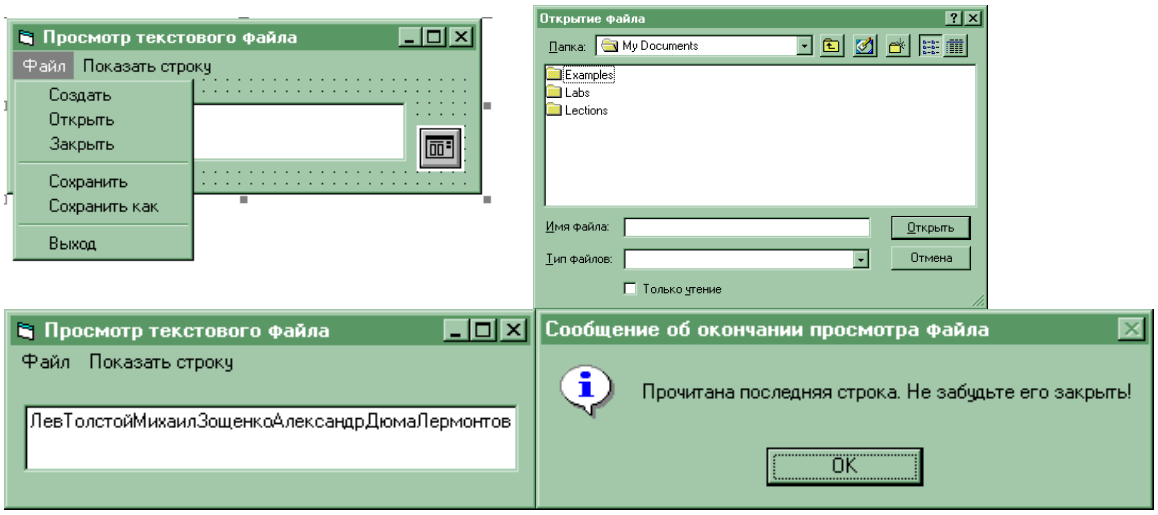


Рисунок 16.6 - Меню для работы с файлами

```
Dim Разрешение As Boolean
Private Sub Form_Load()
Разрешение = False
End Sub
Private Sub МенюФЗакр_Click()
Close #1
End Sub
Private Sub МенюФОткр_Click()
CommonDialog1.ShowOpen
ИмяФайла$ = CommonDialog1.FileName
Open ИмяФайла For Input As #1
Разрешение = True
End Sub
Private Sub МенюПоказСтр_Click()
If Разрешение = True Then
 If EOF(1) = False Then
   Line Input #1, ТекстоваяСтрока$
   Text1.Text = ТекстоваяСтрока
 Else
  MsgBox "Прочитана последняя строка. " & _
```

```
  "Не забудьте его закрыть!", 64, _
  "Сообщение об окончании просмотра файла"
  Разрешение = False
 End If
End If
End Sub
```

Меню для текстового редактора.

Пример. Создать меню, состоящее из 2 пунктов - Шрифт и Цвет. Пункт Цвет включает - Цвет фона и Цвет Символов.

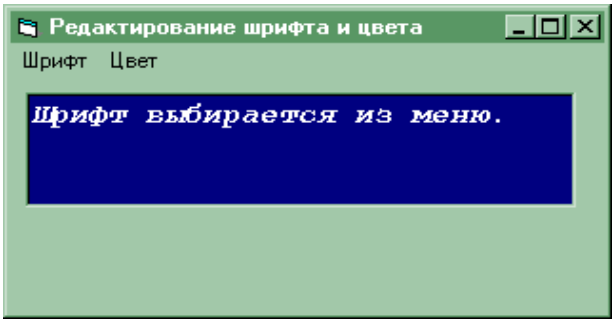


Рисунок 16.7 - Меню для текстового редактора

```
Private Sub МенюЦветСимвол_Click()
CommonDialog1.ShowColor
Text1.ForeColor = CommonDialog1.Color
End Sub
Private Sub МенюЦветФон_Click()
CommonDialog1.ShowColor
Text1.BackColor = CommonDialog1.Color
End Sub
Private Sub МенюШрифт_Click()
CommonDialog1.ShowFont
Text1.Font.Size = CommonDialog1.FontSize
Text1.Font.Name = CommonDialog1.FontName
Text1.Font.Bold = CommonDialog1.FontBold
Text1.Font.Italic = CommonDialog1.FontItalic
'Text1.Font.Strikethrough = CommonDialog1.FontStrikethru
'Text1.Font.Underline = CommonDialog1.FontUnderline
End Sub
```

Лекция 17. База данных

Содержание лекции: понятия базы данных. Виды базы данных. Работа с базами данных.

Цель лекции: дать навыки работы с базами данных.

База данных - это множество наборов единообразных записей.

Единообразие означает, что записи одного набора состоят из одинакового количества полей, и *i*-ые поля всех записей имеют одинаковый тип и размер - длину в символах. Набор единообразных записей, относящихся к определенной теме, называют таблицей базы данных.

Создание файла базы данных.

Среда Visual Basic предоставляет возможность создавать файлы в формате системы *Microsoft Access*. Это программная система, которую можно отнести к категории *СУБД - систем управления базами данных*.

В *Access* заложена технология *Microsoft Jet Database Engine* («пламенный мотор баз данных фирмы Microsoft»), которая обеспечивает возможность работы с базами данных различных форматов.

Файлы баз данных в формате *Access* имеют расширение *.mdb.

Создание файла баз данных начинается с выбора в главном меню опции Add-Ins, а в ней команды Visual Data Manager.

По этой команде вызывается приложение - Диспетчер баз данных (VisData) (рисунок 17.1).

Рисунок 17.1 - Приложение - Диспетчер баз данных

После выполнения последовательности команд File, New, MicrosoftAccess, Version 7.0 MDB на экране появится следующее окно (рисунок 17.2).

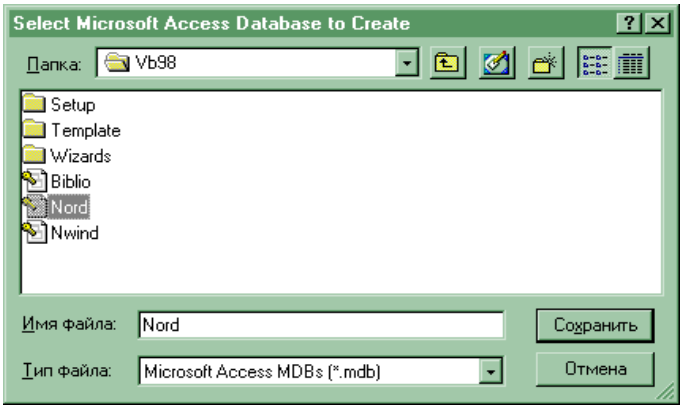


Рисунок 17.2 - Меню базы данных

После щелчка *Сохранить* файл базы данных будет открыт.
На экране появится *Окно базы данных (DatabaseWindow*) (рисунок 17.3).

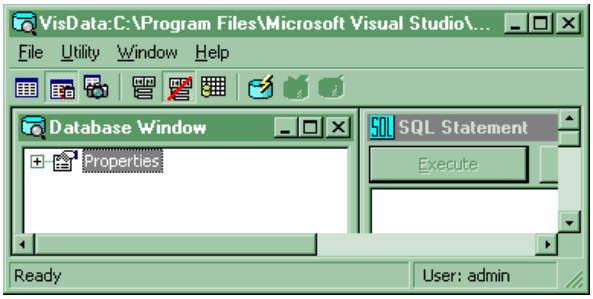
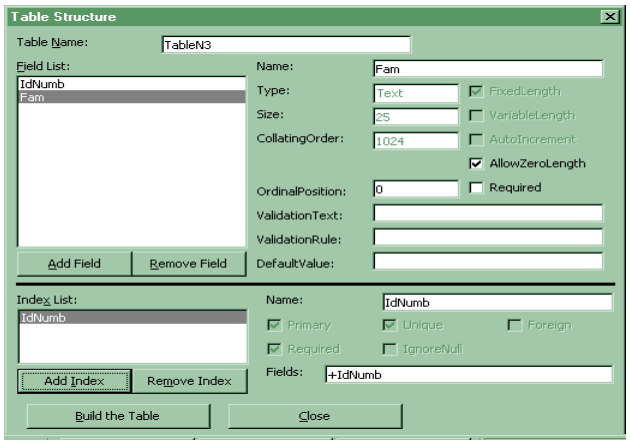


Рисунок 17.3 - Диспетчер баз данных

В любом месте окна *DatabaseWindow* щелкнуть правой клавишей мыши и появится меню, в котором выбрать команду NewTable.

В любом месте окна *DatabaseWindow* щелкнуть правой клавишей мыши и появится меню, в котором выбрать команду NewTable.



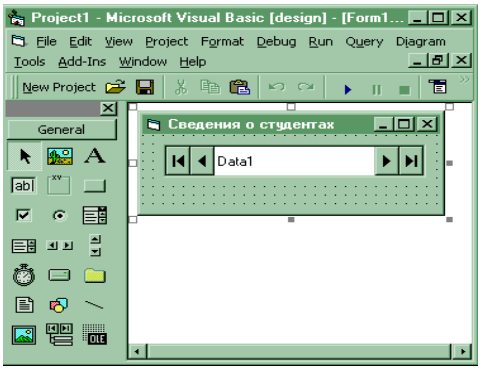


Рисунок 17.4 - Окна базы данных

Для определения полей таблицы необходимо нажать кнопку *AddField (Добавить поле)*.

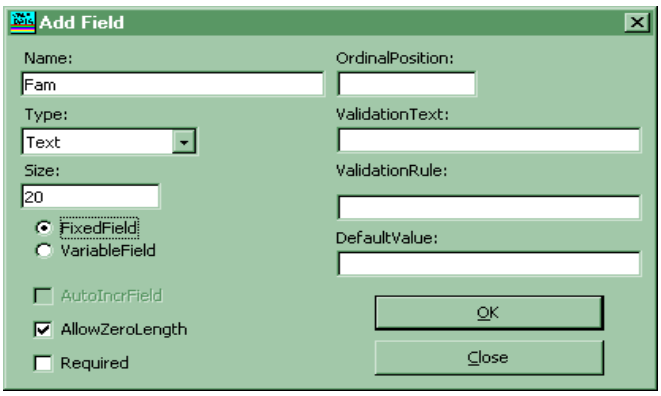


Рисунок 17.5 - Окна базы данных

Чтобы записи таблицы автоматически сортировались по *ключу*, необходимо создать *индекс*, нажав *AddIndex*.

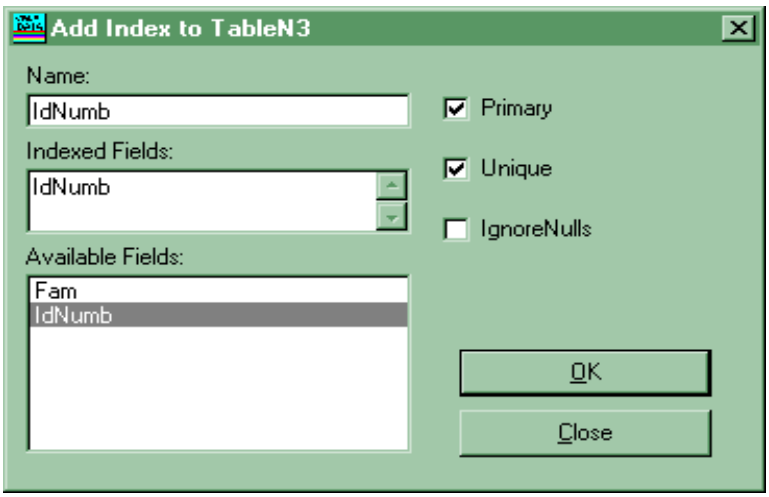


Рисунок 17.6 - Окна базы данных

Заполнение таблицы базы данных.

Одним из способов заполнения таблиц баз данных является создание на экранной форме проекта объекта класса *Данные (Data*).

Вторым шагом является *привязка* объектов к объекту *Данные (Data).*

В качестве привязываемых объектов *(bound - объектов*, bound - форма past англ. глагола bind) могут выступать текстовые поля (TextBox), окна списков (ListBox), комбинированные окна (ComboBox), графические поля (PictureBox) и др.

Привязка bound - объекта к объекту *Данные* - это установка значений некоторых свойств bound - объекта, которая делает его содержимое равным значению, взятому из базы данных.

Трем свойствам каждого из двух объектов *Data* должны быть присвоены следующие значения:

- свойству *Connect* - значение Access
- свойству *DatabaseName* - путь к файлу *.mdb
- свойству *RecordSource* - значение TableN3.

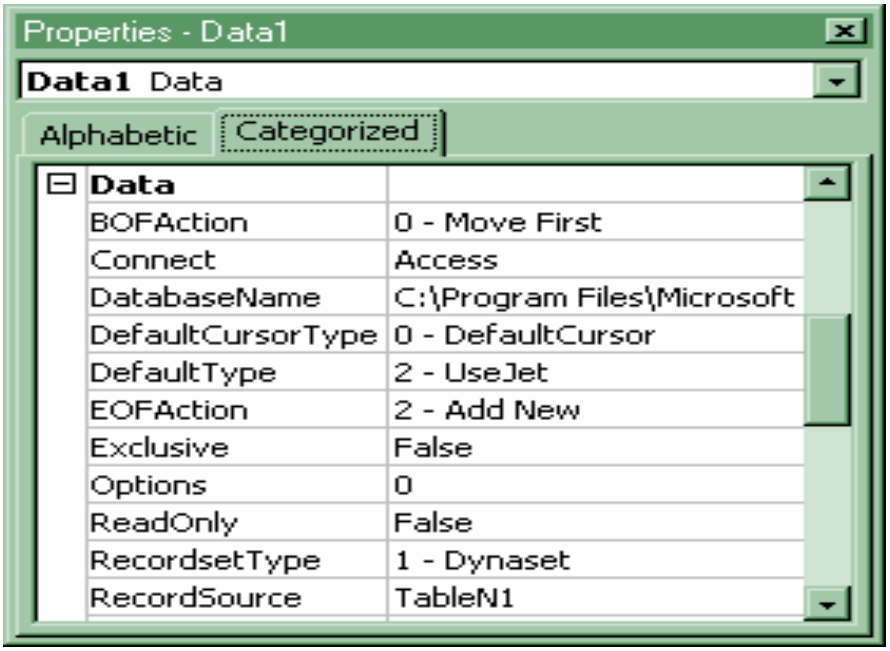


Рисунок 17.7 - Свойства окна базы данных

Двум свойствам текстового поля установить значения:

- свойству *DataSource* - имя объекта *Data;*
- свойству *DataField*- имена полей таблиц.

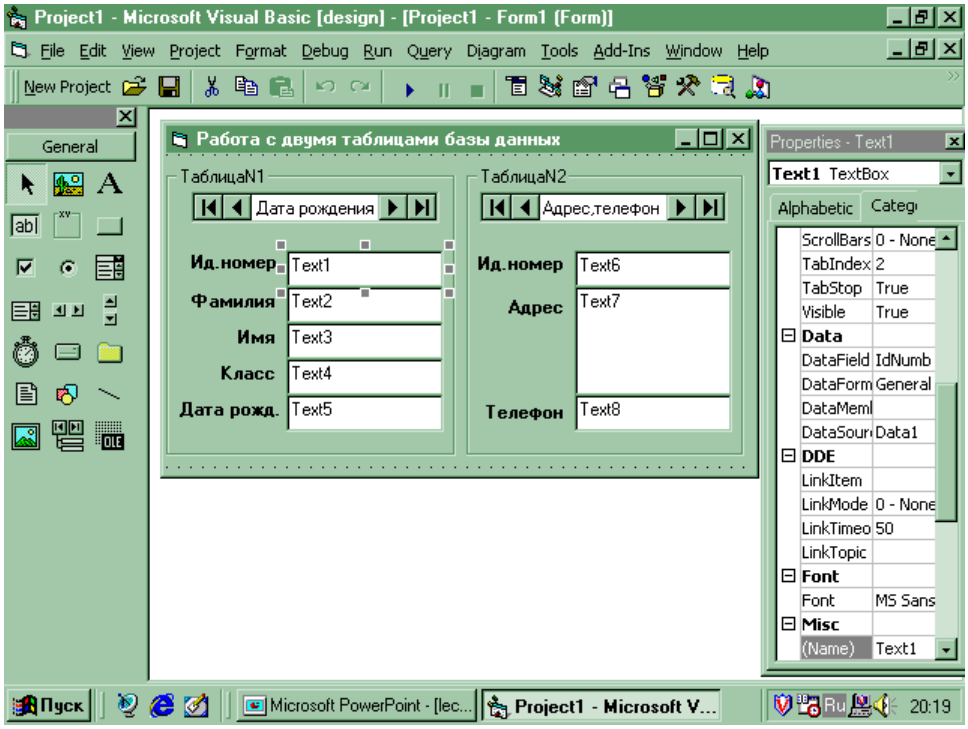


Рисунок 17.8 - Заполнение таблицы данных окна базы данных

Пример.

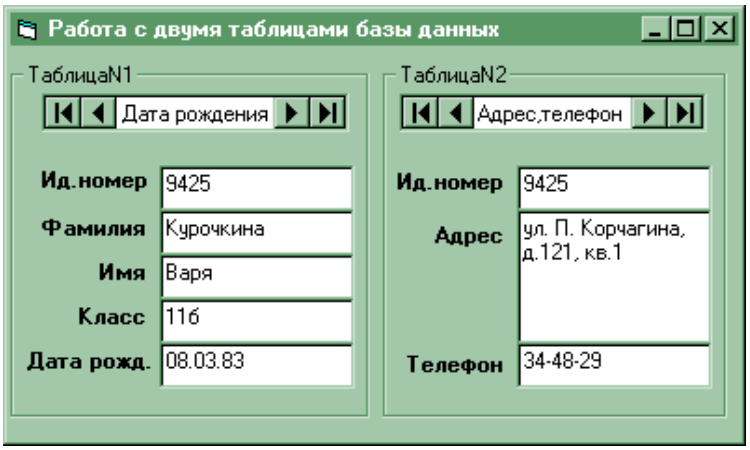


Рисунок 17.9 - Работа с двумя таблицами базы данных

Лекция 18. Анимация. Способы создания анимации

Содержание лекции: понятие анимации. Файлы анимации. Способы создания анимационных файлов.

Цель лекции: научить создавать анимационные файлы.

Существует два способа создания анимации:

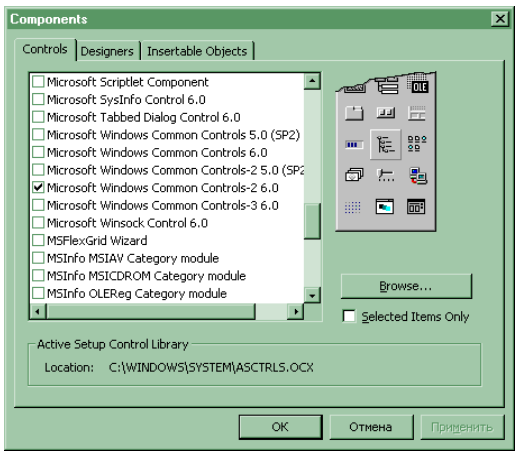
1) Использование уже созданного кем-то файла анимации.

2) Создание собственного файла анимации и программирование его демонстрации.

1. Использование уже созданного кем-то файла анимации.

*Расширение файлов с анимацией *.avi.*

Файлы avi можно прокручивать в любом приложении - для этого надо поместить на экранную форму объект управления Анимация (рисунок 18.1).



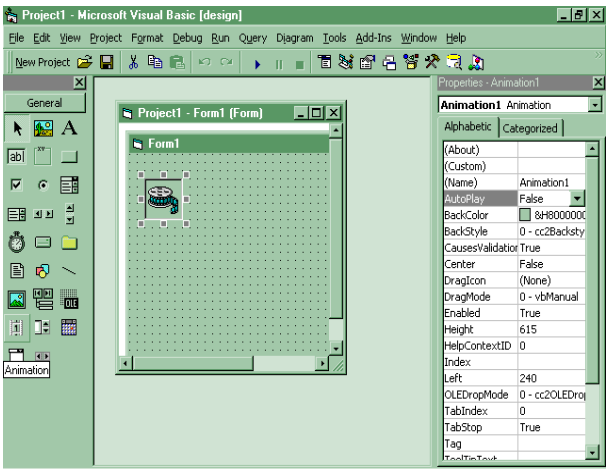


Рисунок 18.1 - Форма для создания мультипликации

Для поиска и открытия файла *avi* (который привязывается к объекту Анимация) на экранную форму желательно поместить объект Общий диалог (рисунок 18.2-18.3).

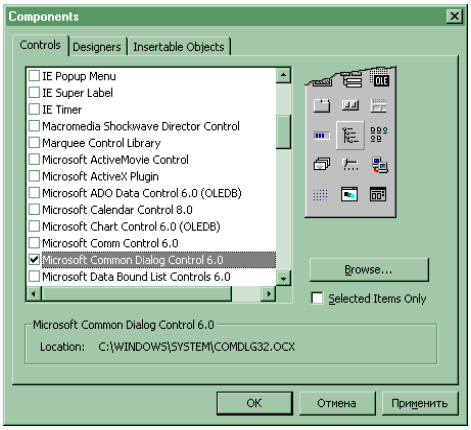


Рисунок 18.2 - Окно свойств «Общий диалог»

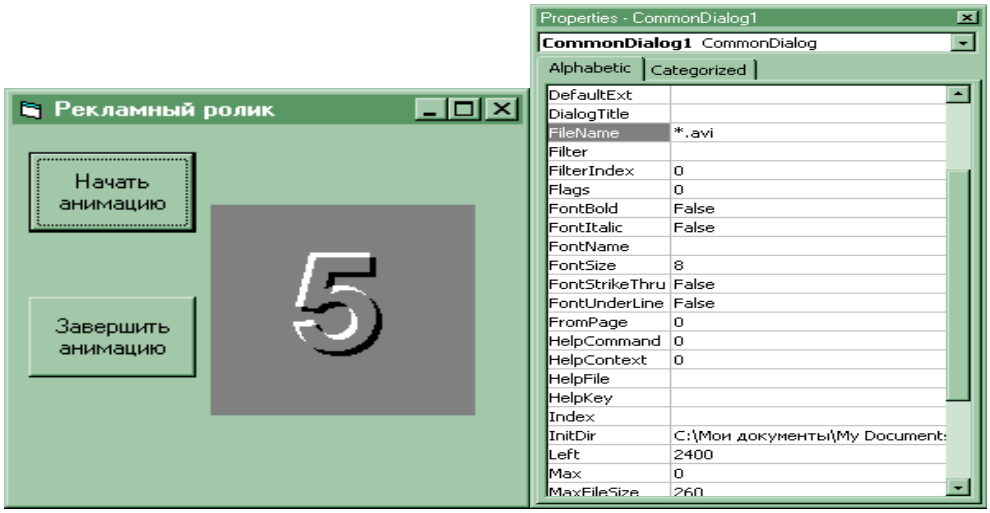


Рисунок 18.3 - Свойства анимации

```
Private Sub Animation1_Click()
End Sub
Private Sub Command1_Click()
'Начатьдемонстрацию
CommonDialog1.ShowOpen
Count16$ = CommonDialog1.FileName
Animation1.Open Count16: Animation1.Play
End Sub
Private Sub Command2_Click()
'Завершитьдемонстрацию
```

```
Animation1.Close
EndSub
```

2. Создание собственного файла анимации и программирование его демонстрации.

Инструментами для создания графических объектов являются PictureBox и Image из стандартного набора Toolbox.

Идея создания анимации заключается в следующем:

На экранную форму помещается несколько графических объектов, в которые загружаются кадры будущего ролика (мультфильма).

Свойство Visible позволяет делать невидимыми нужные кадры (рисунок 18.4).

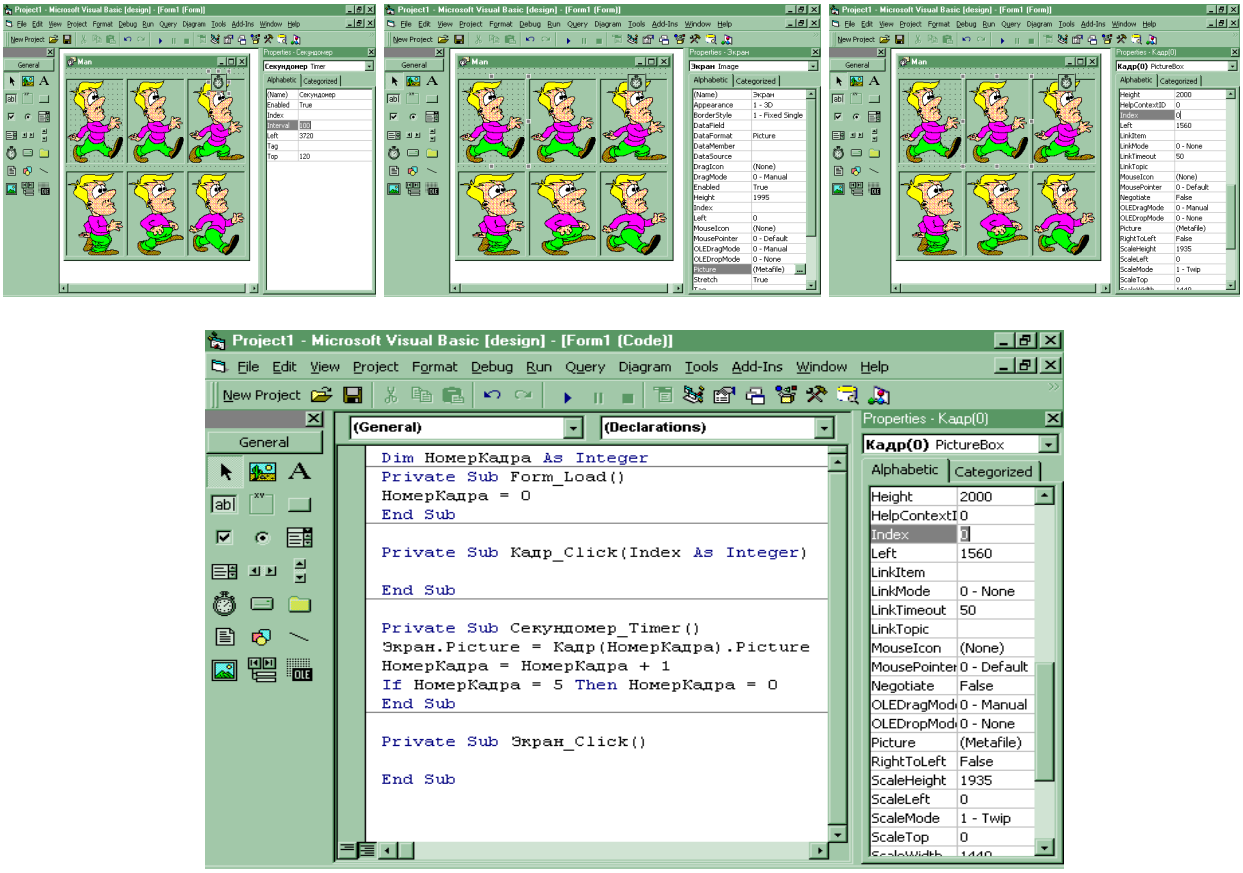


Рисунок 18.4 - Создание мультфильмов

Лекция 19. Дополнительные средства проектирования приложений.

Содержание лекции: линейка инструментов с кнопками. Полоса прокрутки.

Цель лекции: научить пользоваться дополнительными возможностями программы.

Линейка инструментов с кнопками.

Для работы могут быть необходимы 3 следующие группы инструментов:

- Microsoft Windows Common Controls 6.0;
- Microsoft Windows Common Controls-2 6.0;
- Microsoft Windows Common Controls-3 6.0.

Пример 1 (рисунок 19.1, 19.2).

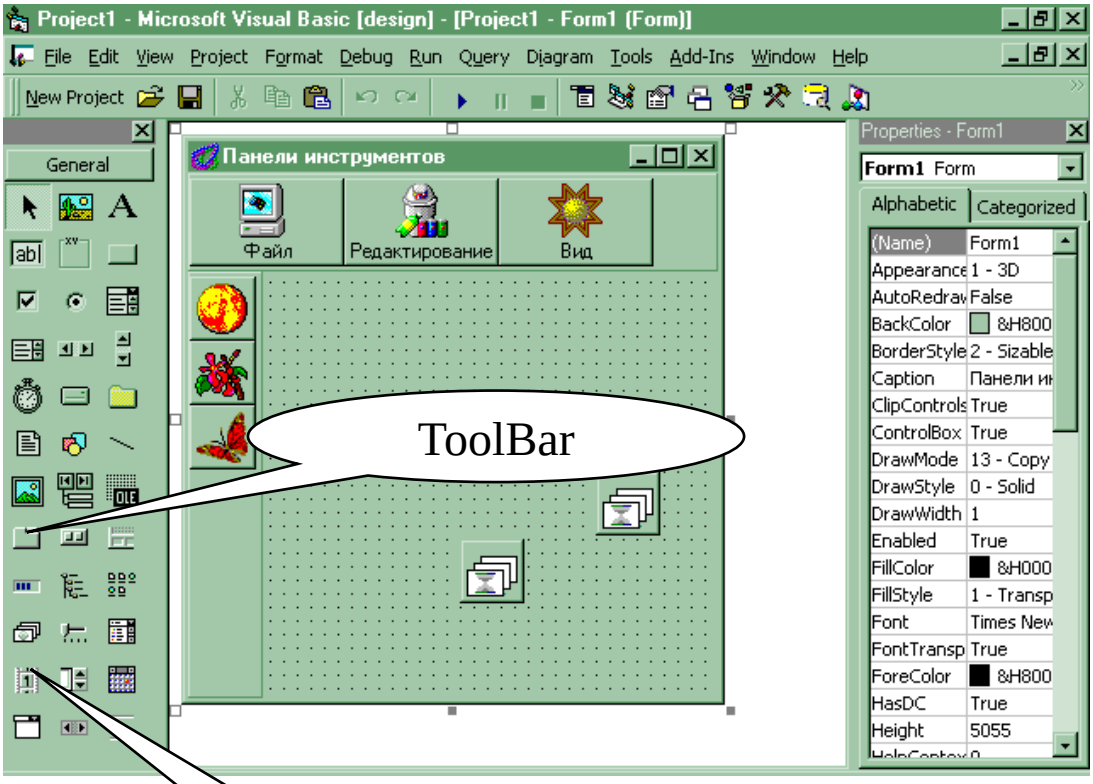
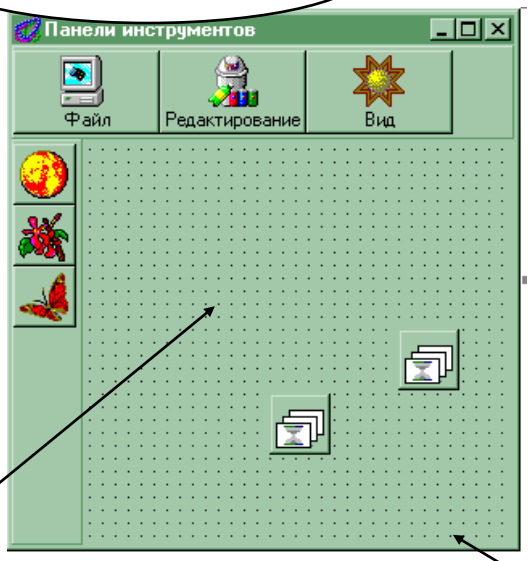


Рисунок 19.1 - Линейка инструментов



Линейки инструментов

Объекты управления Списокизображени

Рисунок 19.2 - Объекты управления

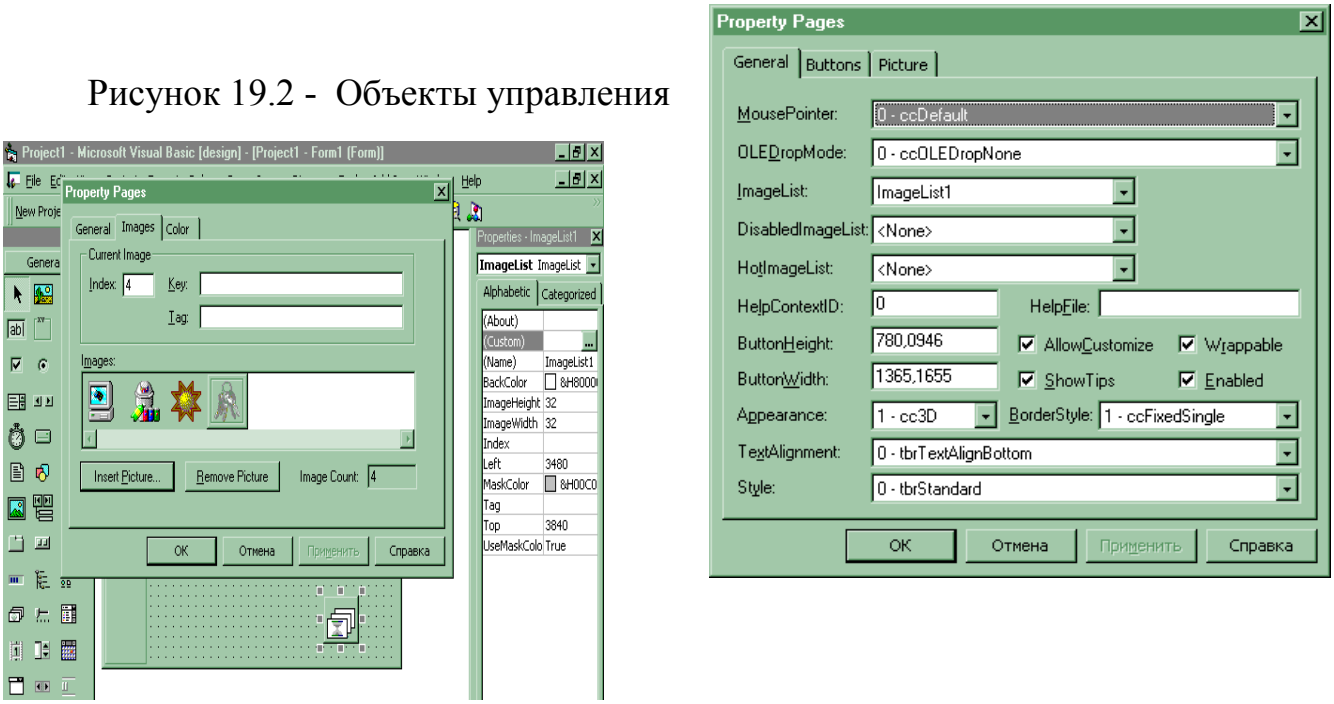


Рисунок 19.3 - Свойства Линейки инструментов

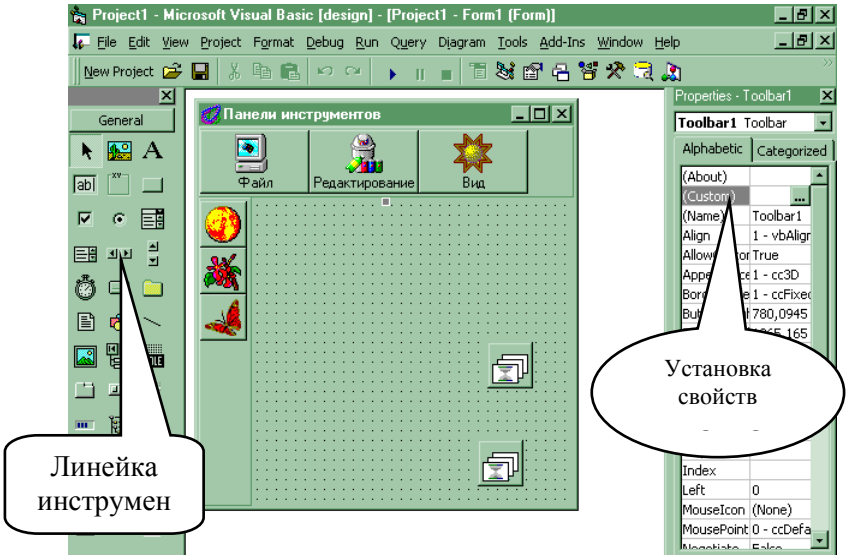


Рисунок 19.4 - Верхняя часть Панели установки свойств объекта

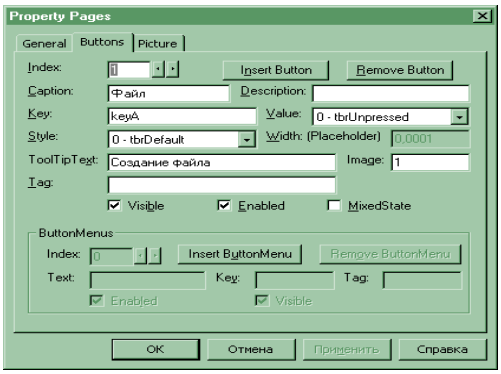


Рисунок 19.5 - Линейка инструментов с открытой закладкой General

При задержке указателя мыши в работающем приложении над кнопкой появляется всплывающее окно подсказки (свойство ToolTipText) (рисунок 19.6).

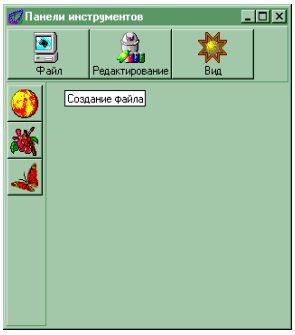


Рисунок 19.6 - Окно подсказки

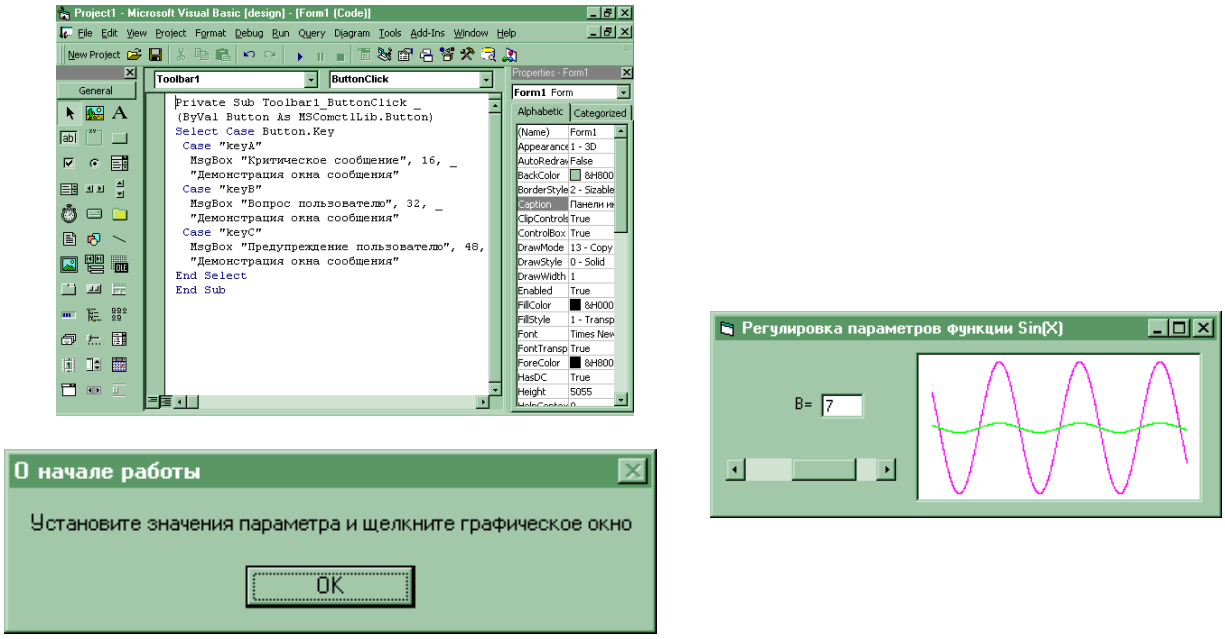


Рисунок 19.7 - Пример программного кода приложения
и полоса прокрутки

Примерный код программы с использованием полосы прокрутки (рисунок 19.8).

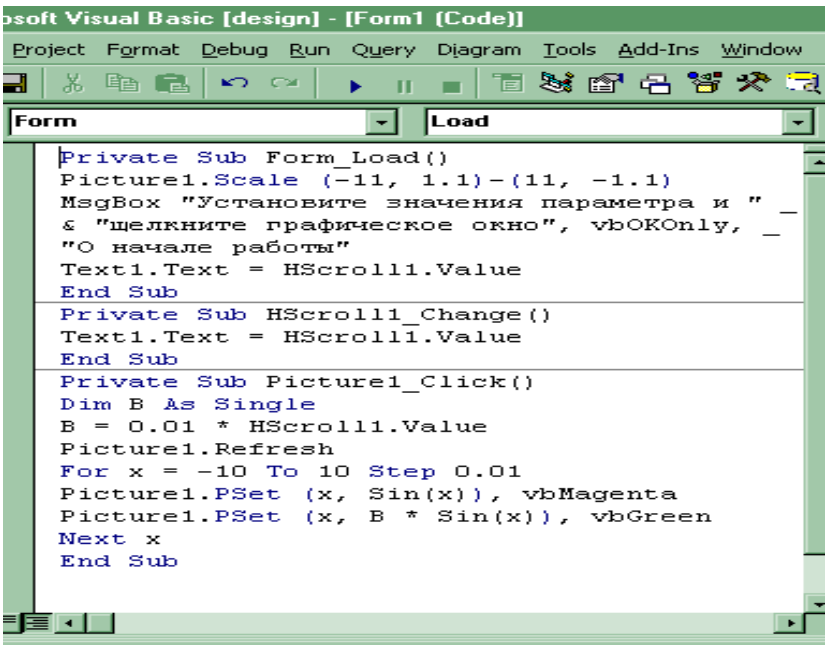


Рисунок 19.8 - Примерный код программы с использованием полосы прокрутки

```
Private Sub SliderRed_Scroll()
SliderRed.Text = "Red = " &Str(SliderRed.Value)
End Sub
```

Лекция 20. Дополнительные средства отображения информации

Содержание лекции: дать сведения о дополнительных средствах информации.

Цель лекции: научить использовать дополнительные средства для отображения информации.

Объекты управления:

1) Полоса закладок TabStrip.
2) Строка состояния StatusBar.
3) Полоса процесса ProgressBar.

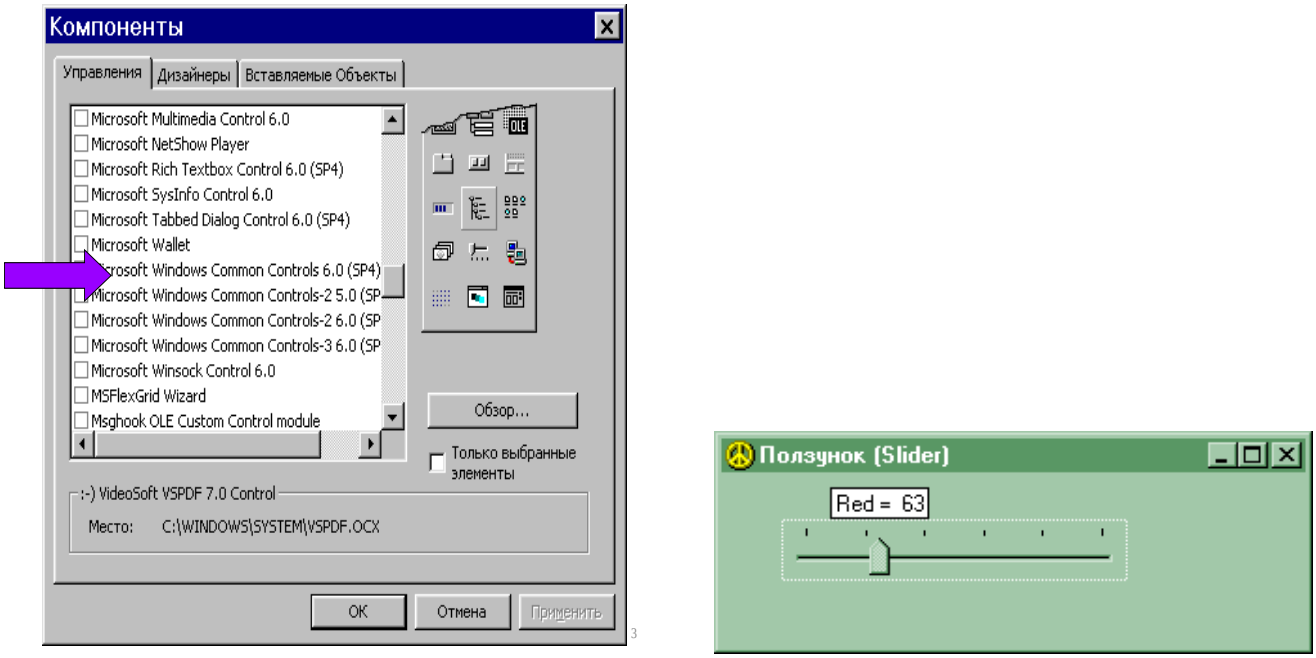


Рисунок 20.1 - Окно свойств объектов управления

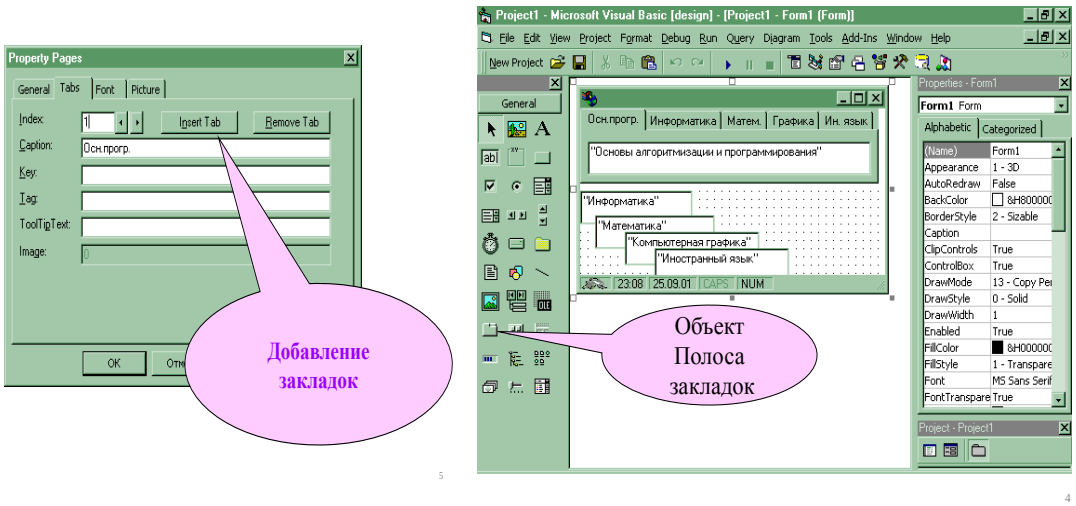


Рисунок 20.2 - Кнопки управления и
полоса прокрутки

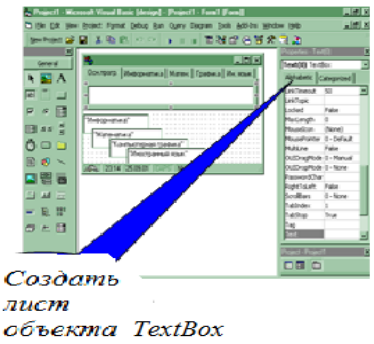


Рисунок 20.3 - Кнопки управления программными приложениями

Рисунок 20.4 - Кнопки управления. Окно свойств

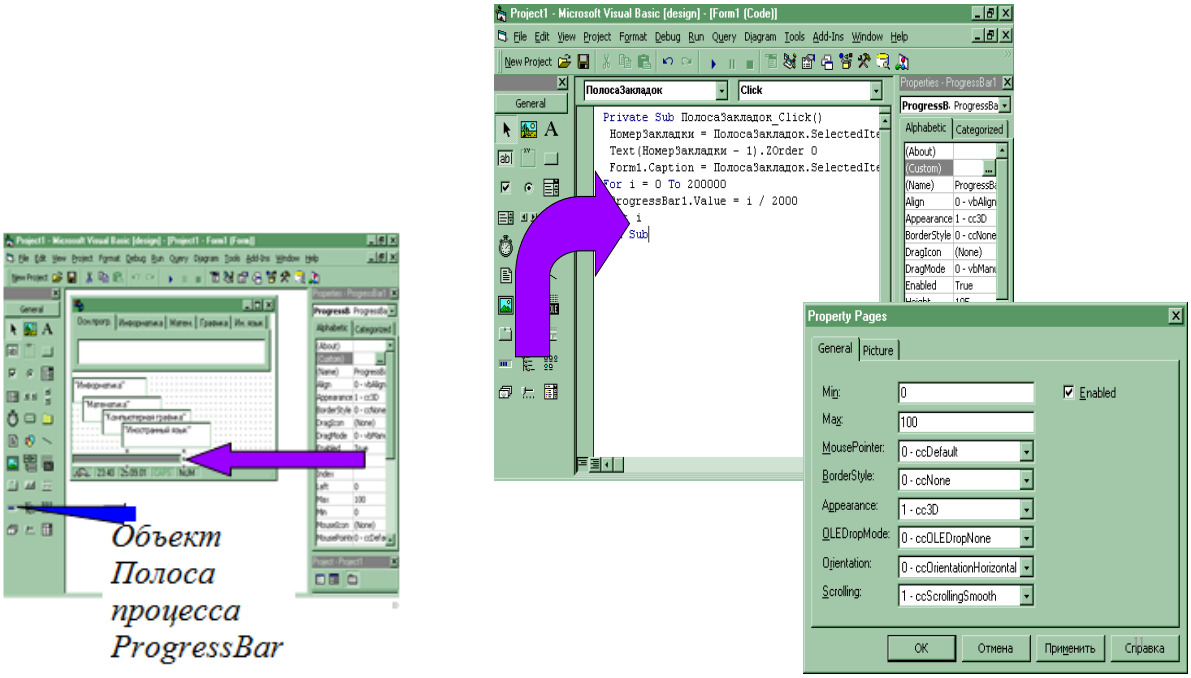


Рисунок 20.5 - Кнопки управления

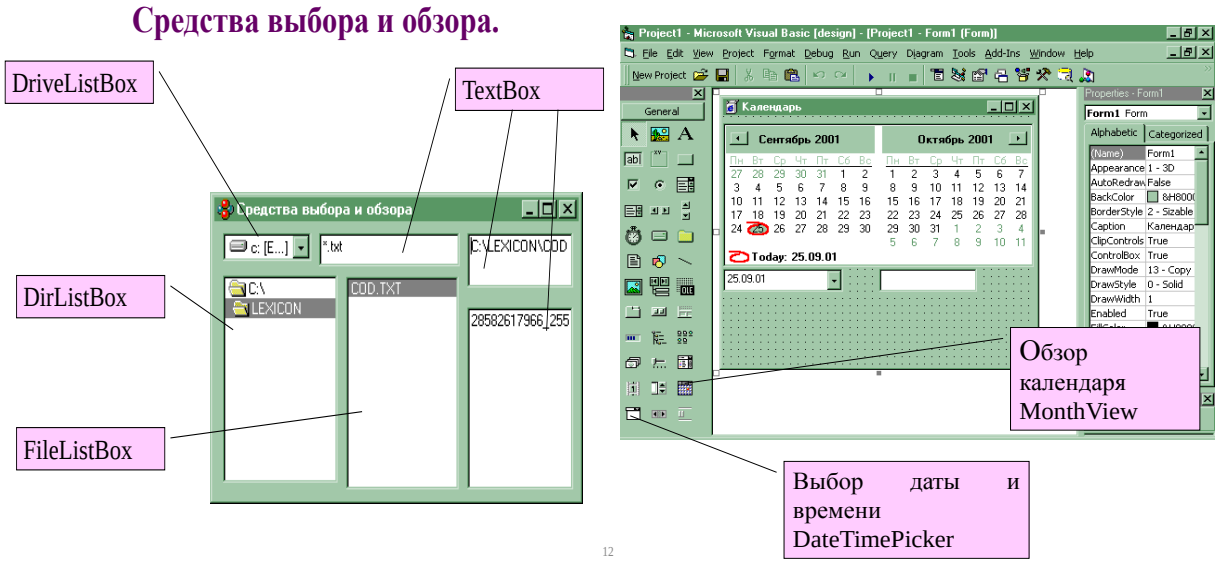


Рисунок 20.6 - Кнопки управления. Средства выбора и обзора

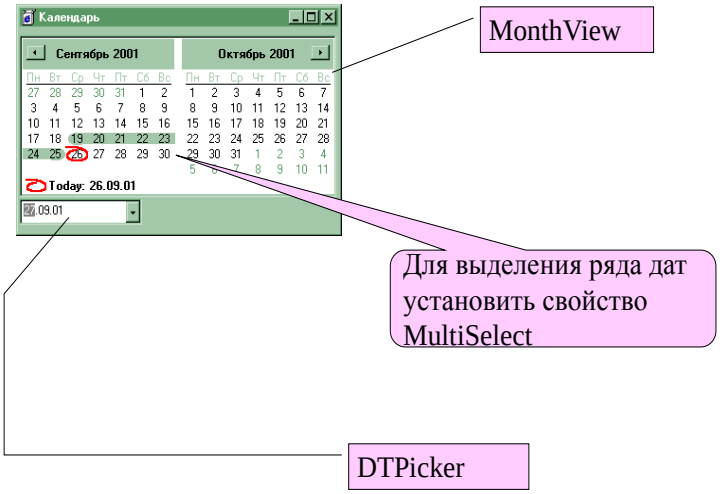
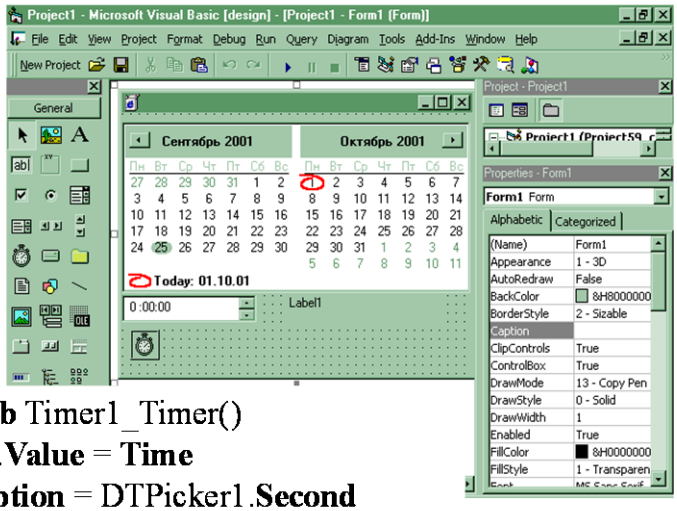


Рисунок 20.7 - Кнопки управления

Секундомер




```
Private Sub Timer1_Timer()
DTPicker1.Value = Time
Label1.Caption = DTPicker1.Second
End Sub
```

Рисунок 20.8 - Фрагмент программы «Секундомер» и Кнопки управления

Объект UpDown

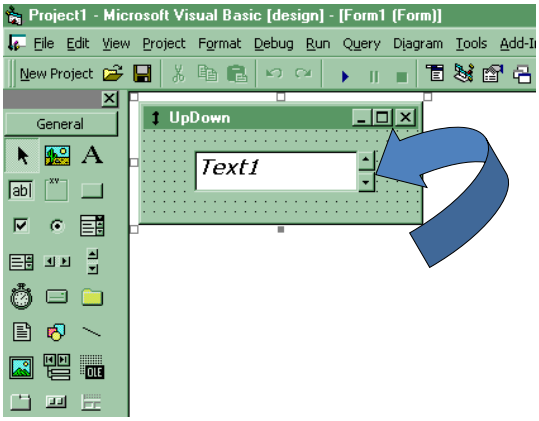


Рисунок 20.9 - Кнопки управления объектом

Свойства объекта

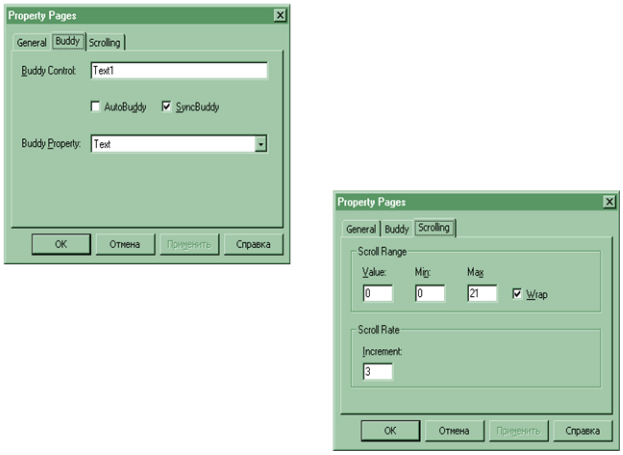


Рисунок 20.10 - Кнопки управления «Свойства объекта»

Лекция 21. Объекты и программирование

Содержание лекции: понятия объекта. Свойства объектов. Применение объектов.

Цель лекции: научить применять объекты в программировании.

Объекты являются инкапсулированными (encapsulated), то есть они содержат и свой код, и свои данные.

Объекты Visual Basic имеют свойства, методы и события. Свойства – это данные, которые описывают объект.

Методы – это то, что объект «умеет» делать.

События – это начало или завершение любого действия в программе, инициируемого пользователем или генерируемого самой программой или окружением (средой). Можно написать код, который будет выполняться, если произойдут определенные события.

Объекты в Visual Basic создаются из классов; говорят, что объект есть экземпляр класса. Класс определяет интерфейсы объекта, область его действия, и при каких обстоятельствах он может быть создан.

Краткое изложение принципов.

Фундаментальные характеристики ООП:

1) Все является объектом.

2) Вычисления осуществляются путем взаимодействия (обмена данными) между объектами, при котором один объект требует, чтобы другой объект выполнил некое действие.

Объекты взаимодействуют, посылая и получая сообщения. Сообщение — это запрос на выполнение действия, дополненный набором аргументов, которые могут понадобиться при выполнении действия.

3) Каждый объект имеет независимую память, которая состоит из других объектов.

4) Каждый объект является представителем класса, который выражает общие свойства объектов (таких, как целые числа или списки).

5) В классе задается поведение (функциональность) объекта. Тем самым все объекты, которые являются экземплярами одного класса, могут выполнять одни и те же действия.

6) Классы организованы в единую древовидную структуру с общим корнем, называемую иерархией наследования. Память и поведение, связанное с экземплярами определенного класса, автоматически доступны любому классу, расположенному ниже в иерархическом дереве.

Класс – это производный структурированный тип, введенный программистом на основе уже существующих типов. Механизм классов позволяет создавать типы в полном соответствии с принципами абстракции данных, т.е. класс задает некоторую структурированную совокупность типизированных данных и позволяет определить набор операций над этими данными.

Лекция 22. Обработка мероприятий мыши. Мероприятие Click

Содержание лекции: дать понятия об обработке мероприятия мыши. Мероприятие Click.

Цель лекции: научить применять различные мероприятия мыши.

Мероприятие мыши:

- генерируется при щелчке мышью на составляющей управления;
- поддерживается практически всеми элементами Visual Basic, но почаще всего применяется для кнопок;
- достаточно два раза щелкнуть на элементе при разработке формы, чтобы появилась заготовка обработчика события Click.

Пример:

- добавьте на форму три метки Label и кнопку Command (рисунок 22.1);
- пусть при щелчке мышью на любой из меток её заголовок отобразится на кнопке, а сама метка станет невидимой. Щелчок же на кнопке вернет все значения свойств элементов в исходное состояние;
- в событиях Click каждого элемента запишите приведенный ниже код:

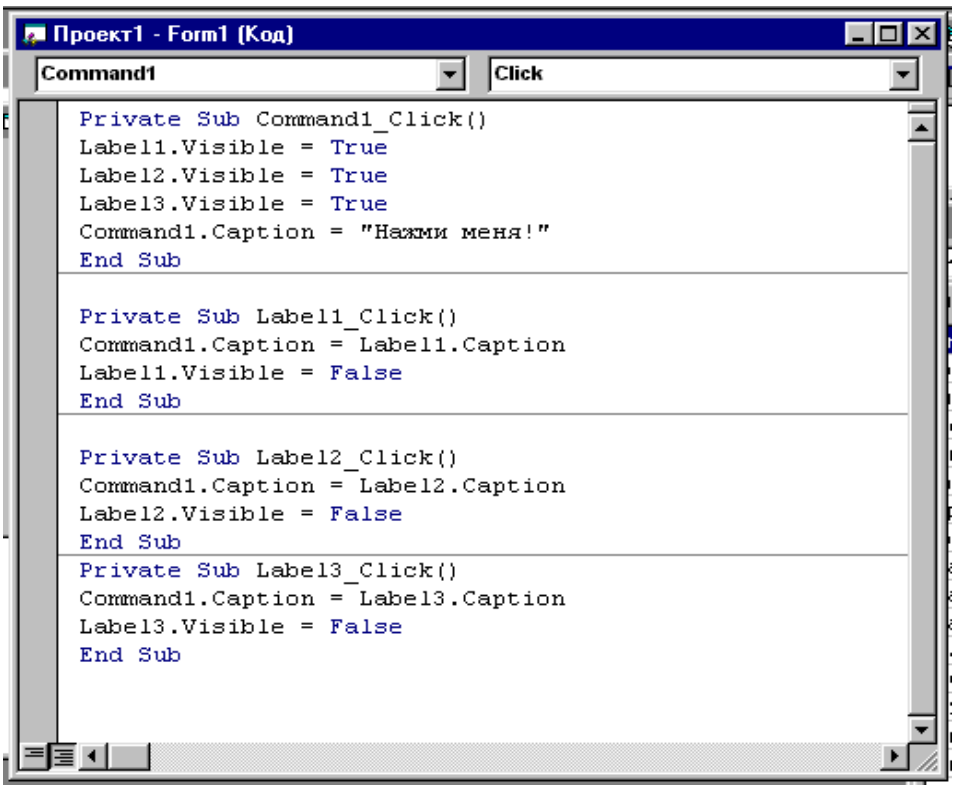


Рисунок 22.1 - Форма мероприятия Click

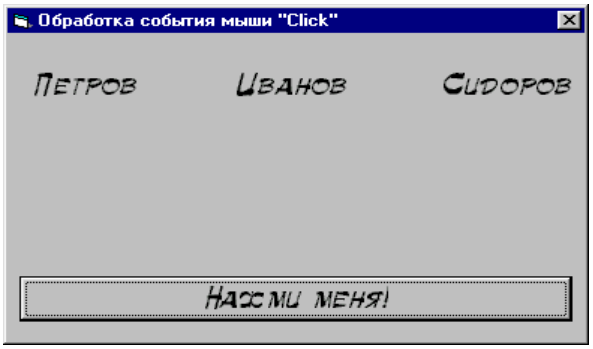


Рисунок 22.2 - Начальный запуск приложения

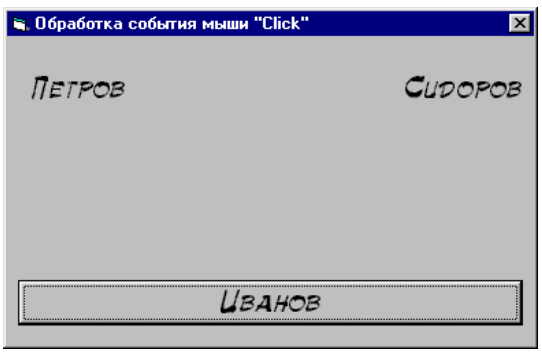


Рисунок 22.3 - Пример работающего мероприятия

Событие MouseDown

- возникает, если нажать кнопку мыши и не отпускать её;
- процедура события имеет следующий синтаксис:

```
Private Sub target_MouseDown(Button As Integer,
```

Событие MouseDown

```
Shift As Integer, X As Single, Y As Single)
End Sub
```

где target - это объект, для которого обрабатывается событие, а передаваемые параметры имеют следующие значения:

1) Целый параметр Button определяет, какая из кнопок мыши нажата:

- 1 - левая кнопка;
- 2 - правая кнопка;
- 4 - средняя кнопка.

Сумма этих величин определяет различные сочетания кнопок. Например, если Button=5, то это значит, что были нажаты одновременно левая и средняя кнопка мыши и т.д.

2) Целочисленный параметр Shift описывает состояние клавиш Shift, Ctrl и Alt на момент нажатия кнопки:

- 1 - нажата клавиша Shift;
- 2 - нажата клавиша Ctrl;
- 4 - нажата клавиша Alt.

Как и в предыдущем случае, сумма этих величин определяет различные сочетания клавиш, например: 6 - нажаты Ctrl и Alt, 3 - нажаты Ctrl и Shift.

X и Y определяют положение указателя.

Событие MouseUp.

- генерируется, если отпустить нажатую кнопку мыши;
- процедура события имеет следующий синтаксис:

```
Private Sub target_MouseUp(Button As Integer, _
Shift As Integer, X As Single, Y As Single)
End Sub
```

Её параметры совпадают с параметрами процедуры MouseDown.

В следующем примере рассматривается приложение, в котором, если нажать и не отпускать кнопку мыши над одной из игральных карт, её изображение появится на картинке в левом верхнем углу, а если отпустить кнопку мыши - карта перевернётся.

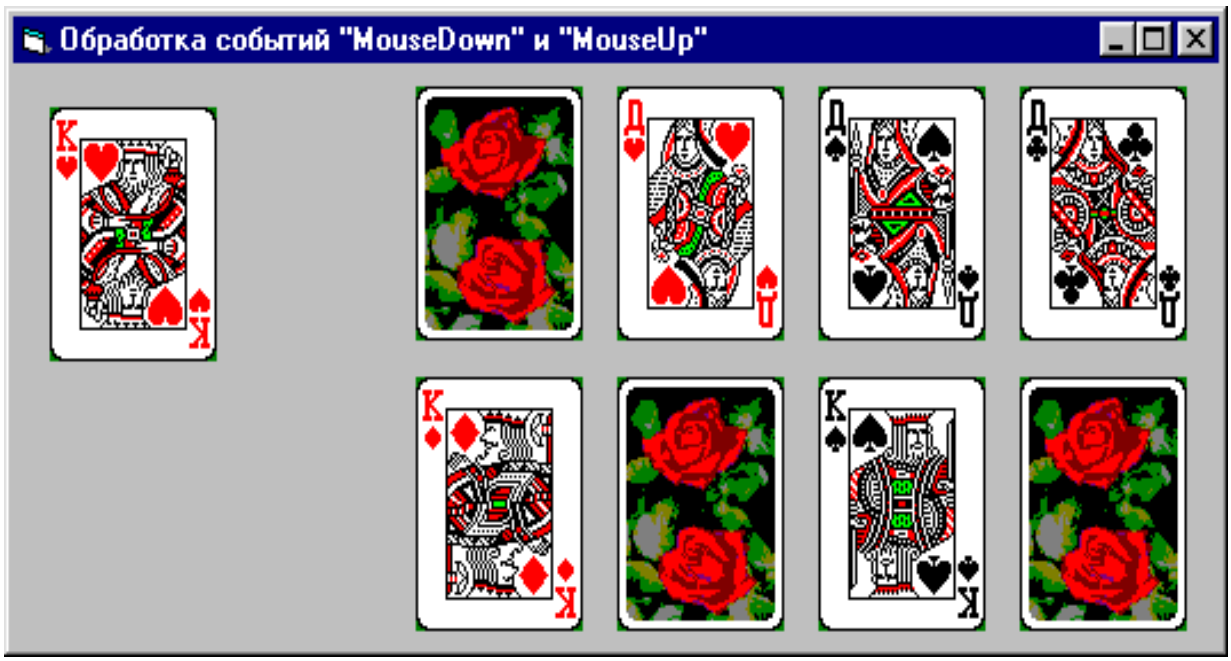


Рисунок 22.4 - Иллюстрация действий с картами

Рисунок 22.5 - Пояснение действий с картами

Следующий программный код реализует данное приложение.

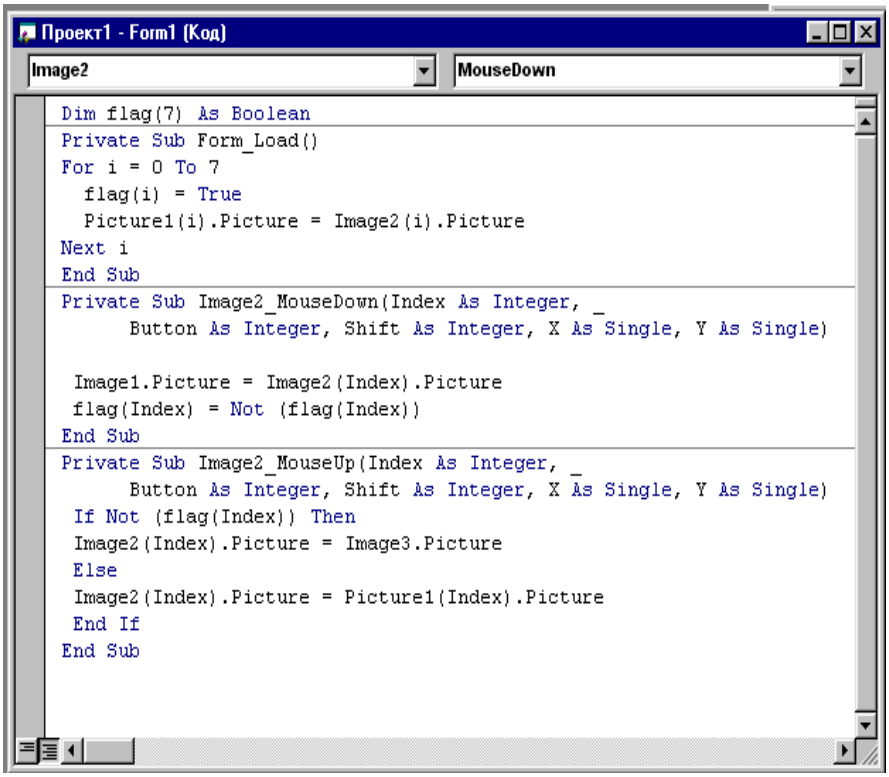


Рисунок 22.6 - Программный код реализации действий с картами

Событие MouseMove.

- генерируется при перемещении мыши;
- процедура события имеет следующий синтаксис:

Private Sub target_MouseMove(Button As Integer, _ Shift As Integer, X As Single, Y As Single) End Sub;

- ее параметры совпадают с параметрами процедур MouseDown и MouseUp

В следующем приложении при перемещении мыши на форме появляются окружности, имеющие случайный радиус и цвет.

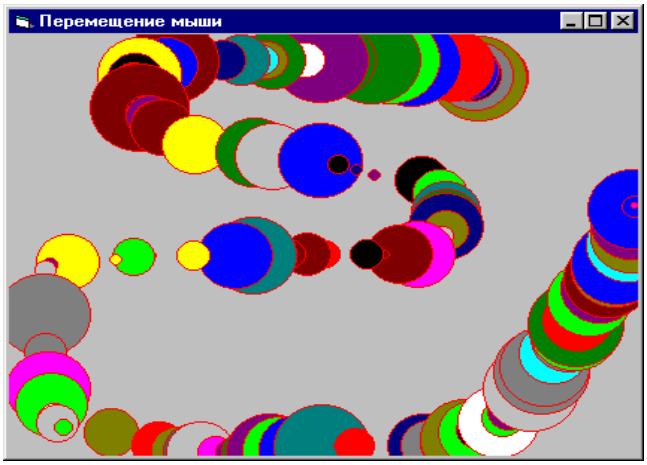


Рисунок 22.7 - Иллюстрация требуемых действий

Следующий программный код реализует требуемые действия.

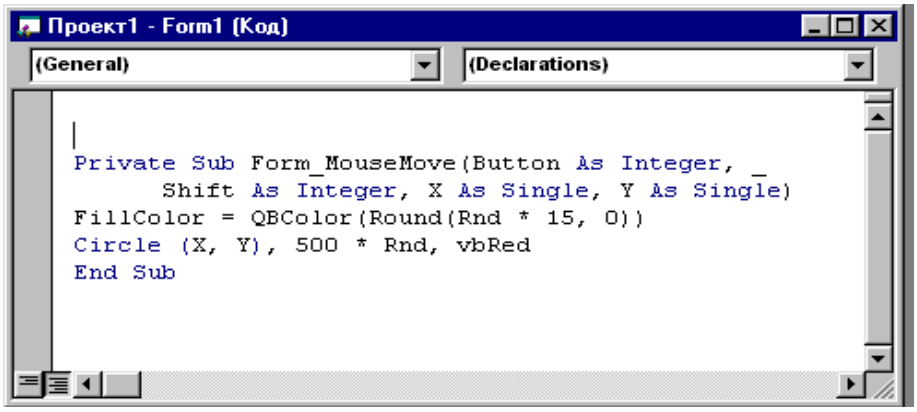


Рисунок 22.8 - Программный код реализации требуемых действий

Лекция 23. Обработка событий клавиатуры

Содержание лекции: понятие события клавиатуры. Применение события клавиатуры.

Цель лекции: научить обрабатывать события клавиатуры.

Программирование реакции приложения на события клавиатуры называют созданием *программного обработчика клавиатуры*. Обработчик клавиатуры может работать на двух уровнях:

а) уровень элемента управления;

б) уровень формы:

- обработчик *уровня элемента управления* (обработчик *нижнего уровня*) позволяет программировать конкретный элемент управления. Например, можно преобразовать весь набранный в текстовом поле текст в прописные буквы.

- обработчик *уровня формы* позволяет сделать так, чтобы форма первая реагировала на события клавиатуры. Затем фокус может быть перемещен на элемент управления.

Обработчик клавиатуры нижнего уровня.

Visual Basic предоставляет три события, которые распознаются формами и элементами управления, допускающими ввод с клавиатуры:

1) KeyPress - нажата клавиша, соответствующая символу ASCII.

2) KeyDown - нажата любая клавиша на клавиатуре.

3) KeyUp - отпущена любая клавиша на клавиатуре.

События *KeyDown* и *KeyUp* следует использовать для определения состояний, которые не может определить событие *KeyPress*, а именно:

- *комбинации клавиш <Shift>, <Ctrl>, <Alt>;*

- *нажатие клавиш со стрелками.* (Некоторые элементы управления (кнопки, переключатели, флажки) не получают событий при нажатии таких

клавиш: их нажатие приводит к перемещению на следующий элемент управления);

- *нажатие клавиш <PageUp>,<PageDown>;*
- распознавание нажатия цифровых клавиш на вспомогательной цифровой клавиатуре и на основной клавиатуре;
- реагирование не только на нажатие, но и на отпуск клавиши. (Событие *KeyPress* отвечает только на нажатие);
- нажатие функциональных клавиш, не связанных с командами меню.

Если пользователь нажимает клавишу, генерируются события *KeyDown* и *KeyPress*, за которыми следует событие *KeyUp*, когда пользователь отпускает клавишу. Если *KeyPress* не распознает нажатую клавишу, происходит только событие *KeyDown*, за которым следует событие *KeyUp*.

Событие KeyPress.

Событие *KeyPress* происходит, когда нажата любая клавиша, соответствующая стандартному символу ASCII-кода (только буквы, цифры, знаки пунктуации и клавиши <Enter>, <Tab>, <BackSpace>).

Например, чтобы перевести все символы в текстовом поле в символы верхнего регистра, следует использовать это событие для изменения регистра симво-лов при их вводе:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
KeyAscii = Asc(UCase(Chr(KeyAscii)))
End Sub
```

Параметр *KeyAscii* содержит целое значение, соответствующее ASCII-коду символа. Функция Chr - преобразует код в соответствующий символ; Ucase - переводит символ в верхний регистр, Asc - возвращает код символа, передаваемого в качестве параметра.

Пример: Создать приложение, в котором, при нажатии на любую клавишу, на форме будет распечатываться ее ASCII-код (рисунок 23.1).

Приложение будет работать, если для совершенно пустой формы в новом проекте написать следующий код:

```
Private Sub Form_KeyPress(KeyAscii As Integer)
Print «’ »; Chr(KeyAscii); « ’»; « = » ; KeyAscii
End Sub
```

Приложение будет работать, если для совершенно пустой формы в новом проекте написать следующий код:

```
Private Sub Form_KeyPress(KeyAscii As Integer)
Print «’ »; Chr(KeyAscii); « ’»; « = » ; KeyAscii
End Sub
```

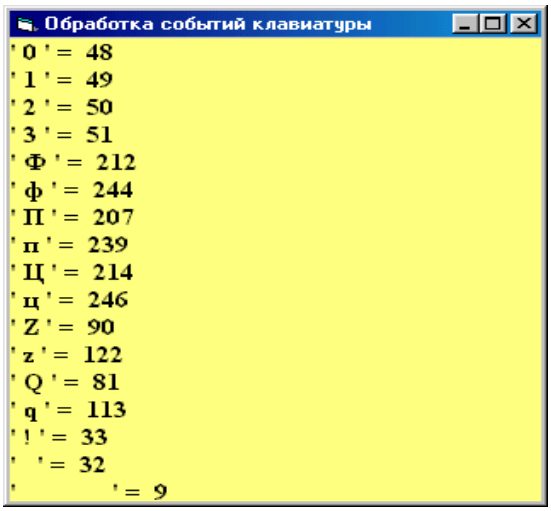


Рисунок 23.1 - Приложение для выполнения программы

Можно использовать константы Visual Basic для кодов клавиш, например:
код клавиши <Tab> = 9, имя соответствующей константы - *vbKeyTab;*
<BackSpace> = 8, константа - *vbKeyBack*

События KeyDown и KeyUp.

События *KeyDown* и *KeyUp* сообщают точное физическое состояние самой клавиатуры: клави-ша нажата (*KeyDown),* и клавиша отпущена (*KeyUp*).

Синтаксис обеих процедур одинаков:

```
Private Sub Form_KeyUp(KeyCode As Integer, _Shift As Integer)
End Sub
```

События KeyDown и KeyUp.

- параметр *KeyCode* содержит код нажатой клавиши. Причем <A> и <a> являются одной и той же клавишей, т. е. имеют одинаковое значение параметра *KeyCode,* соответствующее коду <A>. «1» на основной клавиатуре и «1» на дополнительной цифровой клавиатуре имеют разные значения, хотя генерируют один и тот же символ;
- параметр *Shift* указывает состояние клавиш <Shift> , <Ctrl> и <Alt>, только анализируя этот параметр, можно определить, в каком регистре была нажата клавиша.

В следующем примере используется событие *KeyDown*, для определения, была ли нажата клавиша <Z> или <z>:

```
Private Sub Form_KeyUp(KeyCode As Integer,_Shift As Integer)
 If KeyCode = vbKeyZ Then
   If Shift = 1 Then
      MsgBox "Вы нажали прописной символ <Z>."
   Else
      MsgBox "Вы нажали строчный символ <z>."
```

```
    End If
  End If
End Sub
```

Список литературы

1 Алиев В.К. Информатика в задачах, примерах, алгоритмах. – М.: СОЛОН-Р, 2001.

2 Ламонина Л. В., Познахирева Н. И., Шацкова Е.А. Практикум по алгоритмизации и программированию.- Омск, 2008.

3 Токарева М.А. Алгоритмизация и программирование. - Оренбург. 2006.

4 Гарнаев А.Ю. Использование MS Excel и VBA в экономике и финансах. – СПб.: БХВ – Санкт-Петербург, 2000. – 336 с.

5 Информатика: Базовый курс/ С. В. Симонович и др. – СПб.: Питер, 2001. – 640 с.: ил.

6 Кузьменко В.Г. VBA 2003. – М.: Бином, 2004.

7 Муравский И.Г. «Программирование и решение задач в системе Turbo Basic», учебное пособие. - Омск, ОмГАУ, 1999.

8 Тимофеевская М. Изучаем программирование. – СПб.: Питер, 2002. – 384 с.: ил.

9 http://sergun-2.narod.ru/metod_vba.htm - Методическое пособие по программированию для Microsoft Excel на Visual Basic for Applications (VBA).

10 Омарова Г.А. Алгоритмизация и программирование. 2013.

11 Потопахин В.Современный самоучитель по алгоритмам. - М.: «ДМК Пресс», 2011.

12 Искусство алгоритмизации. Потопахин В. - М.: «ДМК Пресс», 2011.

13 Иванова Г.С. Технология программирования. - М.: «Кнорус». 2013.

Интернет ресурсы

14 http://inform-school.narod.ru/ - электронный учебник по алгоритмизации и программированию.

http://www.nesterova.ru/bibl/algorithm_lang/

15 http://www.nesterova.ru/bibl/algorithm_lang/ Л.В. Зайцева, Основы алгоритмизации и программирования на языках высокого уровня. http://www.tambov.fio.ru/vjpusk/vjp005/rabot/39/index2.htm

16 http://www.tambov.fio.ru/vjpusk/vjp005/rabot/39/index2.htm - Методический материал для изучения темы «Алгоритмизация и программирование».

Сводный план 2016 г., поз. 250

Марат Катубаевич Шимырбаев

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Конспект лекций
для студентов специальности
5В074600- Космическая техника и технологии

Редактор Л.Т. Сластихина
Специалист по стандартизации Н.К. Молдабекова

Подписано в печать_____ Формат 60х84 1/16
Тираж 25 экз. Бумага типографская №1
Объем 7,4 уч.-изд..л. Заказ Цена 3700 тенге

Копировально-множительное бюро
некоммерческого акционерного общества
«Алматинский университет энергетики и связи»
050013, г. Алматы, ул. Байтурсынова, 126