Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра экономической информатики

Авторы:
Е. Н. Живицкая, В. Н. Комличенко, И. Г. Орешко, С. А. Соколов, А. А. Пасовец, А. В. Лепеш, Ф. Н. Козелько

Лабораторный практикум по курсу

«Основы информатики и вычислительной техники»

для студентов экономических специальностей

В 2-х частях

Часть 2

Минск 2001

УДК 002.5+681.3 (075.8)
ББК 22.1 Я 73
Л 12

Е. Н. Живицкая, В. Н. Комличенко, И. Г. Орешко, С. А. Соколов, А. А. Пасовец, А. В. Лепеш, Ф. Н. Козелько:

Л 12 Лабораторный практикум по курсу «Основы информатики и вычислительной техники» для студентов экономических специальностей в 2-х частях. Часть 2. - Мн.: БГУИР, 2001 - 42 с.

ISBN 985-261-6 (ч.2)

В практикуме представлен курс из 9 лабораторных работ, даны краткие теоретические сведения, примеры и варианты заданий для лабораторных работ.

Часть 1. Методическое пособие и учебные материалы по курсу «Основы информатики и вычислительной техники». В 2-х ч. Ч 1. / Е. Н. Живицкая, В. Н. Комличенко, С. А. Соколов и др. - Мн.: БГУИР, 2001, с. 82: Ил. 12

УДК 002.5 (075.8)
ББК 32.81 Я73

ISBN 985-260-8 (общ.)
ISBN 985-261-6 (ч.2)

Содержание

Лабораторная работа №1.
Расчет интеграла

Цели: *1) научиться вычислять интеграл функции;*
2) осуществить сравнение методов расчета интеграла и их погрешности.

При решении задач, связанных с вычислением интеграла, нет возможности осуществлять преобразование функции в соответствии со специальными функциями и правилами, принятым в математике. На практике эта задача сводится к нахождению площади фигуры, образованной (ограниченной) данной функцией, осью координат, прямыми x=a и x=б, где a и б – крайние точки (см. рис.1.1).

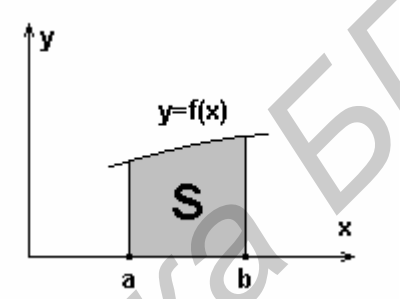


Рис.1.1. Интеграл функции f(x) от a до b.

При расчете площади данной фигуры применяют метод разбивки данной фигуры на множество элементов (прямоугольников или трапеций) с очень маленькой шириной, при этом можно предположить, что функция (верхняя часть этих элементов) представляет собой прямую рис.1.2. Подобную методику называют аппроксимацией.

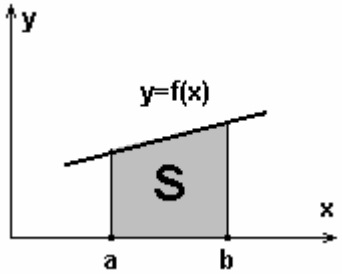


Рис.1.2. Аппроксимация функции f(x)

Выделяют два наиболее простых способа расчета интеграла: метод прямоугольника (метод Симпсона) и метод трапеции. В свою очередь метод Симпсона делится на три варианта: левого прямоугольника, правого и серединного.

Метод левого прямоугольника заключается в том, что площадь прямоугольника вычисляется на основании левой стороны прямоугольника и ширины прямоугольника - шага аппроксимации (рис.1.3.).

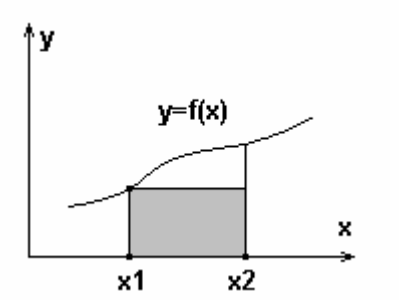


Рис.1.3. Метод левого прямоугольника.

Математически это можно записать следующим образом:

$$dS = f(x1)*dx, \qquad (1)$$

где dS – приращение площади – площадь маленького прямоугольника;
dx – приращение координат (x);
f(x1) – левая сторона прямоугольника.

Метод серединного прямоугольника заключается в том, что площадь прямоугольника вычисляется на основании значения функции в середине прямоугольника и ширины прямоугольника - шага аппроксимации (рис.1.4.)

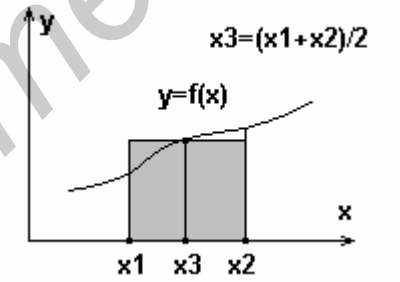


Рис.1.4. Метод серединного прямоугольника

Математически это можно записать как:

$$dS = f((x1+x2)/2)* dx, \qquad (2)$$

где dS – приращение площади – площадь маленького прямоугольника;
dx – приращения координат (x);
f(x1) – левая сторона прямоугольника;
f(x2) –правая сторона прямоугольника.

Метод правого прямоугольника заключается в том, что площадь прямоугольника вычисляется на основании правой стороны прямоугольника и ширины прямоугольника - шага аппроксимации (рис.1.5).

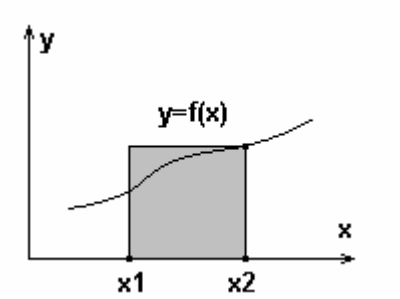


Рис.1.5. Метод правого прямоугольника

Математически это можно записать как:

$$dS = f(x2)* dx, \qquad 3$$

где dS – приращение площади;
dx – приращения координат (x);
f(x2) –правая сторона прямоугольника.

Метод трапеции заключается в том, что площадь трапеции вычисляется на основании произведения среднеарифметического значения левой и правой сторон прямоугольника и ширины прямоугольника - шага аппроксимации (рис.1.6.)

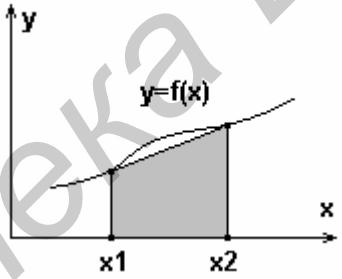


Рис.1.6. Метод правого прямоугольника.

Математически это можно записать как:

$$dS = (f(x1)+f(x2))/2* dx, \qquad (4)$$

где dS – приращение площади;
dx – приращение координат (x);
f(x1) – левая сторона прямоугольника;
f(x2) – правая сторона прямоугольника.

Задание. Рассчитать интеграл функции всеми четырьмя способами в соответствии с вариантом индивидуального задания. Найти среднеарифметическое значение интеграла на основе полученных значений. Найти максимальную погрешность методов относительно среднеарифметического значения.

Рекомендации:

1) использовать тип данных double;
2) шаг аппроксимации лучше всего выбирать 0.001 или 0.0001;
3) значения всех вариантов расчета интеграла поместить в массив.

Рассчитать интеграл функции на указанном интервале в соответствии с приведенными ниже вариантами заданий.

Варианты заданий:

1. f(x)=3sin(x2) на интервале от 0.1 до 0.3.
2. f(x)=cos(2x3) на интервале от 0.2 до 0.4.
3. f(x)=4tg (x2) на интервале от 0.1 до 0.2.
4. f(x)=2ctg(x2) на интервале от 0.1 до 0.3.
5. f(x)=ctg(2x)/2 на интервале от 0.1 до 0.2.
6. f(x)=tg(3x) на интервале от 0.05 до 0.1.
7. f(x)=ctg(sin x) на интервале от 0.15 до 0.2.
8. f(x)=2ctg(x/2) на интервале от 0.15 до 0.2.
9. f(x)=ctg(x3)/3 на интервале от 0.1 до 0.3.
10 f(x)=x sin(x2/2) на интервале от 0.3 до 0.4.
11 f(x)=x cos(x/3) на интервале от 0.2 до 0.3.
12 f(x)=x cos(x2) на интервале от 0.1 до 0.3.

Лабораторная работа №2.
Библиотечные функции. Работа с графикой.

Цель: *приобрести навыки работы с процедурами и функциями в графическом режиме.*

Графический режим реализуется с помощью функций библиотеки GRAPHICS.LIB, прототипы которых хранятся в заголовочном файле GRAPHICS.H, и используется для отображения на экране графической информации. При этом экран дисплея разделяется на прямоугольную сетку, состоящую из множества мельчайших элементов изображения, называемых пикселями. Каждый пиксель обладает свойствами светимости. Таким образом, любое изображение может быть синтезировано из множества отдельных точек (пикселей).

Реализация графического режима в ПЭВМ обеспечивается благодаря наличию специальной схемы, называемой графическим адаптером. ПЭВМ может комплектоваться следующими типами графических адаптеров: CGA, MCGA, EGA, VGA и др.

Работу графического адаптера поддерживает специальная программа, называемая драйвером. Загрузочный модуль драйвера хранится в специальном файле с расширением BGI (в файле CGA.BGI для адаптера CGA). Если в вашей программе используются какие-либо шрифты, то кроме BGI-файла необходимо наличие одного или более файлов шрифтов (CHR-файлов).

Загрузка драйвера осуществляется с помощью процедуры initgraph, имеющей следующий формат:

initgraph (int gd, int gm, string path),

где gd и gm являются параметрами-переменными, gd задает номер графического драйвера: 0 (или DETECT) - автоматический выбор драйвера; 1 - CGA драйвер. gm

задает номер графического режима, допустимого для заданного драйвера (передается процедуре initgraph только в случае, если gd не равен 0). path задает путь к каталогу, в котором находится графический драйвер (BGI-файл).

Если gd равен DETECT или 0, то процедура initgraph автоматически проверяет наличие графической аппаратуры и загружает соответствующий графический драйвер, находящийся в директории path. Если при этом имеется выбор из нескольких типов драйверов (или их режимов), то загружается тот из них, который обеспечивает более высокое качество изображения. Если path = “”, то подразумевается, что файлы драйверов содержатся в текущем каталоге.

В случае, если GraphDriver не равен 0, то значение этой переменной рассматривается как номер драйвера, для которого необходимо определить соответствующий графический режим gm. Режим задается с помощью константы или эквивалентного ей стандартного идентификатора. Каждый графический режим в свою очередь характеризуется двумя параметрами: разрешающей способностью экрана и количеством одновременно отображаемых цветов.

Если в процессе инициализации графического режима произошла ошибка или графическая аппаратура не распознана, то на экран выводится сообщение об ошибке и программа прекращает работу.

Для выхода из графического режима используется процедура closegraph(void), которая восстанавливает режим, существовавший до инициализации графики.

Рассмотрим пример установки графического режима:

```
#include <graphics.h>
void main(void) {
int gd = DETECT, gm;
   initgraph(&gd, &gm, "c:\\bc31\\bgi\\");
   ……….……
   closegraph();
}
```

Основные графические функции и процедуры.

Для работы в графическом режиме используются свыше 50 стандартных процедур и функций. Рассмотрим основные из них:

getgraphmode(void); - возвращает код текущего графического режима;
setgrapmode(номер режима); - устанавливает новый графический режим;
getx(void); - возвращает текущую X-координату;
gety(void); - возвращает текущую Y-координату;
getmaxx(void); - возвращает максимальную координату X;
getmaxy(void); - возвращает максимальную координату Y;

putpixel(int X, int Y, int Color); - процедура, которая выдает на экран точку с координатами X, Y и цветом Color;

getpixel(int X, int Y); - функция, которая возвращает цвет точки с координатами X, Y;

cleardevice(void); - процедура, которая очищает экран и устанавливает те значения всех графических параметров, которые предусмотрены по умолчанию;

clearviewport(void); - процедура, очищающая текущее окно;

lineto(int X, int Y); - процедура, которая проводит прямую линию из точки, где находится текущий указатель, в точку с координатами (X, Y) текущим цветом. Текущий указатель перемещается в точку (X, Y);

linerel(int dX, int dY); - процедура, которая проводит прямую линию из точки, где находится текущий указатель, в точку с приращением координат на dx (по X) и на dy (no Y) текущим цветом. Текущий указатель перемещается в конец линии;

line(int Xl, int Yl, int X2, int Y2); - процедура, которая проводит прямую линию из точки с координатами (XI, Y1) в точку (X2, Y2) текущим цветом. Положение текущего указателя не изменяется;

moveto(int X, int Y); - процедура, которая перемещает текущий указатель в точку (X, Y);

setcolor(номер цвета); - процедура, которая устанавливает цвет выводимого изображения*;*

getmaxcolor(void); - функция, которая возвращает максимальное значение цвета (константы цвета) для установленного графического режима;

setbkcolor(номер цвета); - устанавливает цвет фона;

getbkcolor(void); - возвращает значение текущего цвета фона;

setpalette(старый номер, новый номер); - процедура, которая изменяет в текущей палитре один цвет на другой по их номерам;

rectangle(int Xl, int Yl, int X2, int Y2); - процедура, которая рисует прямоугольник с координатами (XI, Y1) - верхний левый угол и (X2, Y2) - нижний правый угол;

bar(int Xl, int Yl, int X2, int Y2); - процедура, которая рисует закрашенный прямоугольник с координатами (XI, Y1) - верхний левый угол и (X2, Y2) - нижний правый угол, используя стандартный цвет закраски;

circle(int X, int Y; радиус); - процедура, которая рисует окружность с центром (X, Y) и радиусом Radius;

arc(int X, int Y, int StAngle, int EndAngle, int Radius); - рисует дугу окружности от угла StAngle до EndAngle с центром в точке (X, Y) и радиусом Radius;

ellipse(int X, int Y, int StAngle, int EndAngle, int XRadius, int YRadius); - процедура, которая рисует дугу эллипса с центром (X, Y) и с радиусами XRadius (по оси X), YRadius (по оси Y) от начального угла StAngle до конечного угла EndAngle. Значения StAngle=0 и EndAngle=360 приведут к вычерчиванию полного эллипса. Углы задаются в градусах по направлению против часовой стрелки;

outtext(char far *Text); - процедура, которая выдает на экран строку Text, начиная с текущей позиции указателя. Текущий указатель перемещается в конец строки;

outtextXY(int X, int Y, char far *Text); - процедура, которая выдает на экран строку Text, начиная с точки (X, Y). Положение текущего указателя не изменяется;

imagesize(int Xl, int Yl, int X2, int Y2); - функция, возвращающая размер необходимой памяти (в байтах) для сохранения изображения прямоугольного участка экрана с координатами (XI, Y1) - левый верхний угол и (X2, Y2) - нижний правый угол;

getimage(int Xl, int Yl, int X2, int Y2, void far *buffer); - процедура, которая запоминает изображение прямоугольного участка экрана с координатами (XI, Y1) и (X2, Y2) в буфере, на который указывает buffer;

putimage(int X, int Y, void far *buffer, операция); - процедура, которая выдает на экран изображение, сохраненное в буфере памяти, на который указывает buffer; (X, Y) - верхний левый угол прямоугольной области, куда выдается изображение; операция - режим наложе-

ния на существующее изображение при выводе (0 - стирание; 3 - наложение; 4 - инвертирование).

Примечание: Углы во всех процедурах задаются в градусах в направлении против часовой стрелки: 3 часа соответствуют 0 градусам, 12 часов - 90 градусам и т.д.

Рассмотрим несколько примеров использования основных процедур и функций графической библиотеки:

Пример 1

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void) {

    int gdriver = DETECT, gmode, errorcode;
    initgraph(&gdriver, &gmode, "c:\\bc31\\bgi\\");

    setbkcolor(3);
    setlinestyle(0,0,3);

    setcolor(5);
    int midx = getmaxx() / 2;
    int midy = getmaxy() / 2;
    int radius = 100;
    circle(midx, midy, radius);     // рисуется окружность в центре экрана

    setcolor(4);
    int xmax = getmaxx();
    int ymax = getmaxy();
    line(0, 0, xmax, ymax);         /* рисуется линия из левого верхнего угла экрана в
                                       правый нижний */
    setcolor(1);
    int stangle = 0, endangle = 360;
    int xradius = 100, yradius = 50;
    ellipse(midx, midy, stangle, endangle, xradius, yradius);      // рисуется эллипс

    setcolor(2);
    bar(midx-50, midy-50, midx+50, midy+50); // рисуется закрашенный прямоугольник

    setcolor(6);
    outtextxy(midx, midy, "Text");          // выводится текст

    setcolor(5);
    int left = getmaxx() / 2 - 150;
    int top = getmaxy() / 2 - 150;
    int right = getmaxx() / 2 + 150;
    int bottom = getmaxy() / 2 + 150;
    rectangle(left,top,right,bottom);       // рисуется незакрашенный прямоугольник

    getch();
```

```
    closegraph();
    return 0;
}
```

Пример 2. Составим программу для сохранения графического изображения и выдачи его на экран в инвертированном виде:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void) {
    int size;
    void *buffer;
int gdriver = DETECT, gmode, errorcode;
initgraph(&gdriver, &gmode, "c:\\bc31\\bgi\\");

    setbkcolor(3);
    setcolor(getmaxcolor());
    bar(10,10,100,100);                  // рисуется закрашенный прямоугольник
    size = imagesize(10,10,100,100);     /* определяется объем памяти, необходимый для
                                            сохранения изображения */
    buffer = malloc(size);               /* выделяется память в динамически
                                            распределяемой области размером size */
    getimage(10,10,100,100,buffer);      // запоминается изображение
    putimage(50,50,buffer,NOT_PUT);      // изображение выводится в инвертированном виде

getch();
    closegraph();
    return 0;
}
```

Задание. Разработать программу в соответствии с вариантом задания.

Варианты заданий:

1) вывода на экран графика функции с отображением осей координат, координатной сетки и делений по значениям аргумента и функции (программу оформить в виде процедуры, в которую значения функции передаются в виде массива)

2) которая рисует на экране циферблат механических часов, отображая перемещение секундной, минутной и часовой стрелок

3) отображающую процесс перемещения изображения человечка по экрану в горизонтальном и вертикальном направлениях при нажатии соответствующих клавиш

4) отображающую процесс вращения прямоугольника вокруг любой его внутренней точки

5) вывода на экран изображения домика (при нажатии соответствующих клавиш свет в окошке должен загораться желтым светом и гаснуть)

6) преобразования фиксированного графического изображения из позитивной формы (черным цветом по белому) в негативную и наоборот

7) отображающую процесс перемещения изображения зайца по экрану в горизонтальном и вертикальном направлениях при нажатии соответствующих клавиш

8) имитирующую процесс перемещения упругого шарика в замкнутом пространстве прямоугольной формы (начальное направление и скорость движения шарика должны задаваться произвольно)

9) отображающую процесс перемещения изображения цветка по экрану в горизонтальном и вертикальном направлениях при нажатии соответствующих клавиш

10) вывода на экран изображения домика, из трубы которого идет дым

11) отображающую на экране в графическом режиме изображение введенной строки символов с произвольным коэффициентом увеличения, который может изменяться при нажатии клавиш управления курсором

12) вывода на экран кипящего чайника, из носика которого идет пар

13) вывода на экран изображения девочки, машущей флажком при нажатии соответствующих клавиш

Лабораторная работа №3.
Классы хранения и видимость переменных
(правила областей действия).

Цели:

1) *изучить сходства и различия понятий «класс хранения», «область видимости», «время жизни»;*
2) *научиться правильно применять четыре основных класса переменных: внешние, автоматические, статические и регистровые.*
3) *Уметь объяснить примеры 1 – 4.*

Компилятор Си для установления корректной связи идентификаторов с объектами в памяти требует, чтобы для каждого идентификатора были заданы как минимум два атрибута: тип и класс хранения. Тип определяет размер памяти, выделяемой компилятором для объекта, и способ интерпретации выделенной памяти (например, как адрес памяти для указателей или массив однотипных элементов). Класс хранения определяет место, где объект располагается (внутренние регистры процессора, сегмент данных, сегмент стека), и одновременно время жизни объекта (например, все время выполнения программы или только время выполнения конкретной части кода программы). Класс хранения либо задается явно, либо компилятор «сам» определяет это по местоположению описания в тексте программы.

В языке Си различают четыре основных класса переменных: внешние, автоматические, статические и регистровые.

Для описания классов хранения используются ключевые слова: extern, auto, static и register.

Время жизни и класс хранения тесно связаны друг с другом. С точки зрения времени жизни различают три типа объектов: статические, локальные, динамические.

Тесно связанным с понятием «класс хранения» является и понятие видимости, или области определения идентификатора. Область определения – это та часть программы, в пределах которой идентификатор может использоваться для доступа. Есть несколько типов области определения: в пределах блока (локальная видимость), в пределах функции; в пределах файла текста программы.

Видимость – это область исходного кода программы, из пределов которой возможен корректный доступ к памяти или функции с использованием идентификатора. Видимость больше касается не компилятора, а компоновщика. Как правило, видимость и область определения идентификатора совпадают, но возможны и другие ситуации. Можно сказать, что область определения идентификатора задаёт его видимость по умолчанию.

Внешние переменные определены вне любой из функций и доступны для любой из них. Сами функции всегда внешние, так как язык не позволяет объявлять функции внутри других функций. Область действия внешней переменной простирается от точки во входном файле, где она объявлена, до конца файла. Если на внешнюю переменную нужно ссылаться явно, используется объявление идентификатора с ключевым словом extern. Обратите внимание на то, что воздействие на видимость объекта не связано с резервированием памяти компилятором. Поэтому, например, строка программы

```
extern int var1;
```

является не описанием переменной, а объявлением ее как внешней ссылки. Обрабатывая такое предложение, компилятор помечает для себя, что переменная var1 должна быть описана как внешняя. Отсюда вытекают два очевидных следствия:

1) при объявлении переменной как extern просто невозможно выполнить её инициализацию. Например, выражение

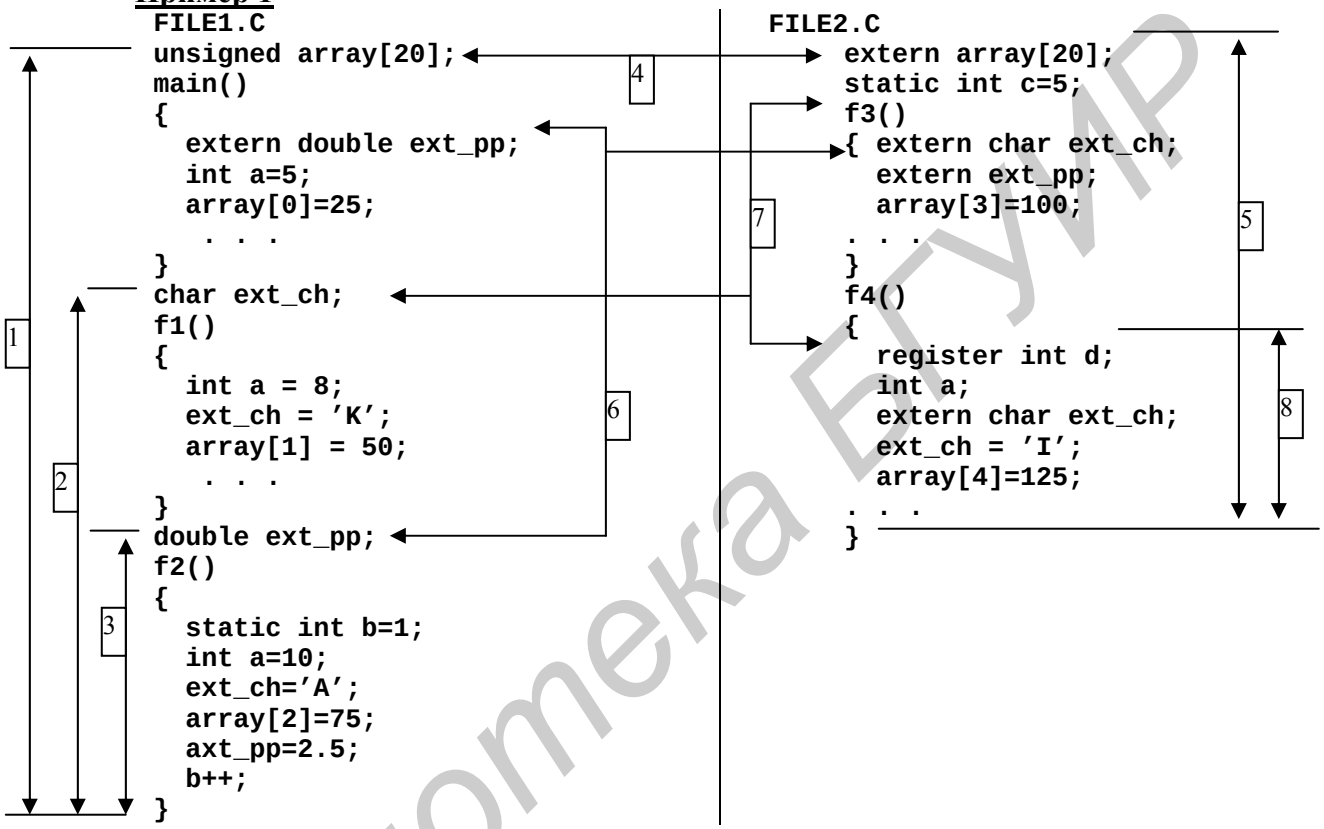
```
extern int var1=20;
```

является ошибкой;

2) для того чтобы какой-то объект данных, в том числе и объявленный как extern, был видимым из всех функций файла, необходимо поместить описание или объявление объекта в самом начале файла, перед первой функцией.

Внешние переменные – один из способов передачи функциям аргументов и возврата из функций необходимых значений. Просто взаимодействующие функции используют общие области памяти.

Пример 1



Линии 1-3, 5 указывают область действия глобальных или внешних переменных: массива array, переменных ext_ch, ext_pp. В пределах области определения их можно использовать в функциях без дополнительного объявления как extern. Однако для того чтобы использовать переменную ext_pp в функции main (), там необходимо объявить имя ext_pp как внешнюю переменную (линия 6).

Во всех функциях, за исключением f3(), описывается внутренняя переменная **a**. Так как для каждой функции **a** имеет область определения, ограниченную пределами функции, в каждой из них переменной **a** соответствует своя ячейка памяти, доступ к которой возможен только из пределов функции, но не из других функций.

По умолчанию переменные, описанные внутри функции, являются автоматическими (auto).

Большинство переменных программы на Си – это переменные, относящиеся к классу хранения auto, или, как часто говорят, автоматические переменные.

Автоматические переменные – это всегда локальные переменные, но не наоборот. Автоматические переменные располагаются в стеке или во внутренних регистрах процессора. В этой связи время, в течение которого определено их значение (время существования), ограничено временем выполнения функции или блока, в котором эти переменные описаны. Автоматические переменные должны явно инициализироваться, иначе содержимое ячеек памяти, выделенное для них, не будет определено.

Частным случаем автоматических переменных являются переменные, описанные с указанием ключевого слова register. Оно говорит о том, что переменная будет интенсивно использоваться. Так могут описываться отдельные переменные целых типов и указатели, но это неприменимо для массивов, структур, объединений и переменных с плавающей точкой. Задание ключевого слова register – это указание компилятору выделить для хранения переменных не ячейки стека, а внутренние регистры процессора. Если свободных регистров нет, таким переменным будут отведены ячейки в стеке. Использование класса register позволяет повысить скорость выполнения программы. Однако того же результата можно добиться и задавая опции компилятору. Для регистровых переменных нельзя взять адрес: они могут быть только автоматическими либо формальными параметрами функции (допустимые типы только int и char).

Автоматические переменные по отношению к функциям являются внутренними. Они начинают действовать при входе в программу (функцию) и уничтожаются при выходе из неё (можно дополнительно использовать ключевое слово auto).

Переменная d (пример 1) f4() объявлена как регистровая. Так как регистровые переменные запоминаются в регистрах центрального процессора, доступ и работа с ними выполняются быстрее, чем в памяти. В случае отсутствия свободных регистров переменная становится простой автоматической переменной. В остальном регистровые переменные аналогичны автоматическим переменным. Область действия d – линия 8.

В противоположность автоматическим статические объекты существуют всё время, пока выполняется программа. Обычно таким объектам компилятор выделяет ячейки памяти в фиксированных сегментах данных в соответствии с используемой при компиляции моделью памяти. По умолчанию статическими объектами являются

строковые литералы и все глобальные переменные. Кроме того, любой объект данных может быть явно отнесён к классу памяти static. При этом объект может оставаться локальным. Другими словами, любая глобальная переменная имеет класс хранения static, но необязательно любая статическая переменная – это глобальная переменная.

По умолчанию все объекты, отнесённые к статической памяти, будут инициализироваться нулевыми байтами. Если при описании статической переменной ей присваивается начальное значение, то это значение «впечатывается» в ячейки памяти, выделенные для переменной. Затем при загрузке программы инициализирующие статистический объект байты переносятся в оперативную память компьютера.

Переменная b (см. пример 1) объявлена как внутренняя статическая. В f2() переменная a инициализируется каждый раз при вызове f2(), при этом b инициализируется только один раз при запуске программы и первом вызове f2(). Внутренние статические переменные имеют такую же область действия, как и автоматические (в данном случае это линия 3), но они не исчезают при завершении работы содержащей их функции. Их значения хранятся в памяти компьютера от одного вызова функции до другого. Статическая переменная **c** (см. пример 1) описана как «внешняя статическая». Разница между внешней переменной и внешней статической переменной заключается в области их действия. В отличие от обычной внешней переменной, которая может использоваться функциями в любом файле, внешняя статическая переменная может использоваться только функциями того же самого файла, причём после определения переменной, здесь переменная **c** доступна только функциям f3() и f4().

Пример 2

```
#include <stdio.h>
void example(int);
void main(void)
{ int count;
  for (count=9;count>=5;count-=2)
  example(count);
}
void example (int c)
{ int f=1;
  static int stat=1;
  printf("c=%d,f=%d, stat=%d\n",c,f,stat);
  stat++;
}
```

При выполнении программы будут получены следующие результаты:

c=9, f=1, stat=1
c=7, f=1, stat=2
c=5, f=1, stat=3.

Обратите внимание на то, что значение статической переменной stat после выхода из функции example() не «забывается», как это имеет место для переменной f. Переменная f – автоматическая переменная, которой при каждом вхождении в функцию вновь присваивается значение 1. Инициализация же статической переменной stat производится только однажды – при загрузке программы.

Статические объекты с локальной областью определения, имея статическое время жизни, невидимы из других функций и не могут этими функциями модифицироваться. Это дает возможность создавать функции-«менеджеры» ресурса: менеджеры очередей, менеджеры памяти и пр. Для доступа к ресурсу обязательно требуется вызов менеджера, так как управляемый ресурс невидим из других функций.

Область определения глобальных идентификаторов для переменных ограничена пределами одного файла (FILE1.C), и, следовательно, эти переменные невидимы для функций в другом файле (FILE2.C). Если попытаться описать внешнюю функцию в другом файле FILE2.C даже с совпадающим именем, например массив array, массивы в разных файлах будут соответствовать различным участкам оперативной памяти, т.е. физически будут разными массивами. Для того чтобы функции файла FILE2.C имели доступ к глобальным переменным, описанным в файле FILE1.C, используется объявление extern. Именно такие объявления (линии 4 и 7) делают array и ext_ch общими переменными для функций в разных файлах. Так, массив array объявлен как внешний в самом начале второго файла вне функций f3() и f4(), это объявление, а следовательно, и массив array, видимы во всех функциях файла FILE2.C.

Приведём пример программы, текст которой размещён в двух файлах (для связи функций программы используется внешняя переменная i).

Пример 3

FILE1.C

```
#include <stdio.h>
extern int I;
void main(void)
{ void next(void); /* прототип*/
  i++;
  printf("В main i=%d\n",i);
  next();
}
int i=3;/*описание переменной*/
void next(void)
{ void other(void); /* прототип*/
  i++;
  printf("В next i=%d\n",i);
  other();
}
```

FILE2.C

```
#include <stdio.h>
extern int I;
void other(void)
{ i++;
  printf("B other i=%d\n",i);
}
```

Результат:

```
B main i=4
B next i=5
B other=6
```

Например, функция f1() может устанавливать необходимое значение переменной **a** и вызывать функцию f2(). Так как **a** видима из f2(), то она может прочесть значение переменной, а затем изменить его. Эти изменения переменной будут видимы в функции f1(). Естественно, связь функций через общие переменные чревата опасностью неожиданной модификации общих ячеек памяти и поэтому требует осторожности. Достоинство метода – значительно более компактный и производительный программный код, так как отпадает необходимость в операциях по записи копий фактических аргументов в стек, сохранению и восстановлению некоторых регистров процессора.

И, наконец, объекты с динамическим временем жизни создаются и разрушаются специальными функциями динамического управления памятью при выполнении программы. Физически динамические объекты располагаются в специально зарезервированной области памяти, называемой «куча»(heap).

Пример 4

Функция получения псевдослучайных чисел. Алгоритм работы функции отталкивается от базового числа (в данном случае базовое число=1), которое используется для создания нового базового числа и т.д. То есть функция псевдослучайных чисел должна «помнить» базовое число, которое она использовала при последнем вызове (необходимо использование статической переменной).

```
main()
{ int count;
  for (count=1; count<=5; count++)
  printf("%d\n",rand());
}
rand()
{
 static int randx=1;
 randx=(randx*25173+13849)%65536;        // формула для получения
        // псевдослучайного числа
 return(randx);
      }
```

Результат:

```
-26514
-4449
20196
-20531
3882
```

ВАРИАНТЫ ЗАДАНИЙ:

1. Использовать варианты задания к лабораторной работе №6. ч 1. Модифицировать программу так, чтобы:
   - ввод, вывод, сортировка были оформлены как отдельные функции;
   - функция сортировки массива структур находилась в отдельном файле;
   - структурная переменная объявлялась как статическая
   - счетчики циклов объявлялись как регистровые переменные;
   - передача значений осуществлялась через внешние переменные.
2. Модифицировать пример 4 так, чтобы пользователь мог устанавливать своё базовое число или модифицировать пример 4, используя в качестве базовых чисел второй и четвёртый элементы массива из лаб.раб. 5.

   Уметь объяснить примеры 1 – 4.

Лабораторная работа №4
Динамические структуры данных. Стеки и очереди

Цели:

1) *получить знания по разработке программ с динамическими структурами данных;*
2) *научиться работать со стеками и очередями;*
3) *разработать программу со стеками и очередями.*

Создание динамических структур данных типа стек. Одним из наиболее распространенных видов списков является стек - это список с одной точкой доступа к его элементам (называемой вершиной стека). Расположение элементов в стеке показано на рис.4.1:

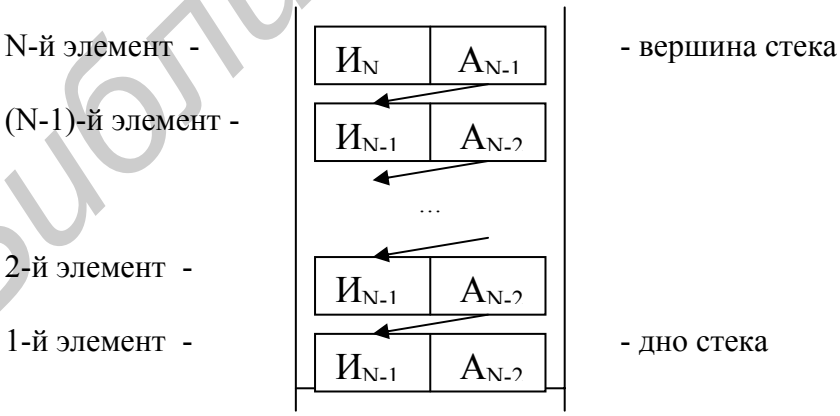


Рис.4.1. Расположение элементов в стеке

Описание отдельных элементов стека и его формирование в целом могут быть реализованы аналогично списку.

Пример 1

Составить программу формирования стека, заполняющегося путем ввода целых положительных чисел с клавиатуры. Как только будет введено первое отрицательное число, содержимое стека выводится на экран. Данная программа выглядит следующим образом:

```
#include <stdio.h>
struct Steck {
  int Info;
  struct Steck *Ref;
} *p1, *p2;

void main (void) {
int i=0;
  p2 = NULL;

while (i>=0) {
    p1 = new(struct Steck);
    scanf("%d", &i);
      p1->Info = i;
      p1->Ref = p2;
      p2 = p1;
}

while (p1!=NULL) {
    printf("Element: %d\n", p1->Info);
    p2 = p1->Ref;
    delete(p1);
    p1 = p2;
}
}
```

Создание динамических структур данных типа очередь. Другим наиболее распространенным видом списков является очередь - это список, в котором извлечение элементов происходит из начала цепочки, а запись новых элементов - в конец цепочки. Расположение элементов в очереди такое же, как в стеке (см. рис. 4.1), только дну стека здесь соответствует начало очереди, а вершине - конец очереди.

Элемент очереди описывается так же, как и элемент стека (при однонаправленной очереди):

```
struct tern {
  int info;
  struct tern *p;
} *tern_begin, *tern_end, *t;
```

Формирование очереди осуществляется с помощью следующей процедуры:

```
void Make_Tern (void) {
   int i=0;
```

```
  scanf("%d", &i);
    t = new(struct tern);
    t->p = NULL;
    t->info = i;
  tern_begin = t;
  tern_end = t;

 while (i>=0) {
  scanf("%d", &i);
    t = new(struct tern);
  tern_end->p=t;
    t->p = NULL;
    t->info = i;
  tern_end = t;
 }
```

Удаление элемента из начала очереди производится с помощью следующей процедуры:

```
void Delete (void){
 t = tern_begin;
 tern_begin = tern_begin->p;
 delete(t);
 printf("Element Deleted.");
}
```

Процедура добавления элемента в конец очереди имеет вид:

```
void Add (void) {
int i;
 printf("Add Element: ");
   scanf("%d", &i);
     t = new(struct tern);
   tern_end->p=t;
     t->p = NULL;
     t->info = i;
   tern_end = t;
 printf("New Element Added.");
}
```

С использованием вышеприведенных процедур основная программа будет иметь вид:

```
void main (void) {
 Make_Tern();
 Add();
 Delete();
   while (tern_begin != NULL) {
     printf("Element = %d \n",tern_begin->info);
     t = tern_begin->p;
     delete(tern_begin);
     tern_begin = t;
    }
}
```

Разработать программу в соответствии с перечисленными вариантами заданий.

Варианты заданий:

1) формирования очереди, содержащей целые числа, и упорядочивания по возрастанию лементов в этой очереди (в процессе упорядочивания элементы очереди перемещаться не должны);

2) формирования стека, куда помещаются целые положительные числа, вводимые с клавиатуры (процесс ввода должен прекращаться, как только среди вводимых чисел появляется отрицательное число, после этого программа должна вывести на экран содержимое стека в том же порядке, в котором они были введены);

3) вычисления значения многочлена $P(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$ в целочисленной точке x (при этом значения коэффициентов $a_0, a_1, ... a_n$, вводятся с клавиатуры и динамически размещаются в памяти в форме очереди);

4) формирования стека, содержащего вещественные числа, и упорядочивания по возрастанию элементов в этом стеке (в процессе упорядочивания элементы стека перемещаться не должны);

5) формирования стека, куда помещается последовательность символов в виде отдельных слов, вводимых с клавиатуры (каждое слово, помещенное в стек, следует вывести на экран терминала, при этом порядок вывода символов в каждом слове должен быть обратным по сравнению с последовательностью их ввода) ;

6) формирования стека, содержащего целые положительные числа, и его преобразования путем удаления из него всех четных чисел (в процессе преобразования стека его элементы в оперативной памяти перемещаться не должны) ;

7) формирования динамическую структуру данных для хранения генеалогического дерева (каждая вершина дерева должна содержать следующую информацию: имя и год рождения);

8) вычисления суммы элементов очереди, состоящего из вещественных чисел;

9) определяющую симметричность произвольного текста любой длины, который всегда должен оканчиваться точкой (эту задачу рекомендуется решать с помощью двух стеков: в первый стек следует поместить весь текст, затем во второй стек перенести его половину так, чтобы последний символ текста находился на дне стека, далее путем поэлементного сравнения этих стеков получить ответ на вопрос о симметричности текста);

10) формирования стека, куда помещается последовательность символов, вводимых с клавиатуры (процесс ввода символов должен прекращаться, как только среди вводимых символов появляется точка, после этого программа должна реверсировать стек; под реверсированием здесь понимается изменение направления ссылок в стеке на противоположное, т.е. после реверсирования вершина и дно стека меняются местами);

11) вычисления значения выражения следующего вида: $x_1 x_n + x_1 x_{n-1} + \ldots + x_n x_1$, при этом значения $x_1, x_2, \ldots x_n$ вводятся с клавиатуры и динамически размещаются в оперативной памяти либо в форме стека (или двух стеков, один из которых реверсирован по отношению к другому, - см. задание 10);

12) слияния двух стеков, содержащих возрастающую последовательность целых положительных чисел, в третий стек так, чтобы его элементы располагались также в порядке возрастания;

13) добавления к стеку, содержащему возрастающую последовательность целых положительных чисел, нового элемента так, чтобы порядок возрастания в стеке не изменялся (при добавлении нового элемента в стек другие его элементы перемещаться не должны);

14) формирования стека с последующим его преобразованием в двунаправленную очередь (двунаправленная очередь является динамической структурой данных, каждый элемент которой хранит не одну ссылку, а две - одна из них указывает на предыдущий элемент, другая - на следующий элемент очереди).

Лабораторная работа №5.
Списки

Цели:

1) *закрепить знания по разработке программ с динамическими переменными;*
2) *научиться работать со списками;*
3) *разработать программу со списком.*

Односвязный и двусвязный списки

Список - это совокупность объектов, называемых далее *элементами списка,* в которой каждый объект содержит информацию о местоположении связанного с ним объекта. Если список располагается в оперативной памяти, то, как правило, информация для поиска следующего объекта - это адрес (указатель) в памяти. Если связный список хранится в файле на диске, то информация о следующем элементе может включать смещение элемента от начала файла, положение указателя записи/считывания файла, ключ записи и любую другую информацию, позволяющую однозначно отыскать следующий элемент списка.

В простейшем виде элемент списка представляет собой структурную переменную, содержащую указатель (и) на следующий элемент, и любое число других полей, называемых далее *информационными.*

Существует множество разновидностей списков. Если движение от элемента к элементу списка возможно только в одном из направлений и список имеет начальную точку такого движения, говорят об *односвязном (однонаправленном) списке (рис.* 5.1).

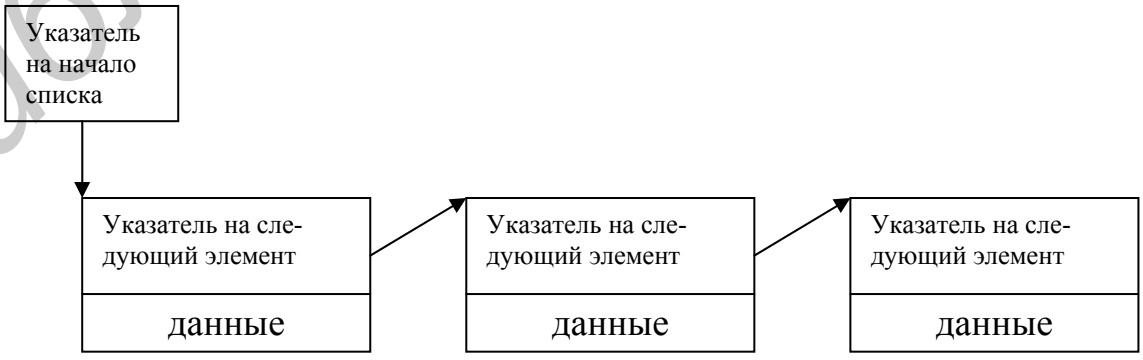


Рис.5.1.Односвязный список

Элемент односвязного списка включает только указатель на следующий элемент. Например, он может быть таким:

```
struct element{
        int value;    // объявление любой (или любых) переменной
        element * next;
};
```

Предполагается, что память, отводимая под элемент списка, выделяется динамически. Поэтому при реализации списка в памяти его длина ограничивается только доступным объемом памяти. Информационное поле представляет собой переменную целого типа value. Признаком последнего элемента списка является равенство NULL поля next. Реализация списка потребует описания указателя, хранящего начало списка:

```
element *start;
```

Частным случаем односвязного списка является кольцевой список, в котором последний элемент замыкается на первый (рис. 5.2).

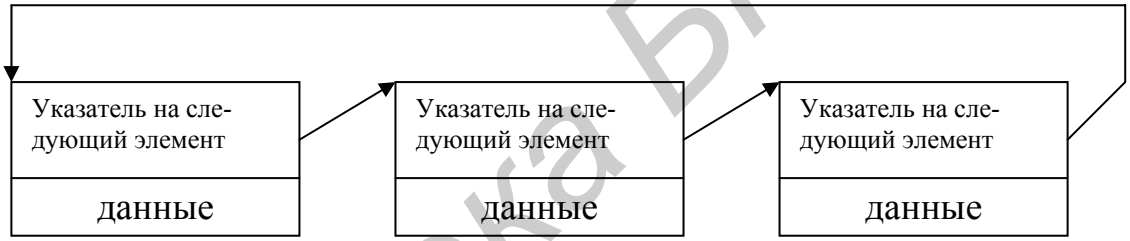


Рис.5.2. Кольцевой список

Альтернативой односвязному является *двусвязный (двунаправленный) список,* позволяющий выполнять "движение" от элемента к элементу в обоих направлениях. В этом случае элемент включает два указателя - на предыдущий (prev) и следующий (next) элементы списка. А так как список имеет и начало и конец, описываются два указателя - "головы" (head) и "хвоста" (tail) списка (рис.5.3).

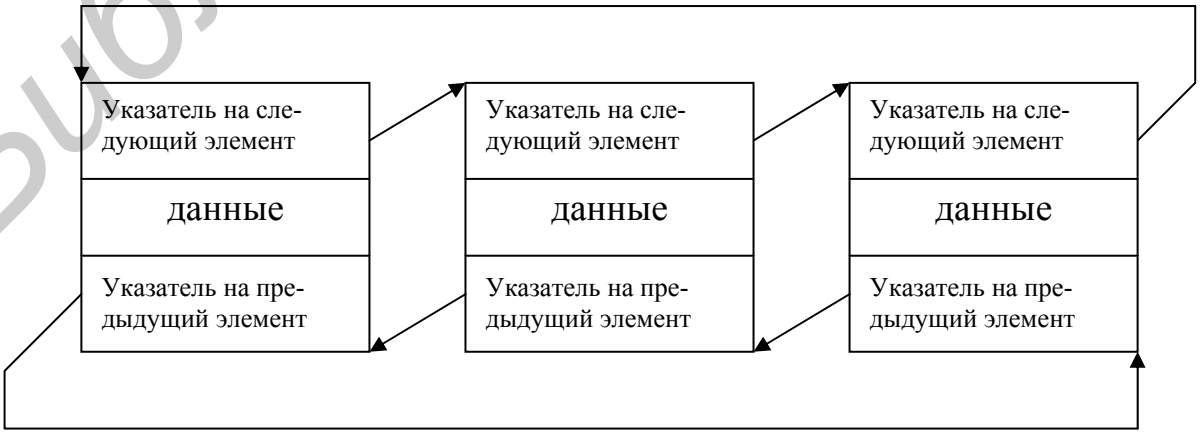


Рис.5.3.Двусвязный кольцевой список

Элемент двусвязного списка включает два указателя и информационные поля.

Например, он может быть таким:

```
struct element{
        element * prev;
        int value;
        element * next;
};
element *start;
element *end;
```

В качестве примера рассмотрим реализацию двухсвязного кольцевого списка в оперативной памяти. Элемент списка имеет структуру element. Информационные поля value целого типа будут заполняться с клавиатуры. После завершения ввода данных на экран выводится информация из всех элементов списка. Для упрощения программы элементы располагаются в статическом массиве. Пример программы с двухсвязным кольцевым списком:

```
#include <stdio.h>
void main (){

    // Объявление структуры
    struct element{
        element * prev;
        int value;
        element * next;
    };

    //  Объявление статического массива
    element mas[5];

    // Ввод данных
    for(int i=0;i<5;i++)
        scanf("%d",&mas[i].value);

    // Формирование ссылок для первого элемента списка
    mas[0].prev = &mas[4];
    mas[0].next = &mas[1];

    // Формирование ссылок для всех элементов списка
    // кроме первого и последнего
    for(i=1;i<4;i++){
        mas[i].prev = &mas[i-1];
        mas[i].next = &mas[i+1];
    }

    // Формирование ссылок для последнего элемента списка
    mas[4].prev = &mas[3];
    mas[4].next = &mas[0];

    // Определение стартового элемента
```

```
        element * start = &mas[1];

        // Определение последнего элемента списка
        element * end = &mas[4];

        // Локальная переменная – указатель на текущий элемент
        element * current = start;

        // Вывод значений поля данных value
        for(i=0;i<5;i++){
                printf("%d",current->value);
                current = current->next;
        }
}
```

Варианты заданий:

1) осуществить вывод списка в обратном направлении, используя prev;

2) объявить переменную типа element и вставить ее 2-й позицией в списке;

3) разработать программу с двухсвязным кольцевым списком, основываясь на задании к лабораторной работе №6 ч.1.

Лабораторная работа №6.
Проектирование структуры базы данных и создание таблиц

Цели: *получить навыки проектирования баз данных;*

изучить основные приемы создания реляционных таблиц средствами MS Access.

Задача. *Предположим, на некотором предприятии необходимо автоматизировать складской учет. Для этого прежде всего необходимо создать базу данных, в которой будут содержаться все необходимые сведения о товарах, заказах, поставщиках, сотрудниках, а также сведения о движении товаров, выполнении заказов и другие. На основании этих данных можно строить системы управления товарооборотом, контроля выполнения заказов, оптимизации транспортных потоков, автоматизированные системы складского учета и др.*

Общие сведения о MS Access

Работу с Access рассмотрим на примере создания базы данных для решения поставленной задачи.

Пример 1

Создать базу данных "Склад", содержащую сведения о товарах, поставщиках, заказах, а также сведения о предприятии.

Для того чтобы создать новую базу данных, после запуска СУБД Access в **Окне начального диалога** указать **Новая база данных** (рис. 6.1.) и подтвердить созда-

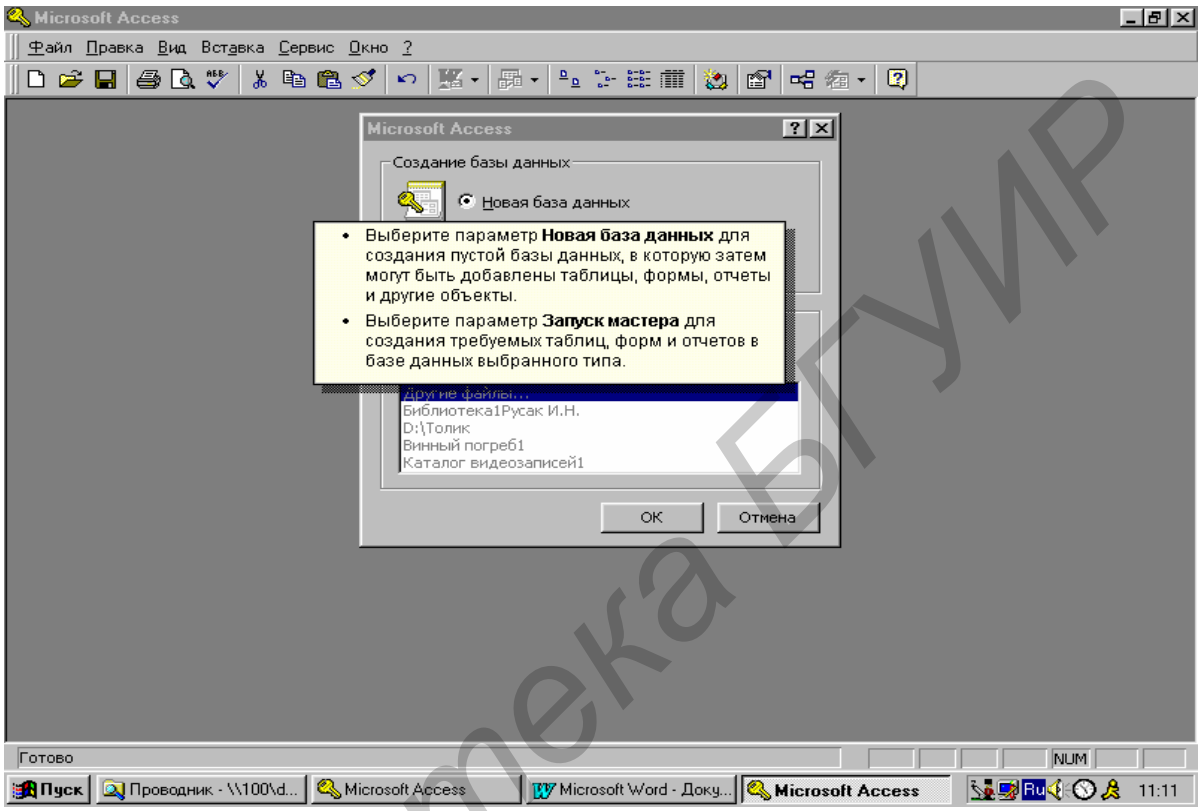


Рис. 6.1 Окно начального диалога в MS Access

ние новой базы данных нажатием кнопки **Ok**. В результате отобразится окно **Файл новой базы данных** (рис. 6.2), в котором предлагается указать путь и имя файла

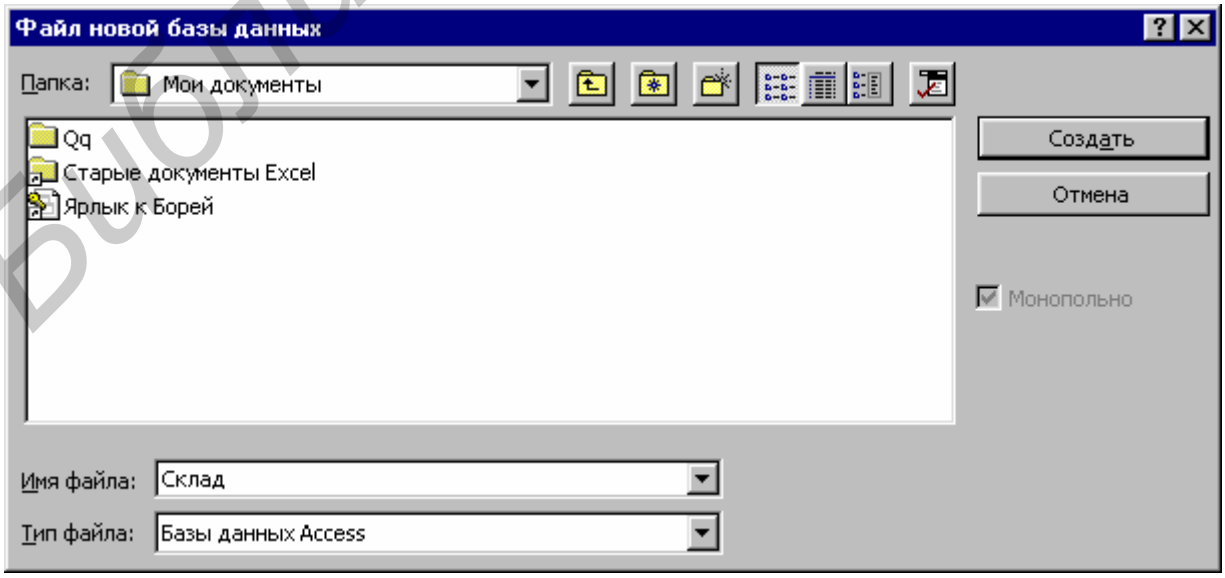


Рис. 6.2 Окно "Файл новой базы данных"

новой базы данных. По умолчанию файлу присваивается имя db1 и размещается он в папке "Мои документы", также принятой по умолчанию.

Переименуем файл, указав в поле ***Имя файла***, новое имя - "Склад". Затем остается подтвердить создание нового файла базы данных, нажав на кнопку **Создать**. В результате в указанной папке ("Мои документы") создан файл БД "Склад". При этом в окне Microsoft Access автоматически появляется окно новой базы данных - ***Склад: база данных*** (рис. 6.3).

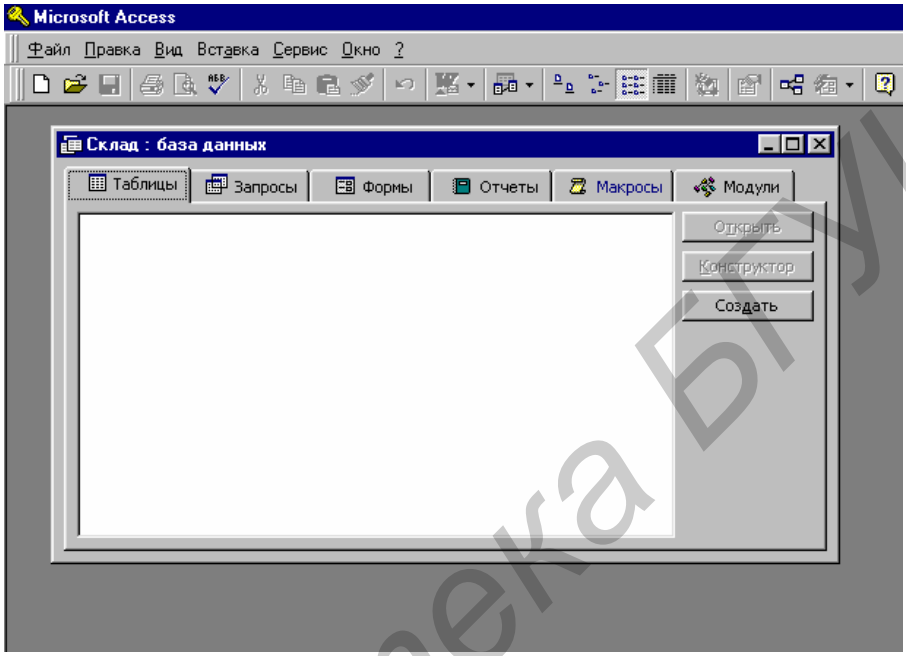


Рис. 6.3. Окно базы данных

Это окно предназначено для хранения, создания и редактирования различных объектов базы данных. В окне новой базы данных автоматически выбирается вкладка ***Таблицы***. Дальнейшие наши действия будут происходить на этой вкладке окна базы данных

Пример 2

Создать таблицу, содержащую сведения о товарах.

Сначала определим состав и типы полей будущей таблицы.

Предположим, наше торговое предприятие осуществляет продажу различных продуктов питания. Поэтому нас будут интересовать следующие характеристики объекта базы данных:

Код товара	- код, однозначно определяющий конкретный товар;
Марка	- марка товара (указанная на упаковке);
Описание товара	- описание, уточняющее состав данного товара;
Код типа	- код типа товара;
Серийный номер	- серийный номер, присвоенный партии товара;
Цена	- стоимость единицы товара;
Минимальный запас	- минимальный запас товара, хранящийся на складе;
Время задержки	- время задержки доставки товара.

Для создания новой таблицы воспользуемся кнопкой **Создать** в окне базы данных. В результате выдается окно ***Новая таблица*** (рис. 6.4), в котором требуется указать способ создания новой таблицы. Выберем ***Конструктор*** для создания структуры таблицы.

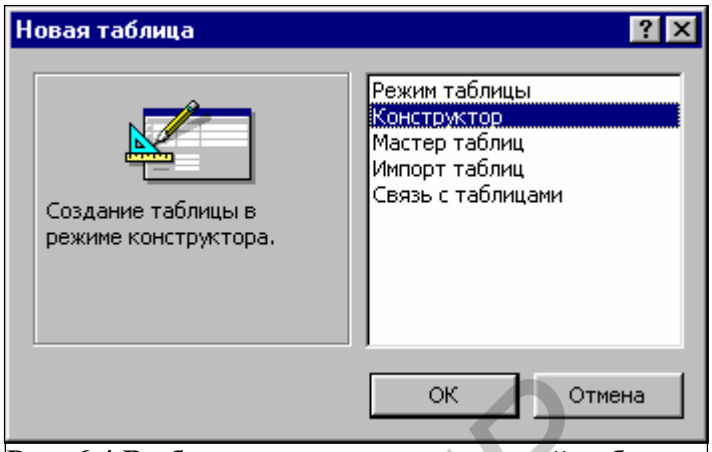


Рис. 6.4 Выбор режима создания новой таблицы

В режиме конструктора (рис. 6.5) опишем элементы структуры создаваемой таблицы - присвоим соответствующие имена полей, выберем их типы и, если необходимо, укажем их размеры. Имена полей в Access могут быть длиной до 64 символов и содержать как латинские буквы, так и символы кириллицы. Поэтому для имен полей можем использовать названия соответствующих характеристик товара. Тип поля будем выбирать в соответствии с информацией, которую предпо-

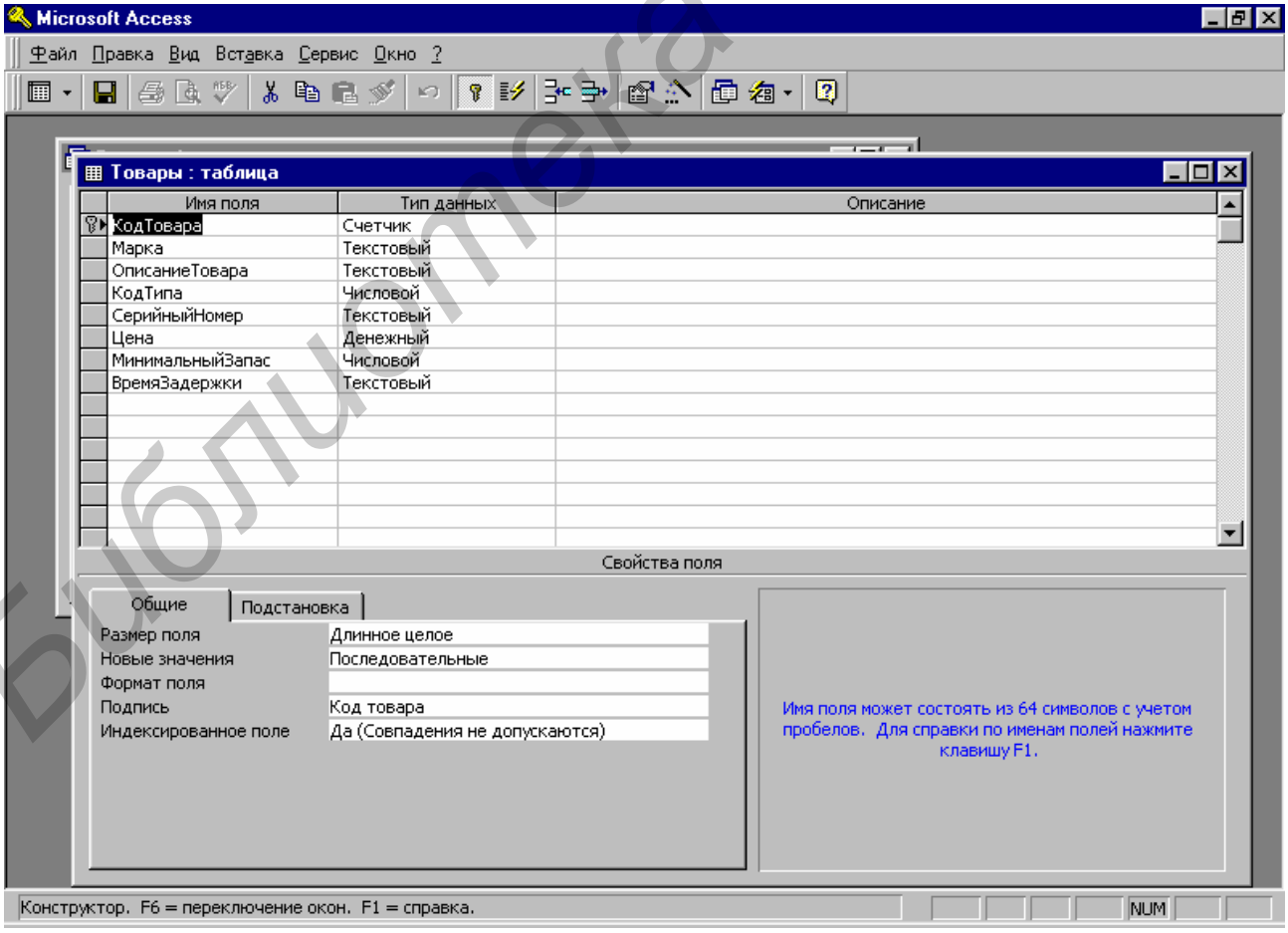


Рис. 6.5. Окно конструктора таблицы

лагается вносить в данное поле. Длину поля выберем, исходя из максимально возможной длины данных, которые могут быть введены в это поле.

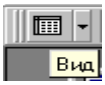
Заполнив таким образом поля структуры "Имя поля" и "Тип данных", получим следующую структуру таблицы "Товары":

Свойства

Число записей:	5	Дата изменения:	10.03.98 9:19:07
Дата создания:	10.03.98 9:19:03	Допускаются обновления:	Истина

Столбец

Имя	Тип	Размер
КодТовара	Числовой (длинное целое)	4
Марка	Текстовый	50
ОписаниеТовара	Текстовый	255
КодТипа	Числовой (длинное целое)	4
СерийныйНомер	Текстовый	50
Цена	Денежный	8
МинимальныйЗапас	Числовой (длинное целое)	4
ВремяЗадержки	Текстовый	30

После завершения работы с конструктором таблиц можно сразу же перейти к вводу информации в созданную таблицу. Для этого можно воспользоваться кнопкой ***Вид*** 

панели инструментов "***Конструктор таблиц***" либо меню ***Вид*** - ***Режим таблицы*** (рис. 6.6). Можно также закрыть окно Конструктора, сохранив макет таблицы, а затем нажать кнопку ***Открыть*** в **Окне базы данных** (рис. 6.3).

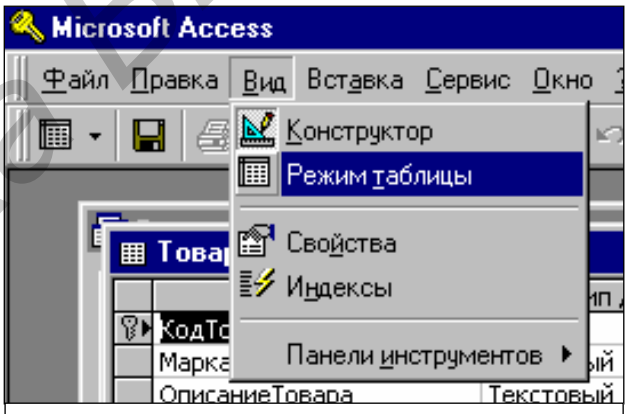


Рис. 6.6. Выбор режима таблицы из меню Вид

Результатом проделанной работы является таблица "Товары", содержащая следующие сведения:

Код товара	Марка	Описание	Код типа	Серийный	Цена	Мин. запас	Срок
1	Цейлонский		кондитерские		18р.	10	10 дней
2	Тибетское		напитки		19р.	25	10 дней
3	Анисовый си-		приправы		10р.	25	10 дней
4	Индейская	Индейская	приправы		22р.	0	10 дней
5	Смесь Антона	Смесь Гумбо	приправы		21р.	0	10 дней

Варианты заданий:

1) Создать базу данных "Склад", содержащую сведения о товарах, поставщиках, заказах, а также сведения о предприятии.
2) Создать таблицу "Поставщики" с помощью **Мастера таблиц**; таблица должна содержать следующие столбцы:

- КодПоставщика
- НазваниеПоставщика
- ДолжностьПолучателя
- Адрес
- Город
- Область
- Страна
- Факс
- УсловияОплаты
- Заметки

3) В режиме таблицы ввести сведения о товарах:

- КодТовара
- Наименование
- Цена
- Количество
- Стоимость
- СрокПоставки

4) В режиме конструктора создать таблицы "Заказы" и "Клиенты":

Структура таблицы "Заказы":

- КодЗаказа
- КодТовара
- КодПоставщика
- КодКлиента
- Наименование
- ДатаЗаказа.

Структура таблицы "Клиенты":

- КодКлиента
- Наименование
- ФИО_Руководителя
- Адрес
- Область
- Страна
- Телефон

5) Используя окно схемы данных, создать связи между таблицами.

6) Ввести данные в полученные таблицы. (не менее 10 записей)

Лабораторная работа №7.
Проектирование запросов для управления данными

Цели: *Получить навыки конструирования запросов различного типа для выборки данных из реляционных таблиц и управления данными;*

освоить технологию создания запросов QBE;

изучить основные конструкции реляционного языка запросов SQL.

Задача. *Пусть требуется получить определенные сведения из одной или нескольких таблиц базы данных. Для решения этой задачи целесообразно создать запросы на выборку, содержащие требуемые критерии отбора данных. В ряде случаев требуется удалять или добавлять записи в имеющиеся таблицы базы данных. Эту задачу решают при помощи запросов на удаление или добавление соответственно, в которых указываются необходимые критерии, указывающие, какие записи должны быть удалены или, наоборот, добавлены в таблицу.*

Общие сведения о запросах MS Access

В больших базах данных часто возникает проблема поиска необходимой информации (или отбора записей), удовлетворяющей определенным критериям. Задача поиска информации является одной из самых трудоемких и во многих случаях - одной из главных.

Для решения этой задачи предназначен механизм запросов. Этот механизм является стандартным и применяется почти одинаково во всех (или, во всяком случае, подавляющем большинстве) СУБД реляционного типа. Он представляет собой набор команд на языке SQL, определяющих критерий отбора записей в реляционной таблице. Таким образом, чтобы получить необходимую информацию из базы данных, следует записать соответствующие команды на языке SQL или, иначе говоря, - сформировать запрос.

Многие СУБД обладают механизмом автоматизации проектирования запросов. Чаще всего запрос формируется на специальном бланке. Такой метод формирования запроса называется QBE (**Query By Example** - **Запрос По Образцу**). В MS Access процесс создания запроса подобен процессу создания таблиц (см. Лабораторная работа №6. Проектирование структуры базы данных и создание таблиц.) Для того, чтобы начать проектирование нового запроса необходимо перейти на вкладку **Запросы** окна Базы Данных и нажать кнопку **Создать**. В результате появится окно диалога **Новый Запрос** (рис. 7.1), аналогичное окну **Новая таблица**. В этом окне будет предложено выбрать один из вариантов создания запроса.

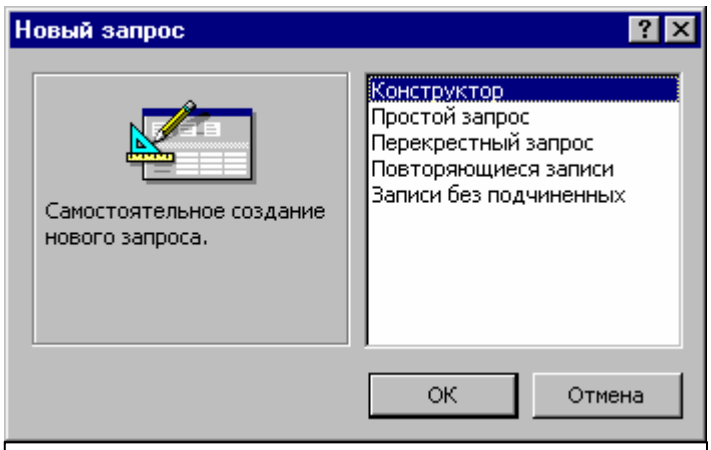


Рис. 7.1 Окно нового запроса

Запрос можно создать самостоятельно при помощи **Конструктора** или использовать готовый. Назначение каждого из режимов поясняется в левой части окна при указании мышью. Как показывает практика, большинство запросов создается с помощью **Конструктора**. По этой причине мы более подробно рассмотрим этот способ.

После подтверждения запуска ***Конструктора*** открывается бланк запроса (рис. 7.2) и окно ***Добавление таблицы*** (рис. 7.3). В этом окне пользователю предостав-

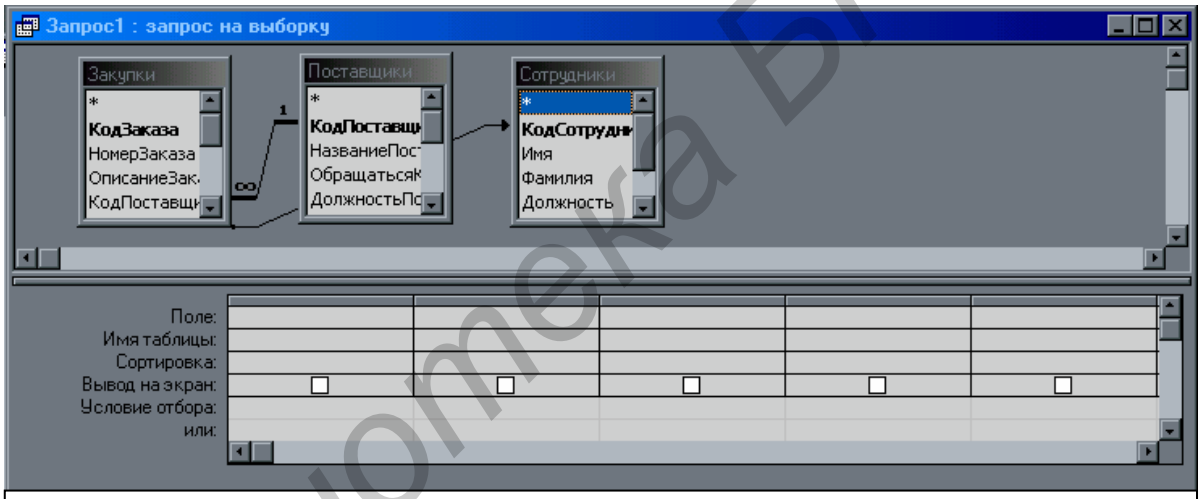


Рис. 7.2 Конструктор запроса

ляется возможность выделить одну или несколько таблиц, участвующих в запросе (выделение таблиц осуществляется аналогично выделению файлов в операционной системе Windows). Чтобы выделенные таблицы поместить в запрос, следует нажать кнопку ***Добавить***. Указанные таблицы отображаются в верхней части окна ***Конструктора Запроса*** вместе со всеми связями, если они имеются.

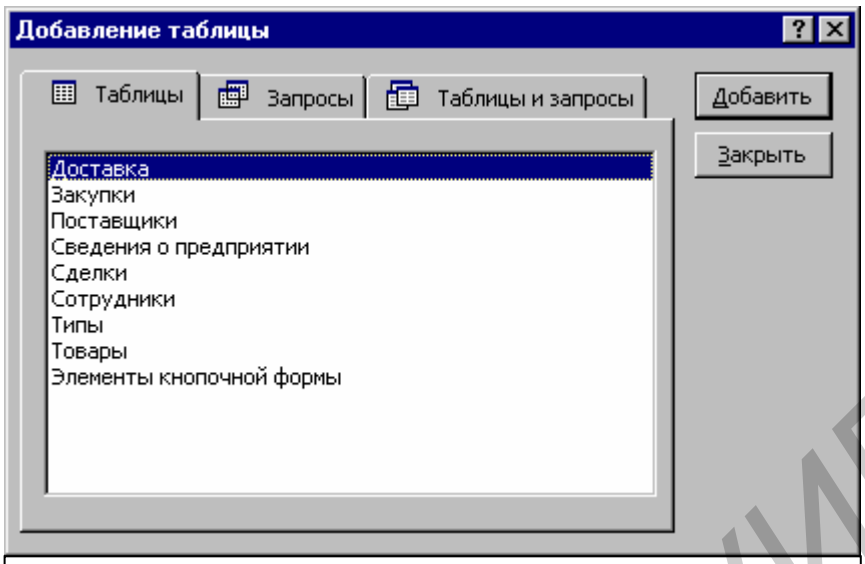


Рис. 7.3 Окно добавления таблиц

Теперь чтобы сформировать запрос, необходимо в бланке **Конструктора** сформировать образец. Он составляется из полей соответствующих таблиц, логических условий и выражений. В простейшем случае, если в бланк помещаются только некоторые поля из таблиц, мы получаем запрос на выборку определенных полей из одной или нескольких таблиц. Указать наименование поля, значения которого должны выводиться или участвовать в запросе, можно одним из следующих способов (рис. 7.4):

- Выбрать поле из списка ***Поле*** в окне ***Конструктора***.
- В таблице из верхней части окна **Конструктора** выделить необходимые поля и перетащить их мышью в ячейку **Поле** бланка запроса в нижней части окна - выбранные поля помещаются в бланке запроса последовательно, начиная с той ячейки, где была освобождена кнопка мыши.

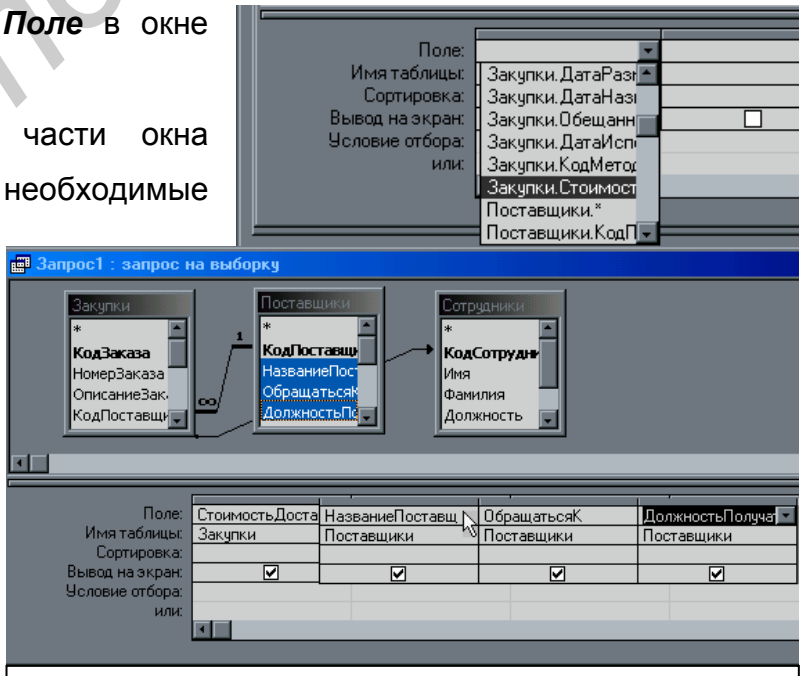


Рис. 7.4. Работа с бланком запроса в режиме конструктора

- Если в образец запроса необходимо поместить все поля из таблицы, то можно одним из вышеуказанных способов поместить в бланк символ * ("звездочка").

Варианты заданий

1. Создать запрос на выборку, позволяющий получить следующие сведения из базы данных:

таблица "Поставщики":

- НазваниеПоставщика
- Адрес
- Город
- Факс

таблица "Товары":

- Наименование
- Цена
- Количество
- Стоимость

таблица Клиенты":

- Наименование
- ФИО_Руководителя
- Адрес

Результирующие данные сортировать по наименованию товара.

2. Создать запрос на выборку для получения данных, удовлетворяющих определенным критериям:
   1) вывести всех поставщиков, обслуживающих клиентов в городах, названия которых начинаются на букву "М";
   2) получить сведения о клиентах, заказавших товары стоимостью от 200 до 400$;
   3) вывести адреса поставщиков, наименования которых начинаются с букв "Д … И", обслуживающих клиентов, заказавших товары стоимостью не менее 250$.;
   4) получить те записи из связанных таблиц (исключая коды), где содержатся сведения об определенном товаре. Записи сортировать по стоимости товара в порядке убывания;
   5) получить сведения о зарубежных клиентах (расположенных не в Беларуси), оформивших заказ на сумму, превышающую 300$;
   6) вывести записи о поставщиках, расположенных в Гомеле и Могилеве;
   7) получить данные о клиентах, которые проживают в Минске и Бресте или которые сделали заказ на поставку компьютеров;
   8) выдать информацию о клиентах, проживающих в городах, названия которых начинаются на букву "Д", или в Могилевской области, но не заказавших телевизоры;
   9) получить сведения о заказах, срок поставки которых истекает после 01/04/01

10) вывести информацию о поставщиках города Минска, которые должны выполнить заказы в период времени с 20/03/01 по 15/04/01;
11) получить сведения о клиентах, заказавших товары стоимостью от 200 до 400$, получающих заказы 12/04/01 или 20/05/01;
12) уточнить информацию о поставщиках, наименование которых точно неизвестно, возможно, "АЛВИГ", "АДВИГ" или "АМВИГ", срок поставки которых для клиентов, проживающих в городах Добруш и Светлогорск, истек 5 дней назад.

Лабораторная работа №8.
Подготовка презентации средствами MS PowerPoint

Цели: *изучить основные возможности системы MS PowerPoint, освоить основные принципы и этапы разработки презентации.*

Задача: *создать презентацию, представляющую предприятие и его производственную деятельность*

Краткое описание MS PowerPoint

Программа PowerPoint, которая входит в комплекс программ MS Office, предназначена для создания презентаций.

После запуска программы PowerPoint вы увидите окно, которое представлено на рис. 8.1.

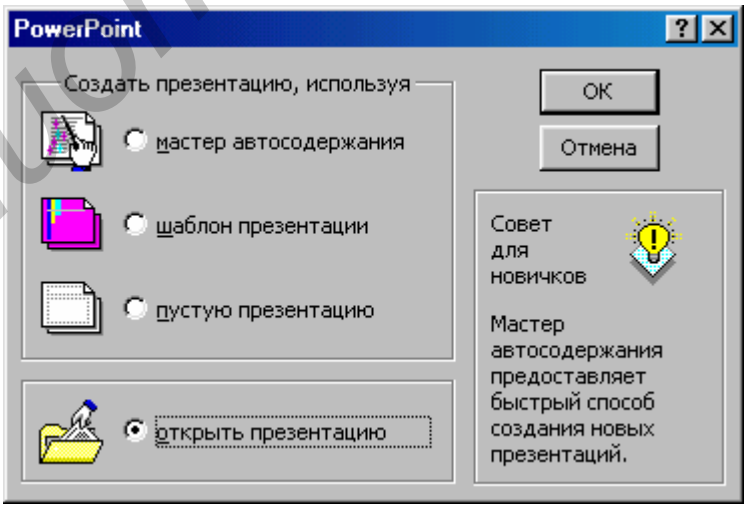


Рис. 8.1

Если хотите открыть презентацию, то необходимо выбрать пункт «Открыть презентацию».

Если хотите создать новую презентацию, то существуют 3 варианта. новую презентацию можно создать, используя:

1) мастер автосодержания;
2) шаблон презентации;
3) пустую презентацию.

«Мастер автосодержания»

Мастер автосодержания позволяет создать презентацию основываясь на некоторой стандартной структуре презентации.

Если выбран режим «Мастер автосодержания», то перед вами появиться окно, которое представлено на рис. 8.2.

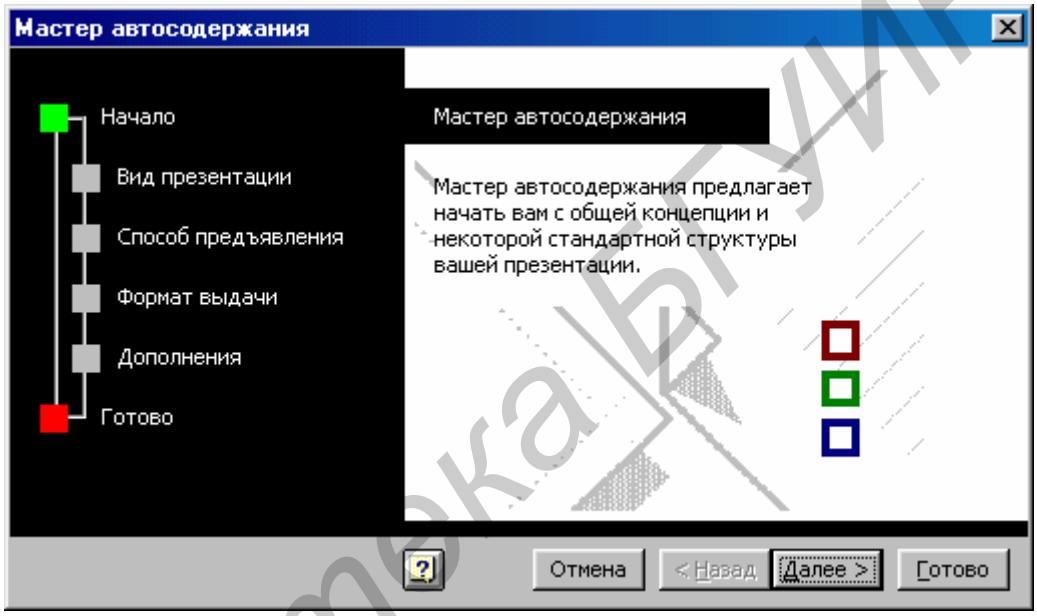


Рис. 8.2

Используя «Мастер автосодержания», можно выбрать вид презентации из уже существующих, способ предъявления, формат выдачи и дополнительные данные. После введения всех данных необходимо нажать кнопку «Готово». Затем программа создаст несколько связанных слайдов, которые необходимо будет откорректировать с помощью инструментов PowerPoint.

«Шаблон презентации».

«Шаблон презентации» позволяет выбрать оформление вашей будущей презентации.

Если выбрали «Шаблон презентации», то появится окно, которое представлено на рис. 8.3.

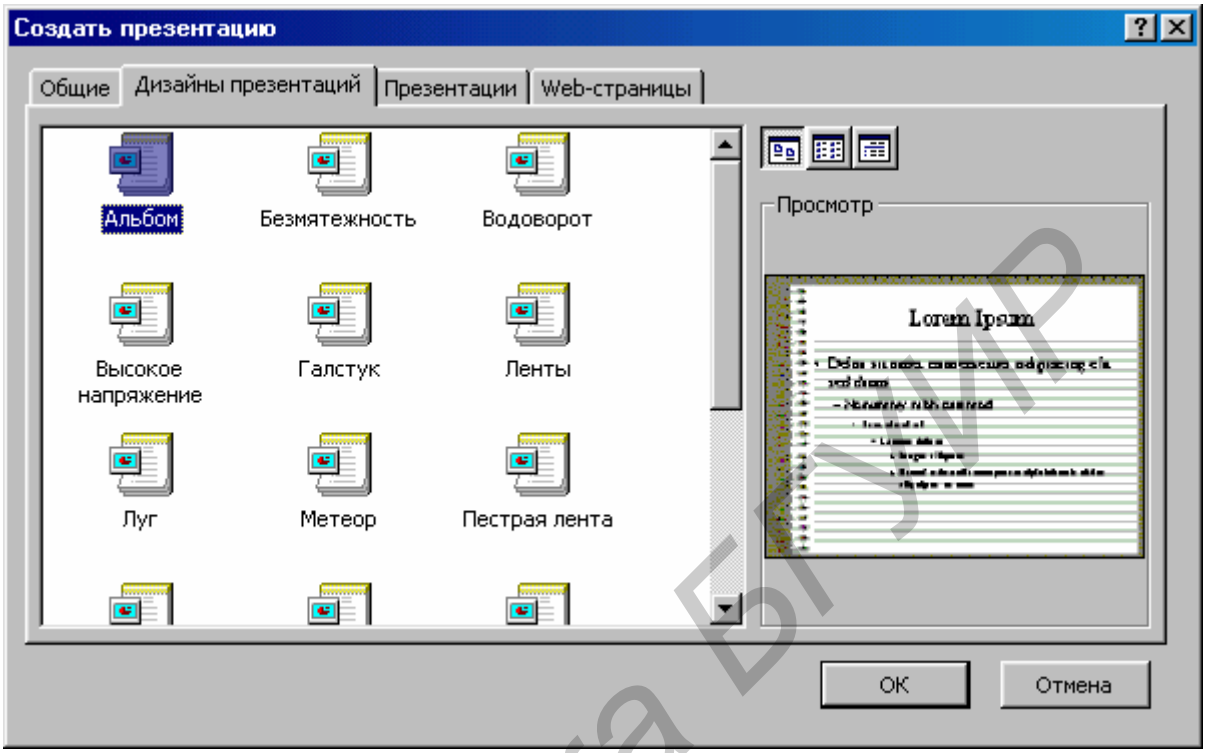


Рис. 8.3

«Шаблон презентации» позволяет выбрать только внешнее оформление презентации. После того, как вы выбрали внешнее оформление и нажали кнопку «ОК», перед вами появится окно, которое позволит выбрать авторазметку вашего первого слайда. Окно выбора авторазметки представлено на рис.8.4.

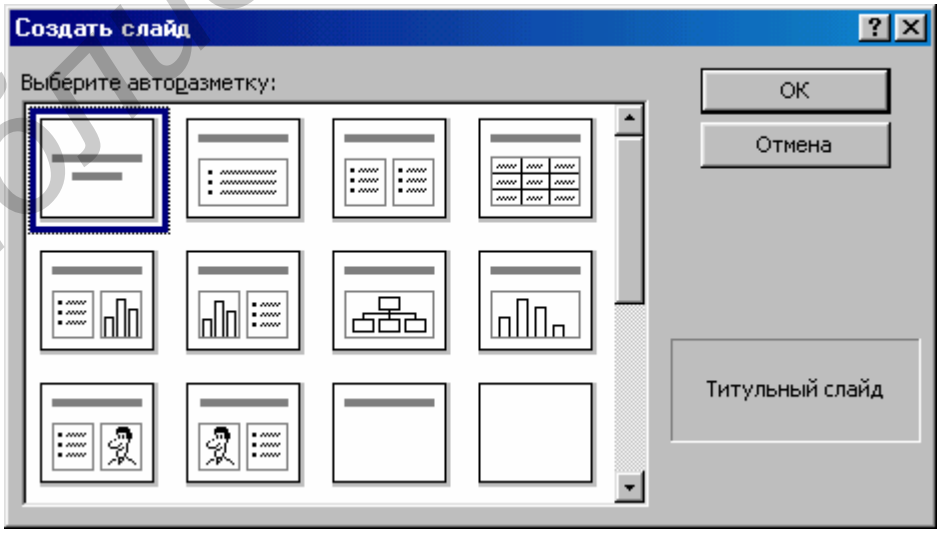


Рис. 8.4

Окно авторазметки позволяет выбрать из 24 вариантов один, который вас устраивает в данный момент.

«Пустая презентация».

Если вы выбрали пустую презентацию, то у вас появится окно, которое представлено на рис. 8.4. После выбора разметки слайда вы нажимаете кнопку «ОК». После нажатия кнопки происходит переход к непосредственному оформлению слайда.

«Создание второго и последующих слайдов».

Чтобы создать ещё один слайд, необходимо сделать следующие действия.

1. В меню «Команды» выбрать «Создать слайд». Если у вас на экране нет пункта «Команды», то вам необходимо выбрать пункт меню «Сервис», затем - «Настройка». Появится окно, которое представлено на рис. 8.5.

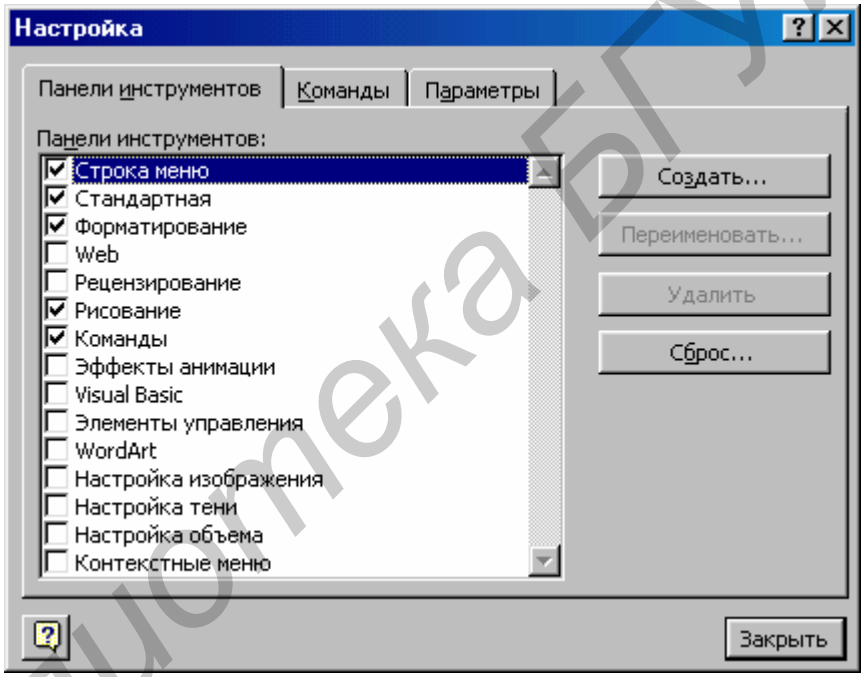


Рис. 8.5.

В данном окне необходимо, чтобы перед надписью «Команды» стояла «галочка», после чего следует нажать кнопку «Закрыть». После завершения указанных действий пользоваться меню «Команды», как было указано выше

«Настройка анимации».

Перед тем как настраивать анимацию, необходимо на слайде разместить хоть какие-либо объекты (текст, рисунок или другие объекты). Чтобы настроить анимацию, необходимо войти в «Показ слайдов», затем выбрать команду «Настройка анимации». После того как вы вошли в окно «Настройка анимации», у вас на экране появится окно, которое изображено на рис. 8.6.

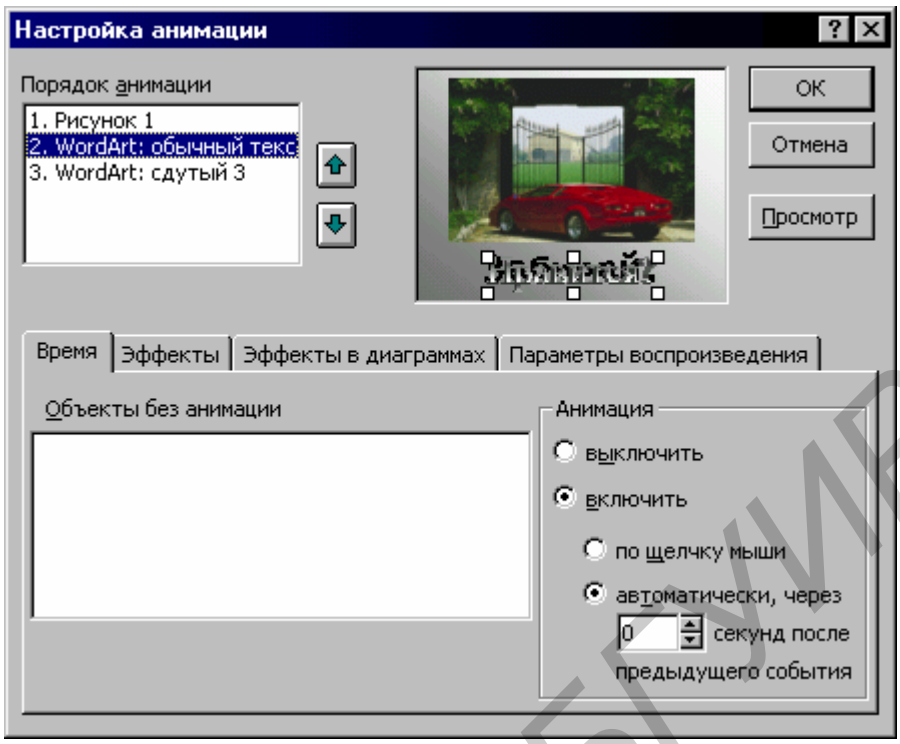


Рис. 8.6.

Чтобы включить анимацию необходимо выбрать элемент в окне «Время» в подокне «Объекты без анимации», в пункте «Анимация» выбрать селектор «Включить» и указать, как именно активизировать анимацию (по щелчку мыши или автоматически, через определённое время).

Разработать презентацию в соответствии с вариантом задания.

Варианты заданий:

1. Рекламное агентство. Презентация должна содержать не менее 7 слайдов и охватывать тематику агентства. Презентация должна быть единой и должна нести единую смысловую нагрузку и одинаковый стиль.
2. Предприятие производящее продукцию легкой промышленности. Презентация должна содержать не менее 7 слайдов и охватывать некий перечень продукции производимой предприятием. В презентации должны присутствовать графики, таблицы, диаграммы и графические объекты. Презентация должна иметь единую тематику и нести единую смысловую нагрузку.
3. Предприятие машиностроительного комплекса. Презентация должна содержать не менее 7 слайдов и охватывать некий перечень продукции производимой предприятием. В презентации должны присутствовать графики, таблицы, диаграммы и графические объекты. Презентация должна иметь единую тематику и нести единую смысловую нагрузку.

4. Предприятие радиоэлектронной промышленности. Презентация должна содержать не менее 7 слайдов и охватывать некий перечень продукции производимой предприятием. В презентации должны присутствовать графики, таблицы, диаграммы и графические объекты. Презентация должна иметь единую тематику и нести единую смысловую нагрузку.

Список литературы

1. Фигурнов В.Э. Программное обеспечение персональных ЭВМ. – М.: Наука, 1988.
2. Гукин Д. Word for Windows для начинающих: Пер. с англ. – Киев.: Диалектика, 1994.
3. Бемер С., Фратер Г. MS Access для пользователя: Пер. с нем. – Киев.: Торгово-издат. бюро BHV, 1994.
4. Николь Наташа, Албрехт Ральф. Электронные таблицы Excel 5.0: Практ. Пособие. - М.: ЭКОМ., 1994.
5. П. Нортон. Программно-аппаратная организация IBM PC./ Пер.с англ. – М.: Радио и связь, -1992.
6. Керниган Б., Ритчи Д. Язык программирования Си. – М.: Финансы и статистика, 1985.
7. Уэйт М., Прата С., Мартин Л. Язык Си. – М.: Мир, 1988.
8. Бруно Бабе. Просто и ясно о Borland C++: Пер. с англ. – М.: Бином,1992.
9. Касаткин А. И., Вальвачев А. Н. От TURBO C к Borland C++. - Мн.: Выш. шк., 1992.