

**Министерство сельского хозяйства РФ**  
**Кубанский государственный аграрный**  
**университет**

**Кафедра Системного анализа и обработки информации**

# **Разработка приложений на языке C++**

**Методические указания к лабораторным работам по дисциплине  
«Программирование»**

для студентов первого курса направлений подготовки

09.03.03 «ПРИКЛАДНАЯ ИНФОРМАТИКА»

и

38.03.05 «БИЗНЕС-ИНФОРМАТИКА»

всех форм обучения

**Краснодар, 2014**

Работу подготовили по решению методической комиссии факультета прикладной информатики и кафедры Системного анализа и обработки информации (протокол №6 от 19.02.2007 г.)

канд. техн. наук, доцент А.Г. Мурлин,  
ст. преп. Е.А. Иванова,  
ст. преп. Н.В. Ефанова,  
ст. преп. С.А. Яхонтов

**Разработка приложений на языках высокого уровня.** Методические указания к лабораторным работам для студентов первого курса направлений подготовки 09.03.03 «ПРИКЛАДНАЯ ИНФОРМАТИКА» и 38.03.05 «БИЗНЕС-ИНФОРМАТИКА» всех форм обучения / Кубан. гос. аграрн. ун-т., Сост. А.Г. Мурлин, Е.А. Иванова, Н.В. Ефанова, 2015, 112 с.

Составлены в соответствии с рабочей программой курса “Программирование” для студентов первого курса направлений подготовки 09.03.03 «ПРИКЛАДНАЯ ИНФОРМАТИКА» и 38.03.05 «БИЗНЕС-ИНФОРМАТИКА».

Содержат описание лабораторных работ, методические указания к их выполнению и требования к оформлению отчета.

Табл. 6. Ил. 32. Библиогр.: 13 назв.

Рецензенты: проф. Т.П. Барановская (КубГАУ),  
проф. Л.А. Максименко (КубГТУ).

## СОДЕРЖАНИЕ

Лабораторная работа №1. Разработка приложений, использующих строковые типы данных.....	4
Лабораторная работа №2. Разработка приложений с использованием структур.....	15
Лабораторная работа №3. Разработка приложений с использованием динамических переменных .....	22
Лабораторная работа №4. Разработка приложений с использованием динамических массивов.....	37
Лабораторная работа №5. Разработка приложений с использованием файлового ввода/вывода.....	44
Лабораторная работа №6. Разработка приложений в среде «Borland C++Builder 6».....	59
Лабораторная работа №7. Программирование с использованием компонент работы со строками .....	71
Лабораторная работа №8. Разработка приложений с использованием диалогов для выбора файлов и главного меню.....	78
Лабораторная работа №9. Разработка приложений с использованием средств для отображения графической информации .....	88
Лабораторная работа №10. Разработка приложений с использованием доступа к таблицам баз данных .....	96
ЛИТЕРАТУРА.....	112

# Лабораторная работа №1. Разработка приложений, использующих строковые типы данных

## 1 Цель работы

Изучить принципы разработки программ с использованием строк. Изучить стандартные функции работы со строками модуля `string.h` и модуля `stdlib.h`.

## 2 Порядок выполнения работы

Получить задание на выполнение лабораторной работы (раздел 6) согласно своему варианту. Разработать и отладить программу. Составить и защитить отчет о лабораторной работе у преподавателя.

## 3 Содержание отчета

- наименование и цель работы;
- задание на лабораторную работу согласно варианту;
- схема алгоритма, текст программы на алгоритмическом языке;
- результаты работы программы.

## 4 Краткая теория

**Строкой** называется последовательность символов определенной длины. Таким образом, строковый тип данных определяет множество символьных цепочек произвольной длины (от нуля символов до заданного их числа). Строковый тип обобщает понятие символьного массива, позволяя динамически изменять длину строки.

Строки языка C++ имеют одну особенность: последний символ каждой строки – нулевой символ. Этот символ пишется как `\0` и является символом, имеющим нулевой код ASCII; он служит для обозначения конца строки. Например, рассмотрим следующие два объявления:

```
char s1[5] = {'h','e','l','l','o'};    //не строка!  
char s2[5] = {'t','e','s','t','\0'};    //строка!
```

Оба массива являются массивами символов, но только второй из них – строка.

Инициализировать символьный массив можно также обычным присвоением строковой константы:

```
char s3[10] = "Hello!";  
char s4[] = "new string";
```

При этом необходимо удостовериться, что массив достаточно большой, чтобы вместить все символы строки, включая нулевой. Если размер массива указан больше, чем присваиваемая строка, оставшиеся элементы заполняют-

ся нулевыми символами.

Строки в кавычках всегда неявно включают конечный нулевой символ. Кроме того, различные средства ввода C++ для чтения строки с клавиатуры в массив символов автоматически добавляют конечный нулевой символ. Таким образом, внутреннее представление в памяти строки `s3` предыдущего примера будет следующим (рис. 1.1).

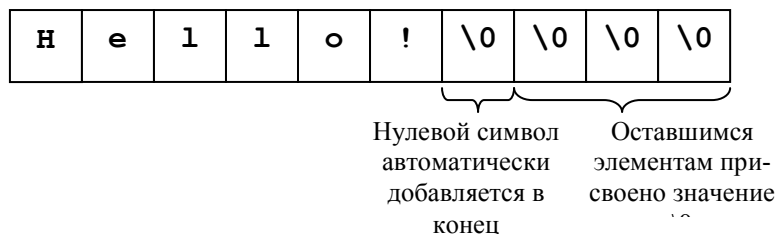


Рисунок 1.1 – Внутреннее представление строк C++

Никаких ограничений на длину строки в C++ нет.

Также для описания строковых данных применяются так называемые динамические массивы, работа с которыми производится с помощью указателей. Например:

```
char* str = "строка";
```

Практически все функции работы со строками возвращают или имеют в качестве параметров данные типа `char*`. При этом в качестве фактических параметров им могут передаваться обычные (статические) символьные массивы.

#### 4.1 Особенности ввода с клавиатуры строковых данных

При вводе с клавиатуры строковых данных с помощью объекта входного потока `cin` возникают следующие трудности.

Во-первых, нельзя вводить нулевой символ с клавиатуры, поэтому `cin`, будучи не в состоянии определить конец вводимой строки, использует для этого первый введенный пробельный символ. Практическим результатом этого является то, что при попытке ввести с клавиатуры в символьный массив строку из нескольких слов реальным значением массива будет только первое слово вводимой строки, до первого пробела. Например:

```
char name[20];  
cin>>name;      //вводим: Вася Иванов  
cout<<name;     //выводится: Вася
```

Другой проблемой является то, что вводимая строка может оказаться длиннее, чем массив назначения.

Избежать вышеуказанных трудностей помогает использование для ввода строк специальной функции класса `istream` – `getline()`. Прототипы ее следующие:

```
istream& getline(signed char*,int,char='\n');  
istream& getline(unsigned char*,int,char='\n');
```

Первый параметр – название массива, предназначенного для того, чтобы сохранить вводимую строку, второй параметр ограничивает количество символов, которые нужно считать. Если этот предел, скажем, 20, функция читает не больше чем 19 символов, оставляя участок памяти, чтобы автоматически добавить нулевой символ в конец. Третий параметр задает символ-разделитель (по умолчанию – символ новой строки). Функция останавливает чтение ввода, когда достигает числового предела или когда читает символ-разделитель, независимо от того, какое из двух событий произошло первым.

Для предыдущего примера функцию `getline()` можно было бы использовать следующим образом:

```
cin.getline(name,20);
```

В этом случае полная строка (возможно, из нескольких слов) считывается в массив `name` при условии, что строка состоит из 19 или меньшего количества символов.

Смешанный строчно-числовой ввод тоже может вызывать проблемы. Рассмотрим следующий пример:

```
int year;  
cin>>year;  
char name[20];  
cin.getline(name,20);
```

Здесь у пользователя никогда не будет возможности ввести переменную `name`. Проблема состоит в том, что когда `cin` читает значение `year`, то оставляет символ новой строки, сгенерированный с помощью клавиши ENTER, во входной очереди. Затем `cin.getline()` воспринимает символ новой строки, как будто это пустая строка, и назначает ее массиву `name`. Исправить эту ошибку можно, читая и отбрасывая символ новой строки перед чтением переменной `name`. Это можно сделать с помощью функции `get()`:

```
cin>>year;  
cin.get();
```

или:

```
(cin>>year).get();
```

## 4.2 Операции со строками

В C++ существует два пути обработки переменных строкового типа. *Первый путь* предполагает обработку всей строки как единого целого (единого объекта). *Второй путь* рассматривает строку как составной объект, состоящий из отдельных символов, т.е. элементов типа `char`, которые при обработке доступны каждый в отдельности.

Так, например, первый путь предоставляет возможность инициализации строчной переменной за одну операцию значением целой строки символов:

```
char s1[20] = "Первая строка!";
```

Присваиваемое значение строки заключается в двойные кавычки. Следует обратить внимание, что присвоить, таким образом, значение символьному массиву можно только при его объявлении. Дальнейшее изменение содержимого строки осуществляется с помощью специальных функций, описанных ниже.

Второй путь обеспечивает доступ к отдельным символам строки по номеру их позиции. То есть применяется конструкция "переменная с индексом", где в [ ] указывается номер позиции элемента (символа) в строке. При этом нижняя граница индекса равна 0.

Например:

```
s1[0]='п';  
//занести в качестве первого символа строки 'п'  
s1[6]=95;  
/*занести в качестве седьмого символа строки '_'  
(ASCII-код '_' соответствует числу 95)*/
```

Сравнение строковых данных также происходит с помощью специально определенных функций, при выполнении которых действуют следующие правила:

а) сравнение происходит посимвольно, начиная с первого символа: сравниваются коды соответствующих символов до тех пор, пока не нарушится равенство, при этом сразу делается вывод о знаке неравенства;

б) строки равны, если имеют одинаковую длину и посимвольно эквивалентны.

в) более короткая строка всегда меньше более длинной.

Например:

```
"Minus" < "minus"    (код 'M' < код 'm');  
"minut" > "minus"    (код 't' > код 's');  
"Minus" > "'min"      (длина первой строки больше);  
"минус"=="минус"     (равны по длине и совпадают посимвольно).
```

### 4.3 Функции работы со строками

Язык C++ предоставляет в распоряжение пользователей целый ряд функций, предназначенных для обработки строковых и символьных данных. Эти функции содержатся в модулях `stdlib.h`, `string.h` и `ctype.h`.

#### 4.3.1 Функции модуля `stdlib.h`

В данном модуле содержится ряд функций, предназначенных для преобразования строковых данных в числовые.

```
int atoi(char* str);
```

Преобразует строку `str` в десятичное число.

```
char* itoa(int v, char* str, int baz);
```

Преобразует целое **v** в строку **str**. При изображении числа используется основание **baz** ( $2 \leq \text{baz} \leq 36$ ). Для отрицательного числа и **baz=10** первый символ – «минус» (-).

```
long atol(char* str);
```

Преобразует строку **str** в длинное десятичное число.

```
char* ltoa(long v, char* str, int baz);
```

Преобразует длинное целое **v** в строку **str**. При изображении числа используется основание **baz** ( $2 \leq \text{baz} \leq 36$ ).

```
char* ultoa(unsigned long v, char* str, int baz);
```

Преобразует беззнаковое длинное целое **v** в строку **str**.

```
double atof(char* str);
```

Преобразует строку **str** в вещественное число типа **double**.

```
char* gcvt(double v, int ndec, char* str);
```

Преобразует вещественное число **v** типа **double** в строку **str**. количество значащих цифр задается параметром **ndec**.

### 4.3.2 Функции модуля **string.h**

В данном модуле описаны функции, позволяющие производить различные действия над строками: преобразование, копирование, сцепление, поиск и т.п.

```
char* strcat(char* sp, char* si);
```

Добавляет строку **si** к строке **sp** (конкатенация строк).

```
char* strncat(char* sp, char* si, int kol);
```

Добавляет **kol** символов строки **si** к строке **sp** (конкатенация).

```
char* strdup(const char* str);
```

Выделяет память и переносит в нее копию строки **str**.

```
char* strset(char* str, int c);
```

Заполняет строку **str** заданным символом **c**.

```
char* strnset(char* str, int c, int kol);
```

Заполняет первые **kol** символов строки **str** заданным символом **c**.

```
unsigned strlen(char* str);
```

Вычисляет длину строки **str**.

```
char* strcpy(char* sp, char* si);
```

Копирует байты строки **si** в строку **sp**.

```
char* strncpy(char* sp, char* si, int kol);
```

Копирует **kol** символов строки **si** в строку **sp** («хвост» отбрасывается или дополняется пробелами).

```
int strcmp(char* str1, char* str2);
```

Сравнивает строки **str1** и **str2**. Результат отрицателен, если **str1 < str2**; равен нулю, если **str1 == str2** и положителен, если **str1 > str2** (сравнение беззнаковое).

```
int strncmp(char* str1, char* str2, int kol);
```



Аналогично предыдущему, но сравниваются только первые **kol** символов.

```
int stricmp(char* str1, char* str2, int kol);
```

Аналогично предыдущему, но при сравнении не делается различия регистров.

```
char* strchr(char* str, int c);
```

Ищет в строке **str** первое вхождение символа **c**.

```
char* strrchr(char* str, int c);
```

Ищет в строке **str** последнее вхождение символа **c**.

```
char* strstr(const char* str1, const char* str2);
```

Ищет в строке **str1** подстроку **str2**. Возвращает указатель на тот элемент в строке **str1**, с которого начинается подстрока **str2**.

```
char* strpbrk(char* str1, char* str2);
```

Ищет в строке **str1** первое появление любого из множества символов, входящих в строку **str2**.

```
int strspn(char* str1, char* str2);
```

Определяет длину первого сегмента строки **str1**, содержащего только символы из множества символов строки **str2**.

```
int strcspn(char* str1, char* str2);
```

Определяет длину первого сегмента строки **str1**, содержащего символы, не входящие во множество символов строки **str2**.

```
char* strtok(char* str1, const char* str2);
```

Ищет в строке **str1** лексемы, выделенные символами из второй строки.

```
char* strlwr(char* str);
```

Преобразует буквы верхнего регистра в строке в соответствующие буквы нижнего регистра.

```
char*strupr(char* str);
```

Преобразует буквы нижнего регистра в строке **str** в буквы верхнего регистра.

### 4.3.3 Функции модуля ctype.h

Данный модуль содержит функции проверки и преобразования символов, которые могут быть полезны при посимвольной обработке строк.

```
int isalnum(int c);
```

Дает значение не ноль, если **c** – код буквы или цифры (A-Z, a-z, 0-9), и ноль – в противном случае.

```
int isalpha(int c);
```

Дает значение не ноль, если **c** – код буквы (A-Z, a-z), и ноль – в противном случае.

```
int isaascii (int c);
```

Дает значение не ноль, если **c** есть код ASCII, т.е. принимает значение от 0 до 127, в противном случае – ноль.

```
int iscntrl (int c);
```

Дает значение не ноль, если **c** – управляющий символ с кодами 0x00-0x1F или 0x7F, и ноль – в противном случае.

```
int isdigit (int c);
```

Дает значение не ноль, если **c** – цифра (0-9) в коде ASCII, и ноль – в противном случае.

```
int isgraph(int c);
```

Дает значение не ноль, если **c** – видимый (отображаемый) символ с кодом 0x21-0x7E, и ноль – в противном случае.

```
int islower(int c);
```

Дает значение не ноль, если **c** – код символа в нижнем регистре (a-z), и ноль – в противном случае.

```
int isprint (int c);
```

Дает значение не ноль, если **c** – печатный символ с кодом 0x20-0x7E, и ноль – в противном случае.

```
int ispunct (int c);
```

Дает значение не ноль, если **c** – символ-разделитель (соответствует `isctrl` или `isspace`), и ноль – в противном случае.

```
int isspace(int c);
```

Дает значение не ноль, если **c** – обобщенный пробел: пробел, символ табуляции, символ новой строки или новой страницы, символ возврата каретки (0x09-0x0D, 0x20), и ноль – в противном случае.

```
int isupper (int c);
```

Дает значение не ноль, если **c** – код символа в верхнем регистре (A-Z), и ноль – в противном случае.

```
int isxdigit (int c);
```

Дает значение не ноль, если **c** – код шестнадцатеричной цифры (0-9, A-F, a-f), и ноль – в противном случае.

```
int toascii(int c);
```

Преобразует целое число **c** в символ кода ASCII, обнуляя все биты, кроме младших семи. Результат от 0 до 127.

```
int tolower(int c);
```

Преобразует код буквы **c** к нижнему регистру, остальные коды не изменяются.

```
int toupper(int c);
```

Преобразует код буквы **c** к верхнему регистру, остальные коды не изменяются.

## 5 Пример программы

**5.1** Дана строка, в которой слова разделены одним или несколькими пробелами. Выделить слова из этой строки и отсортировать их в алфавитном порядке.

```
#include <iostream.h>
#include <string.h>
void main()
{
    //макс. кол-во символов и слов в строке
    const n=100,kol=10;
    //описание типа "строка"
    typedef char String[n];
    String s,mas[kol];
    cout<<"Введите строку: ";
    cin.getline(s,100);
    cout<<s<<endl;
    int k=0; //кол-во слов
    //выделить первое слово из строки
    char* token=strtok(s," ");
    //пока не кончились слова
    while (token)
    {
        //скопировать слово в массив
        strcpy(mas[k++],token);
        //выделить следующее слово (NULL показывает,
        //что мы продолжаем работать с той же строкой)
        token = strtok(NULL," ");
    }
    cout<<"Массив слов:\n";
    for (int i=0;i<k;i++)
        cout<<mas[i]<<": " <<strlen(mas[i])<<endl;
    String t;
    //сортировка слов методом "пузырька"
    for (i=0;i<k;i++)
        for (int j=k-1;j>i;j--)
            if (strcmp(mas[j],mas[j-1])<0)
            {
                strcpy(t,mas[j]);
                strcpy(mas[j],mas[j-1]);
                strcpy(mas[j-1],t);
            }
    cout<<"Массив слов после сортировки:\n";
    for (i=0;i<k;i++)
        cout<<mas[i]<<endl;
}
```

**5.2** Дана текстовая строка. Сформировать новую строку из тех символов, которые стоят на нечетных позициях в исходной строке и не являются

цифрами.

```
#include <iostream.h>
#include <string.h> //для функции strlen()
#include <ctype.h>  //для функции isdigit()
void main()
{
    const n=100;
    char s1[n],s2[n];
    cout<<"Введите строку: ";
    cin.getline(s1,100);
    cout<<s1<<endl;
    int k=0;  //кол-во символов в новой строке
    //просмотр символов исходной строки
    for (int i=0;i<strlen(s1);i++)
        //если выполняются заданные условия,
        if (i%2!=0 && !isdigit(s1[i]))
            //то добавляем символ в новую строку
            s2[k++]=s1[i];
    //в конец строки заносим символ с кодом 0
    s2[k]='\0';
    cout<<"Result: " << s2 << endl;
}
```

### 5.3 Найти сумму цифр введенного числа.

```
#include <iostream.h>
#include <string.h> //для функции strlen()
#include <stdlib.h> //для функции itoa()
void main()
{
    long n;
    cout<<"Введите число: ";
    cin>>n;
    int sum=0;
    char str[20];
    //переводим число в строку (10-я система счисления)
    itoa(n,str,10);
    for (int i=0;i<strlen(str);i++)
        //очередная цифра = код символа - код символа '0'
        sum += str[i]-'0';
    cout<<"sum = " << sum << endl;
}
```

## 6 Варианты заданий для самостоятельного решения

1. Дана последовательность слов. Проверить правильность написания сочетаний "жи", "ши", "ча", "ща", "чу", "щу". Если надо, то исправить ошибки их написания.
2. Дан список фамилий. Составить новый список, который будет содержать

только женские фамилии. (Примечание: те фамилии, по которым пол трудно определить, не считать).

3. Найти произведение и минимальную четную цифру, составляющую некоторое число X.

4. Написать программу, которая запрашивает полное имя пользователя, затем пользователь вводит строку символов. Компьютер должен подсчитать, сколько раз в тексте встречается каждая буква из имени.

5. Перевести число из 10-ой системы в 16-тиричную и наоборот.

6. Дан текст, состоящий не менее чем из пяти слов. Сформировать строку, в которую попадут только те слова, где одинаковые буквы встречаются более двух раз. Например, молоко.

7. Из имеющегося набора слов выбрать наиболее длинное, в котором все буквы разные. Например, *лейкопластырь*, *неряшливость*.

8. Дана строка цифр. Сформировать строку, в которую войдут все цифры из исходной строки, кроме той, которая встречается наибольшее количество раз. Ее вывести отдельно.

9. Дана последовательность, состоящая из цифр, букв и знаков пунктуации в произвольном порядке. Подсчитать чего больше и составить строки только из цифр, букв и знаков пунктуации. Вывести строки в порядке возрастания их длины.

10. Дана некоторая последовательность букв русского алфавита. Написать программу, которая запрашивает Ваше имя и определяет, можно ли из букв исходной строки составить его. Если нет, то выводит буквы, которых не хватает.

11. Дан текст, записанный в виде криптограммы (шифrogramмы), в которой буквы истинного текста размещаются в позициях, кратных 3. Прочитать исходный текст.

12. Дана строка цифр. Составить из них 5-значные числа. Если на последнее число не хватит цифр, дополнить его первыми цифрами исходной строки.

13. Перевести число из 2-ой системы в 10-тичную и наоборот.

14. Дан текст. Найти самое короткое слово, длиной не более 3 букв.

15. Написать программу, которая переставит слова в строке по мере увеличения их длины.

16. Дан текст, состоящий не менее чем из пяти слов. Вывести на экран слова, в которых отсутствует буква "Е".

17. Дана последовательность чисел. Написать программу, которая запрашивает дату рождения человека (можно только число, месяц или год) и определяет, сколько раз эта дата встречается в строке.

18. Найти количество нечетных цифр и максимальную нечетную цифру, составляющую некоторое число X.

19. Дан текст, состоящий не менее чем из пяти слов. Определить, есть ли в нем слова, начинающиеся и заканчивающиеся с буквы "А", а также количе-

ство таких слов.

20. Дан текст, состоящий не менее чем из пяти слов. Написать программу, которая выполняет следующие преобразования: в первом слове делает заглавной первую букву, во втором слове делает заглавной вторую букву и т.д. Примечание: слова вводятся строчными латинскими буквами!

21. Написать программу, которая проверяет, является ли введенное слово палиндромом. Палиндромом называется слово, которое читается одинаково слева направо и справа налево, например, "КАЗАК".

22. Дана строка слов. Определить буквы, которые встречаются наибольшее и наименьшее количество раз.

23. Дан текст, состоящий не менее чем из пяти слов. Вывести на экран слова, которые имеют одинаковые окончания "ИЯ", "ИСТ", "ИКА".

24. Найти минимальную и максимальную цифры среди четных и нечетных цифр, составляющих некоторое число X.

25. Дан текст. Преобразовать его по следующему правилу: если нет символа '\*', то оставить его без изменения, иначе заменить каждый символ, встречающийся после первого вхождения символа '\*', на символ '-'.

26. Подсчитать сумму и количество всех цифр, входящих в некоторое предложение, вводимое с клавиатуры.

27. Дан текст, состоящий не менее чем из семи слов. Все слова из четырех букв записать наоборот.

28. Дана последовательность из латинских букв и цифр. Определить, чего больше.

29. Дан текст, оканчивающийся точкой. Найти количество слов, у которых первый и последний символы совпадают.

30. Найти среднее арифметическое значение между минимальной и максимальной цифрами некоторого числа X.

31. Дан текст, состоящий не менее чем из семи слов. Определить, есть ли в нем слова, начинающиеся с буквы "Ф", а также количество таких слов.

32. Дан текст. Исключить из него символы, расположенные между скобками '(', ')'. Сами скобки тоже должны быть исключены. Предполагается, что внутри каждой пары скобок нет других скобок.

## Лабораторная работа №2. Разработка приложений с использованием структур

### 1 Цель работы

Получить практические навыки использования комбинированного типа данных СТРУКТУРА в разработке приложений.

### 2 Порядок выполнения работы

Получить задание на выполнение лабораторной работы согласно своему варианту. Разработать и отладить программу. Составить и защитить отчет о лабораторной работе у преподавателя.

### 3 Содержание отчета

- наименование и цель работы;
- задание на лабораторную работу согласно варианту;
- схема алгоритма, текст программы на алгоритмическом языке;
- результаты работы программы.

### 4 Краткая теория

*Структуры* – это составные типы данных, построенные с использованием других типов. Они представляют собой объединенный общим именем набор данных различных типов. Именно тем, что в них могут храниться данные разных типов, они и отличаются от массивов, хранящих данные одного типа.

Отдельные данные структуры называются *элементами* или *полями*. Все это напоминает запись в базе данных, только хранящуюся в оперативной памяти компьютера.

Простейший вариант объявления структуры может выглядеть следующим образом:

```
struct address
{
    char name[25];
    int number;
    char street[20];
    char town[15];
    long zip;
};
```

Ключевое слово **struct** начинает определение структуры. Идентификатор **address** – обозначение, имя типа структуры. Оно используется при объявлении переменных структур данного типа. Имена, объявленные в фигур-

ных скобках описания структуры – это элементы структуры. Элементы одной и той же структуры должны иметь уникальные имена, но две разные структуры могут содержать не конфликтующие элементы с одинаковыми именами. Каждое определение структуры должно заканчиваться точкой с запятой.

Определение **address** содержит пять элементов. Предполагается, что такая структура может хранить данные о почтовом адресе кого-либо. Типы данных разные: элементы **name**, **street** и **town** – строки, хранящие соответственно фамилию адресата, улицу и город, где он проживает. Элемент **number** целого типа хранит номер дома, элемент **zip** типа **long** хранит сведения об индексе.

Само по себе объявление структуры не резервирует никакого пространства в памяти; оно только создает новый тип данных, который может использоваться для объявления переменных. Переменные структуры объявляются так же, как переменные других типов. Строка

```
address addr, addrArray[10];
```

объявляет переменную **addr** типа **address** и массив **addrArray** – с 10 элементами типа **address**.

Переменные структуры могут объявляться и непосредственно в объявлении самой структуры после закрывающейся фигурной скобки. В этом случае указание имени типа структуры не обязательно:

```
struct  
{  
    char name[25];  
    int number;  
    char street[20];  
    char town[15];  
    long zip;  
} addr, addrArray[10];
```

При определении структур возможна их инициализация, т.е. задание начальных значений их элементов. Например, введя структурный тип **address**, можно следующим образом определить и инициализировать конкретную структуру:

```
Address MyAddr = {"Петров Петр Петрович", 5,  
"Кореновская", "Краснодар", 350005};
```

Для доступа к элементам структуры используется операция точка (**.**), которая обращается к элементу структуры по имени объекта или по ссылке на объект. Например:

```
strcpy(addr.name, "Иванов Иван Иванович");  
addr.number = 79;  
strcpy(addr.street, "Красная");  
strcpy(addr.town, "Краснодар");  
addr.zip = 350000;
```

Структурный тип можно сопоставлять с типами, вводимыми с помощью служебного слова **typedef**:



```
typedef struct PRIM
{ char name[10];
  long sum;
} NEWSTRUCT;
NEWSTRUCT st, as[8];
PRIM prim_st, prim_as[8];
```

В этом примере введен структурный тип **PRIM**, и ему дополнительно с помощью **typedef** присвоено имя **NEWSTRUCT**. Эти два имени совершенно равноправны. При определении с помощью **typedef** имени для структурного типа у последнего может отсутствовать основное имя. Например, структурный тип можно ввести и следующим образом:

```
typedef struct
{ char name[10];
  long sum;
} STRUCT;
STRUCT struct_st, struct_as[8];
```

Элементом определяемой структуры может быть структура, тип которой уже определен:

```
struct point
{ int x;
  int y;
};
struct rect
{ point LeftTop;
  point RightBottom;
  char color[15];
};
```

Рассмотрим «взаимоотношение» структур и функций.

Функция может возвращать структуру как результат:

```
struct message
{ char name[20];
  int number;
};
message func1() ; // Прототип функции
```

Функция может возвращать ссылку на структуру:

```
message& func2() ; // Прототип функции
```

Через аппарат параметров информация о структуре может передаваться в функцию либо непосредственно, либо с помощью ссылки:

```
void func3 (message str) ; // Прямое использование
void func4 (message& rst); // С помощью ссылки
```

Напомним, что применение ссылки на объект в качестве параметра позволяет избежать дублирования объекта в памяти.

## 5 Примеры программ

Программа находит в массиве данных о людях самого младшего (по годам) человека.

```

#include <iostream.h>
#include <string.h>
struct Date
{
    int Day;
    int Month;
    int Year;
};
struct Person
{
    char FIO[20];
    Date Birthday;
};
void InputPersInfo(int n, Person arr[])
{
    for (int i=0; i<n; i++)
    {
        cout<<"Введите ФИО: ";
        cin.getline(arr[i].FIO, 20);
        cout<<"Введите дату рождения (ДД ММ ГГГГ): ";
        cin>>arr[i].Birthday.Day>>arr[i].Birthday.Month
            >>arr[i].Birthday.Year;
        cin.get();
    }
}
void OutputPersInfo(int n, Person arr[])
{
    for (int i=0; i<n; i++)
    {
        cout<<arr[i].FIO<<": ";<<arr[i].Birthday.Day<<'.';
        cout<<arr[i].Birthday.Month<<'.'<<arr[i].Birthday.
            Year<<endl;
    }
}
void main()
{
    const N=3;
    Person PersArr[N];
    InputPersInfo(N, PersArr);
    OutputPersInfo(N, PersArr);
    Person Min = PersArr[0];
    int Ind = 0;
    for (int i=1; i<N; i++)
        if (PersArr[i].Birthday.Year > Min.Birthday.Year)
        {
            Min = PersArr[i];
            Ind = i;
        }
    cout<<"Самый младший - "<<PersArr[Ind].FIO<<endl;
}

```

## 6 Варианты заданий для самостоятельного решения

1. Для каждого из N студентов группы известны ФИО и оценки (в баллах) по четырем дисциплинам. Найти среднюю оценку каждого студента и выбрать человека, имеющего максимальный средний бал.

2. Дан массив студентов некоторого московского ВУЗа: ФИО, возраст, регион, факультет. Вывести на экран результирующую таблицу: регион, количество студентов из этого региона. Отсортировать данные по названию региона.
3. Багаж пассажира характеризуется количеством вещей и общим весом вещей. Дан массив, содержащий сведения о багаже нескольких пассажиров. Найти багаж, средний вес одной вещи в котором отличается не более чем на 0.3 кг от общего среднего веса одной вещи.
4. Дан массив данных о клиентах пункта проката автомобилей: ФИО, адрес (улица, дом, квартира) и марка машины. Во второй массив записать данные только тех людей, кто ездит на "Audi".
5. Дан список английских глаголов: тип (правильный/неправильный), первая форма, вторая форма, третья форма. Вывести только те глаголы, у которых все три формы совпадают. Вывести только правильные (неправильные) глаголы в алфавитном порядке.
6. Рациональное число можно представить записью с двумя полями: числитель и знаменатель. Дан массив из  $N$  рациональных чисел. Разработать функцию для нахождения максимального среди них.
7. Дан массив, в котором хранятся данные о расписании самолетов на неделю: пункт назначения, время вылета, количество свободных мест. В кассу аэропорта обращается пассажир, желающий купить  $n$  билетов на первую половину сегодняшнего дня до выбранного пункта назначения. Определить, есть ли возможность выполнить его заказ, предоставить ему возможные варианты на выбор, отсортировав по времени вылета.
8. Дан массив данных о работниках фирмы: ФИО и дата поступления на работу (месяц, год). Во второй массив записать только данные тех из них, кто на сегодняшний день проработал уже не менее 5 лет.
9. Дан массив данных о работниках фирмы: фамилия, имя, отчество, адрес (улица, дом, квартира) и дата поступления на работу (месяц, год). Определить, есть ли в списке Петровы (Петров, Петрова), если есть, то вывести их адрес (адреса).
10. Дан список иногородних студентов из  $n$  человек: ФИО, адрес (город, улица, дом-квартира), приблизительное расстояние до Краснодара. Для них в общежитии выделено  $k$  мест. Вывести список студентов, которых необходимо селить в общежитие в первую очередь. Критерий отбора: расстояние до города.
11. Дан массив данных о студентах некоторой группы: фамилия, имя, отчество и дата рождения (день, месяц, год). Вывести на экран фамилию и имя тех студентов, у кого сегодня день рождения (сегодняшнюю дату вводить с клавиатуры).
12. Дан массив, содержащий сведения о студентах некоторой группы: ФИО, оценки по пяти экзаменационным дисциплинам. Вывести сначала студентов,

получающих повышенную стипендию, затем обычную, и, наконец, без стипендии. Организовать поиск студента по фамилии с выводом информации о нем.

13. Дан массив книг: название, автор, количество страниц. Вывести все книги А.С. Пушкина в алфавитном порядке.

14. Дан массив, в котором хранятся данные о расписании поездов на сегодняшний день: номер поезда, название (т.е. откуда – куда, например, Новороссийск-Москва), время прибытия на станцию и время отправления (часы, минуты). По данному времени определить, какие из поездов стоят сейчас на станции.

15. Дан массив, содержащий сведения о студентах некоторой группы: ФИО, адрес (улица, дом, квартира) и телефон (если есть). Вывести на экран сведения о тех из них, до которых нельзя дозвониться.

16. Точка плоскости может быть представлена двумя координатами X и Y. Дан массив, содержащий N точек. Найти точки, которые максимально удалены от начала координат.

17. Багаж пассажира характеризуется количеством вещей и общим весом вещей. Дан массив, содержащий сведения о багаже нескольких пассажиров. Выяснить имеется ли пассажир, багаж которого состоит из одной вещи весом менее 30 кг.

18. Комплексное число можно представить через реальную ( $Re$ ) и мнимую ( $Im$ ) часть. Разработать процедуры сложения, вычитания, умножения и деления комплексных чисел. Пусть  $Z_1=(Re_1, Im_1)$ ,  $Z_2=(Re_2, Im_2)$ , тогда: 1) сумма:  $Z = (Re_1+Re_2, Im_1+Im_2)$ ; 2) разность:  $Z = (Re_1-Re_2, Im_1-Im_2)$ ; 3) произведение:  $Z = (Re_1*Re_2 - Im_1*Im_2, Re_1*Im_2 + Re_2*Im_1)$ ; 4) частное:  $Z = \frac{Re_1 \cdot Re_2 + Im_1 \cdot Im_2}{Re_2^2 + Im_2^2}, \frac{Re_2 \cdot Im_1 - Re_1 \cdot Im_2}{Re_2^2 + Im_2^2}$ .

19. Дан перечень постельного белья: название (подушка/одеяло/...), цвет, размер (односпальный/полуторный/двухспальный). Подсчитать, сколько комплектов можно составить (в комплекте белье с одинаковым размером и цветом). Вывести то, что осталось вне комплектов.

20. Дан массив, содержащий сведения о студентах группы: фамилия, имя, отчество, дата рождения (день, месяц, год). Найти самого младшего студента по полной дате рождения.

21. Время можно представить с помощью часов, минут и секунд. Написать функцию *проверка(t1,t2)*, проверяющую, предшествует ли время  $t1$  времени  $t2$  (в рамках суток).

22. Для каждого из N студентов группы известны ФИО и оценки (в баллах) по пяти дисциплинам. Вывести на экран фамилию и имя тех из них, у которых количество положительных оценок больше, чем отрицательных.

23. Рациональное число можно представить записью с двумя полями: числитель и знаменатель. Разработать процедуру, позволяющую из неправильной

доби получить правильную. Неправильной называется дробь, у которой числитель больше знаменателя.

24. Рациональное число можно представить записью с двумя полями: числитель и знаменатель. Дан массив рациональных чисел. Найти их сумму и по возможности сократить, выделить целую и дробную части.

25. Багаж пассажира характеризуется количеством вещей и общим весом вещей. Дан массив, содержащий сведения о багаже нескольких пассажиров. Найти число пассажиров, имеющих более двух вещей и число пассажиров, количество вещей которых превосходит среднее число вещей.

26. Имеется список членов коллектива с указанием принадлежности каждого к различным общественным организациям (профком, ученый совет, общество книголюбов и т.п.). Напечатать приглашение всем членам на очередное заседание указанной организации. Задается только вид организации, место и время сбора.

27. Дан массив, содержащий информацию об учениках некоторой школы. Вывести на экран сведения об учениках только десятых классов. На сколько человек в девятых классах больше, чем в десятых.

28. Дан массив, содержащий сведения о книгах: название, жанр, автор. Вывести книги только классического жанра, отсортировав их по фамилии автора. Найти количество таких книг, насколько их больше (меньше), чем остальных.

29. Время можно представить с помощью часов, минут и секунд. Написать функцию *перевод( $t_1, t_2$ )*, присваивающую параметру  $t_2$  время на 1 секунду большее времени  $t_1$  (учесть смену суток).

30. Дан массив книг: название, тип, автор, количество страниц, страна-родина автора. Организовать поиск по автору, по типу. Вывести все книги зарубежных и российских авторов.

31. Рациональное число можно представить записью с двумя полями: числитель и знаменатель. Разработать функцию *сократить( $m$ )* приведения рационального числа  $m$  к несократимому виду.

32. Дан массив данных о работниках фирмы: ФИО и адрес (улица, дом, квартира). Во второй массив записать только тех из них, которые живут на улице Красной. Вывести их на экран в алфавитном порядке.

## Лабораторная работа №3. Разработка приложений с использованием динамических переменных

### 1 Цель работы

Изучить динамические структуры данных. Получить практические навыки в написании приложений, использующих динамические переменные.

### 2 Порядок выполнения работы

Получить задание на выполнение лабораторной работы согласно своему варианту. Разработать и отладить программу. Составить отчет о лабораторной работе и защитить его у преподавателя.

### 3 Содержание отчета

- наименование и цель работы;
- задание на лабораторную работу согласно варианту;
- схема алгоритма, текст программы на алгоритмическом языке;
- результаты работы программы.

### 4 Краткая теория

#### 4.1 Статические и динамические данные

Существует следующая классификация данных: статические данные и динамические данные.

**Статические данные** существуют на протяжении всего времени выполнения программы, и их размер остается неизменным. Любая глобальная структура программы – статическое данное. Однако часто в процессе написания программы возникают ситуации, когда размер структур данных должен меняться. Этим свойством обладают динамические данные.

Для работы с динамическими данными используются **указатели**. **Указатель** – это ссылка на определенную ячейку памяти, начиная с которой записывается значение переменной, поэтому данные такого типа называются еще и *ссылочным* типом данных.

Для размещения динамических переменных используется специальная область памяти – **куча** (динамическая область памяти). Куча («heap») существует отдельно от стекового сегмента, предназначенного для распределения памяти под локальные статические переменные, и размещается в памяти компьютера следом за областью памяти, которую занимает программа и статические данные.

В простейшем случае определение и описание указателя на некоторый объект имеет вид:

**тип \*имя\_указателя;**

где **тип** – обозначение типа; **имя\_указателя** – идентификатор; **\*** - унарная операция раскрытия ссылки (операция разыменования; операция обращения по адресу; операция доступа по адресу), операндом которой должен быть указатель. Например, определение

```
int *p1, *p2, *p3, k;
```

вводит три указателя на объекты целого типа **p1, p2, p3** и одну переменную **k** целого типа.

При определении указателя в большинстве случаев целесообразно выполнить его инициализацию:

```
type *имя_указателя = инициализирующее_выражение;
```

```
type *имя_указателя (инициализирующее_выражение) ;
```

В качестве инициализирующего выражения могут использоваться:

- явно заданный адрес участка памяти;
- указатель, уже имеющий значение;
- выражение, позволяющее получить адрес объекта с помощью операции “&”.

Если значение инициализирующего выражения равно нулю, то нулевое значение преобразуется к пустому (нулевому) указателю, имеющего специальное обозначение **NULL**. Примеры определения указателей:

```
char Ch = 'a' ;           //символьная переменная
char *pCh = &Ch;          //инициализированный указатель
int *ptr (NULL) ;          //нулевой указатель
float *p;                  //неинициализированный указатель
```

После определения (с инициализацией) указателя **pCh** доступ к значению переменной **Ch** возможен как с помощью ее имени, так и с помощью адреса, являющегося значением указателя переменной **pCh**. В последнем случае должна применяться операция разыменования **\*** (получение значения через указатель). Например:

```
cout<<Ch;                 //результат - 'a'
cout<<*pCh;                //результат - 'a'
```

Пустой указатель разыменовывать нельзя.

Присвоив указателю адрес конкретного участка памяти, можно с помощью операции разыменования не только получать, но и изменять содержимое этого участка памяти. Например:

```
*pCh = '+' ;              //Ch станет = '+'
```

Выражение **\*указатель** обладает всеми правами имени переменной. Однако это справедливо лишь для явно проинициализированных указателей. (**\*p = 1.5**; - ошибка!!! Надо присвоить указателю адрес конкретного объекта).

Для связи неинициализированного указателя с новым участком памяти используется операция **new** или присваивается указателю явный адрес:

```
p = new float; //выделили память для переменной типа
               //float и связали указатель p с этим
```

```

//участком памяти
p = (float *)0x5A120B80; // явно заданный адрес

```

Для освобождения динамически выделенной памяти применяется операция **delete**:

```
delete p;
```

## 4.2 Односвязные списки

Наиболее распространенными динамическими структурами являются *односвязные списки*.

Элемент списка в общем случае представляет собой структуру, состоящую как минимум из двух полей: одно поле типа указатель, а второе – поле данных. Поле-указатель предназначено для связи со следующим элементом списка. Доступ к элементу списка осуществляется только от его начала. Если поле-указатель является пустым указателем (**NULL**) – это означает конец списка.

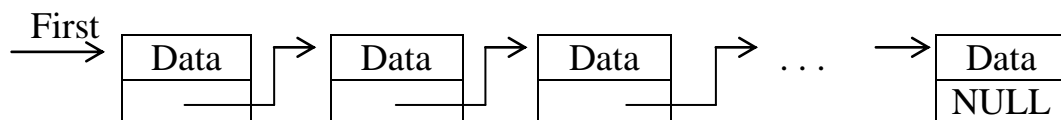


Рисунок 3.1 – Пример представления списка

Пример объявления списка:

```

struct TNode;
typedef TNode* PNode;
struct TNode
{
    int Data;
    PNode Next;
};
PNode First = NULL;

```

Поле **Data** целого типа служит для размещения данных в элементе списка. Другое поле **Next** представляет собой типизированный указатель, служащий для организации списковой структуры.

Далее описан указатель **First** – указатель на первый узел списка (и тем самым на весь список). Последний элемент списка имеет "пустой" указатель (**NULL**). Первоначально, когда список пуст, указатель **First** устанавливается в **NULL**.

Следует отметить, что необходимо всегда иметь указатель на первый элемент списка, так как он указывает и на весь список в целом. Потеря такого указателя влечет за собой потерю доступа ко всему списку. Таким образом, если необходимо получить доступ, например, к предпоследнему элементу, то вводят так называемый рабочий указатель, который также устанавливается на начало данного списка, и все дальнейшие операции происходят через этот указатель.

Для доступа к полям структуры через указатель на эту структуру ис-



пользуется специальная операция: «->», например:

```
First->Data = 3;  
First->Next = NULL;
```

В языке C++ проводится принцип, согласно которому, идентификатор, перед его использованием, должен быть описан. Поэтому в вышеуказанном примере имеется так называемое предварительное описание структуры **TNode**, необходимое просто для «введения» в программу этого идентификатора. Однако описание указателя еще не означает создание элемента списка.

Рассмотрим основные операции работы с элементами списками:

### 1) Добавление в начало списка

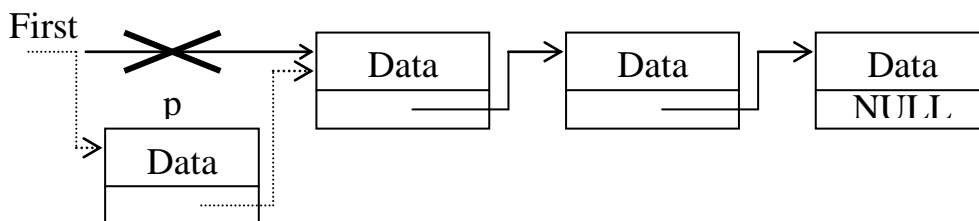


Рисунок 3.2 – Вставка элемента в начало списка

Новый элемент имеет указатель **p**. Операцией **new** под него выделяется память. А для того, чтобы добавить его в начало списка, необходимо присвоить его информационному полю **Data** некоторое значение и "перекинуть" указатели, соответственно рисунку 3.2:

```
p->Data = Data;  
p->Next = First;  
First = p;
```

### 2) Добавление в середину списка

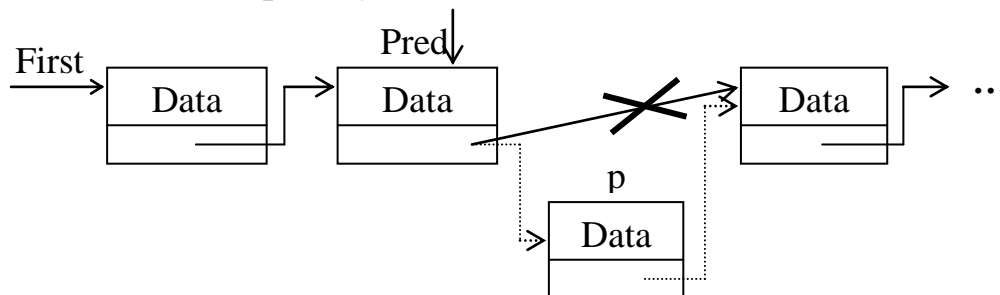


Рисунок 3.3 – Вставка элемента в середину списка

Для добавления в середину списка нового элемента **p** необходимо иметь рабочий указатель **Pred** на элемент, после которого необходимо вставить **p**.

```
PNode p = new TNode; //выделение памяти под новый элемент  
p->Data = Data; //инф.полю присвоить некоторое значение
```

Указатели "перекидываются" соответственно рисунку 3.3:

```
p->Next = Pred->Next;
```

**Pred->Next = p;**

2) *Удаление первого элемента списка.*



Рисунок 3.4 – Удаление элемента из начала списка

Удаление элемента из начала списка происходит очень просто: достаточно всего лишь указатель на начало списка **First** перекинуть на следующий за ним элемент (рисунок 3.4): **First = First->Next;**. "Узким местом" в данном случае является то, что память от удаленного элемента не освободилась. Поэтому целесообразно ввести временный указатель **p**, также указывающий на начало списка (**PNode p = First**). После этого уже "перекинуть" указатели, согласно рисунку 3.4, и удалить этот временный указатель с помощью операции **delete p**, чтобы удалить элемент не только из списка, но и из памяти.

3) *Удаление элемента из середины списка*

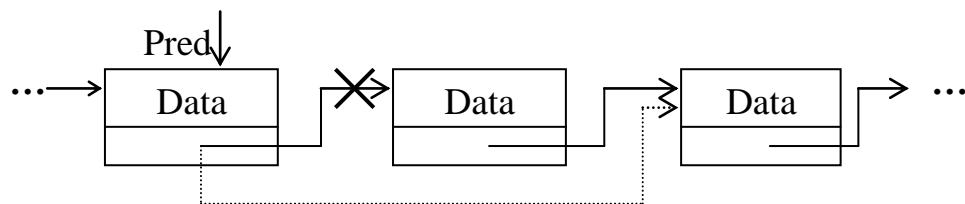


Рисунок 3.5 - Удаление элемента из середины списка

Пусть **Pred** – рабочий указатель на элемент, предшествующий удаляемому. Тогда **Pred->Next = Pred->Next->Next**.

То есть идти по ссылке, хранящейся в **Pred** в поле **Next**, (т.е. адрес удаляемого элемента), затем идти по ссылке **Next** удаляемого элемента (а это адрес следующего за удаляемым элементом), полученное значение записать в поле **Next** элемента **Pred**.

Также как и в случае удаления из начала списка необходимо освободить память от удаляемого элемента. Поэтому вводится временный указатель **p**, указывающий на удаляемый элемент (**p = Pred->Next**). Затем "перекинуть" указатели, согласно рисунку 3.5, и удалить этот временный указатель с помощью операции **delete**.

Пример создания и работы со списком приведен в разделе 5.1.

На списках часто организуют такие динамические структуры данных, как стек и очередь. Стек и очередь – это структуры данных, которые хранят некоторую последовательность значений, и доступ к этой последовательно-

сти ограничен (нельзя прочитать произвольный элемент последовательности). Для этих структур определяется набор допустимых операций, с помощью которых можно записывать и читать элементы последовательностей.

### 4.3 Стек

*Стек* – структура данных с односторонним доступом (рисунок 3.6). Стек является одной из наиболее употребляемых структур данных, которая оказывается весьма удобной при решении различных задач.

Стек работает по принципу LIFO (Last in – first out; последним пришел – первым ушел).

Особенностью стека является то, что в каждый момент времени доступен только один элемент, который находится в конце стека – *вершине стека* (*Top*).

**Операции над стеками:**

1. Занесение элемента в стек – **Push**
2. Выборка элемента из стека – **Pop**
3. Определение пустоты стека – **Empty**
4. Прочтение элемента без его выборки из стека – **StackTop**
5. Запись элемента – **PutItem**

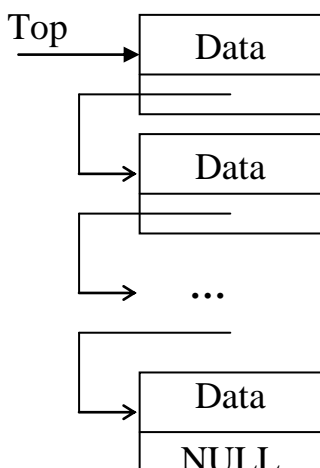


Рисунок 3.6 -Стек

При «добавлении элемента» новое значение добавляется в конец стека. При выполнении операции «взять элемент» – извлекается из него. Операции «прочитать элемент» и «записать элемент» работают с вершиной стека. Операции **Push** и **Pop** соответствуют операциям добавления и удаления первого элемента списка соответственно.

При выполнении операции выборки из стека целесообразно осуществить проверку на пустоту стека.

Пример реализации и работы со стеком приведен в разделе 5.2.

### 4.4 Очередь

*Очередь* – структура данных с двусторонним доступом, то есть в один конец структуры добавляются элементы, а с другого конца исключаются.

Очередь работает по принципу FIFO (First in – first out; первым пришел – первым ушел).

Структура данных очередь полностью аналогична обыденному пониманию этого слова. Когда человек занимает очередь в магазине, он становится в конец очереди, так и элемент, добавляемый в очередь, дописывается в ее

конец. Когда приходит очередь брать товар, человек берет товар и уходит, так и при извлечении элемент берется из начала очереди.

Таким образом, для организации такой структуры используются две

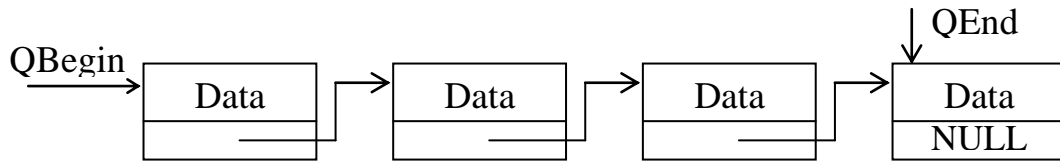


Рисунок 3.7 – Очередь

переменные: для указания начала и конца очереди.

На рисунке 3.7 изображена очередь, где указатель **QBegin** указывает на начало очереди, а указатель **QEnd** указывает на конец очереди.

**Операции над очередями:**

- инициализация очереди – в очередь попадает первый элемент;
- проверить на пустоту;
- добавить элемент в конец;
- взять элемент из начала.

Признаком пустоты очереди является **QBegin=nil**. Это связано с тем, что элементы из очереди берутся только из начала очереди. Операция извлечения элемента из очереди идентична удалению элемента из начала списка. На рисунке 3.8 показана последовательность действий при добавлении элемента в конец очереди.

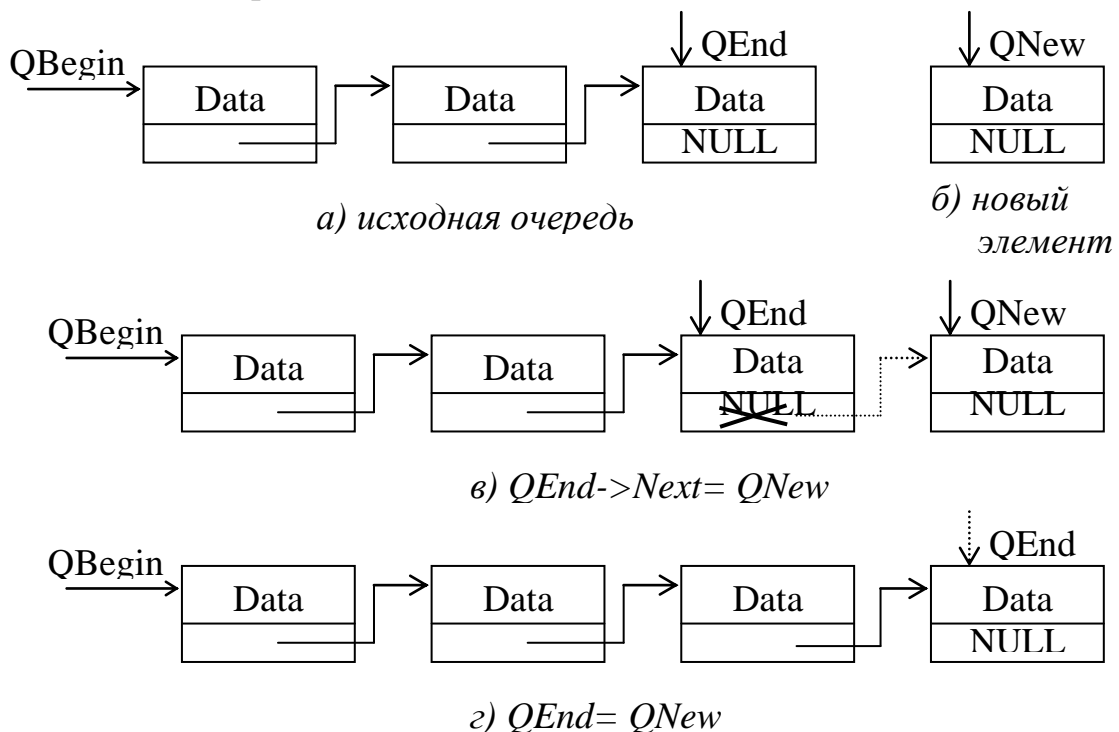


Рисунок 3.8 – Добавление элемента в конец очереди

Исходная очередь (а) состоит из трех элементов. С помощью операции **new** создаем новый элемент (б) с указателем **QNew**, в поле данных (**QNew->Data**) которого записываем некоторое значение; указатель на следующий элемент (**QNew->Next**) устанавливаем в **NULL**, что является признаком того, что этот элемент станет последним.

После этого "перекидываем" указатели так, как показано на фрагментах (в) и (г). Пример реализации и работы с очередью приведен в разделе 5.3.

## 5 Примеры программ

### 5.1 Работа со списком

В данном примере реализованы основные операции работы со списком, информационные поля которого могут хранить целые числа. Кроме того, в виде отдельных функций реализованы операции по заполнению списка элементами, обходу и печати списка, поиска со вставкой новых элементов и поиска с удалением элементов по заданному значению информационного поля. Обозначения, встречающиеся в программе: **First** – указатель на начало списка; **p**, **q** – вспомогательные (рабочие) указатели.

```
#include <iostream.h>
//предварительное описание структуры элемента списка
struct TNode;
//описываем тип-указатель на элемент структуры
typedef TNode* PNode;
//элемент списка
struct TNode
{
    int Data;          //поле данных
    PNode Next;        //поле указателя на следующий элемент
};
//Добавление элемента в начало списка (см. рисунок 3.2)
void AddFirst(PNode& First, int Data)
{
    PNode p = new TNode;    //указатель на новый элемент
    p->Data = Data;
    p->Next = First;
    First = p;
}
//Вставка элемента в середину списка (см. рисунок 4.3)
void AddAfter(PNode& Pred, int Data)
{
    PNode p = new TNode;
    p->Data = Data;
    p->Next = Pred->Next;
    Pred->Next = p;
}
//Вставка элемента в конец списка
void AddLast(PNode& First, int Data)
{
    PNode p1, p2 = First;
    while (p2->Next != NULL)
```

```

        p2 = p2->Next;
        p1 = new TNode;
        p1->Data = Data;
        p2->Next = p1;
        p1->Next = NULL;
    }
    //Удаление элемента из начала списка (см. рисунок 3.4)
    void DelFirst(PNode& First)
    {
        PNode p = First;
        First = First->Next;
        delete p;
    }
    //Удаление элемента из середины списка (см. рисунок 3.5)
    void DelAfter(PNode& Pred)
    {
        PNode p = Pred->Next;
        if (p)
        {
            Pred->Next = p->Next;
            delete p;
        }
    }
    //Удаление элемента из конца списка
    void DelLast(PNode& First)
    {
        PNode p1 = First, p2;
        while (p1->Next->Next != NULL)
            p1 = p1->Next;
        p2 = p1->Next;
        p1->Next = NULL;
        delete p2;
    }
    //Заполнение списка
    void InputList(PNode& First)
    {
        int n, Data;
        cout<<"Введите количество элементов: ";
        cin>>n;
        for (int i=1; i<=n; i++)
        {
            cout<<"Введите элемент: ";
            cin>>Data;
            AddFirst(First, Data);
        }
    }
    //Обход и печать списка
    void PrintList(PNode First)
    {
        PNode p = First;
        while (p)
        {
            cout<<p->Data<<' ';
            p = p->Next;
        }
        cout<<endl;
    }
}

```

```

//Поиск со вставкой элемента в список
void FindIns(PNode& First, int x)
{
    PNode p=First, q=NULL;    //q отстает от p на один шаг
    //пока не конец списка или не найден искомый элемент...
    while (p && x != p->Data)
    {
        q = p;
        p = p->Next;    //идем по списку
    }
    //если найденный элемент - первый, добавляем его еще раз
    if (!q)
        AddFirst(First, x);
    else //а если не первый или нет такого элемента, то...
        AddAfter(q,x);
}

//Поиск с удалением элемента из списка
void FindDel(PNode& First, int x)
{
    PNode p=First, q=NULL;    //q отстает от p на один шаг
    //пока не дошли до конца списка
    while (p)
        //если нашли элемент...
        if (p->Data == x)
        {
            if (!q)    //...и он первый в списке
            {
                DelFirst(First);
                p = First; //p снова установить на начало
            }
            else //...а если он не первый в списке
            {
                DelAfter(q);
                p = q;
            }
        }
        else
        {
            q = p;
            p = p->Next;    //идем по списку
        }
    }
}

void main()
{
    PNode First = NULL;
    InputList(First);
    PrintList(First);
    cout<<"Введите значение для вставки: ";
    int Data;
    cin>>Data;
    FindIns(First, Data);
    PrintList(First);
    cout<<"Введите значение для удаления: ";
    cin>>Data;
    FindDel(First, Data);
    PrintList(First);
}

```

## 5.2 Работа со стеком

В данном примере реализованы основные операции работы со стеком целых чисел. В основной программе происходит заполнение стека, затем извлечение и вывод его элементов на экран.

```
#include <iostream.h>
struct Element          //описание структуры элемента стека
{
    int Item;
    Element* Next;
};
Element* TopPointer;     //указатель на вершину стека
int IsEmpty()            //проверка стека на пустоту
{
    return (!TopPointer);
}
void Get(int& Item)       //чтение вершины без ее извлечения из стека
{
    if (!TopPointer) cout<<"Стек пуст!\n";
    else Item = TopPointer->Item;
}
//изменение значения вершины без ее извлечения из стека
void Put(int Item)
{
    if (!TopPointer)      cout<<"Стек пуст!\n";
    else TopPointer->Item = Item;
}
//добавление элемента в стек
void Push(int Item)       //(аналогично вставке в начало списка)
{
    Element* temp; //временный указатель
    temp = new Element;
    temp->Item = Item;
    temp->Next = TopPointer;
    TopPointer = temp;
}
//извлечение элемента из стека
// (аналогично удалению из начала списка)
void Pop(int& Item)
{
    if (!TopPointer)      cout<<"Стек пуст!\n";
    else
    {
        Element* temp; //временный указатель
        Item = TopPointer->Item;
        temp = TopPointer->Next;
        delete TopPointer; //освобождение памяти
        TopPointer = temp;
    }
}
void main()
{
    int i;
    do          //заполнение стека
    {
        cout<<"Введите элемент (0 - конец ввода): ";
        cin>>i;
        if (i!=0) Push(i);
    }
```



```

    }
    while (i!=0); //до тех пор, пока не введен 0
    //печать стека
    while (!IsEmpty())
    {
        Pop(i);
        cout<<i<<' ';
    }
}

```

### 5.3 Работа с очередью

В данном примере реализованы основные операции работы с очередью, элементами которой могут быть целые числа. Операции по заполнению и выводу элементов очереди на экран реализованы в виде функций, которые затем вызываются в основной программе.

В примере широко используется механизм формальных параметров функций, где **QB**, **QE** – указатели на начало и конец очереди соответственно; **x** – значение для поля данных.

В главной функции программы объявлены идентификаторы **QBegin** и **QEnd**, которые и являются фактическими переменными-указателями на начало и конец очереди, именно они передаются в функции (если необходимо) при их вызове.

```

#include <iostream.h>
//предварительное описание структуры элемента очереди
struct Node;
//описание типа указателя на элемент очереди
typedef Node* PNode;
//структура элемента очереди
struct Node
{
    int Data;
    PNode Next;
};
//Инициализация очереди: создается первый элемент очереди
//(который также пока является и последним), на него
//устанавливаются указатели QE и QB
void Init(PNode& QB, PNode& QE, int x)
{
    QE = new Node;
    QE->Data = x;
    QE->Next = NULL;
    QB = QE;
}
//проверка очереди на пустоту
int Empty(PNode QB)
{
    return (!QB);
}
//Добавление элемента в конец очереди (см. рисунок 3.8)
void Insert(PNode& QE, int x)
{
    PNode p = new Node;

```

```

        p->Data = x;
        p->Next = NULL;
        QE->Next = p;
        QE = p;
    }
    //Удаление элемента из начала очереди
    //(аналогично удалению из начала списка)
    void Remove(PNode& QB, int& x)
    {
        if (Empty(QB)) cout<<"Очередь пуста!";
        else
        {
            PNode p;
            x = QB->Data;
            p = QB;
            QB = QB->Next;
            delete p;
        }
    }
    //заполнение очереди элементами
    void InputQueue(PNode& QB, PNode& QE)
    {
        int Data;
        cout<<"Введите элементы (0 - конец ввода): ";
        cin>>Data;
        Init(QB,QE,Data);
        do
        {
            cin>>Data;
            if (Data!=0)
                Insert(QE,Data);
        }
        while (Data!=0);
    }
    //Извлечение элементов из очереди и вывод их на экран
    void PrintQueue(PNode QB)
    {
        int Data;
        cout<<"Очередь:\n";
        while (!Empty(QB))
        {
            Remove(QB,Data);
            cout<<Data<<' ';
        }
        cout<<endl;
    }
    void main()
    {
        //фактические указатели на начало и конец очереди
        PNode QBegin, QEnd;
        InputQueue(QBegin,QEnd);
        PrintQueue(QBegin);
    }

```

## 6 Варианты заданий для самостоятельного решения

1. Дан список, содержащий фамилии студентов и средний балл сессии каждого студента. Создать новый список, в который войдут студенты, средний

балл у которых не меньше "4", а в старом останутся все остальные.

2. Дан список, содержащий числовые данные. Сформировать два новых списка таким образом, чтобы половина элементов исходного списка попала в первый новый список (1, 3, 5, ...), а вторая половина – во второй новый (2, 4, 6, ...).

3. Дан список, содержащий перечень товаров различных фирм. Из элементов этого списка создать новый список, который будет содержать товары, изготовленные фирмой SONY.

4. Дан список, содержащий информацию о клиентах пункта проката автомобилей: ФИО и марка машины. Во второй список записать данные только тех людей, кто ездит на "Audi".

5. Сформировать очередь из всех звонких согласных некоторого текста в обратном порядке их встречаемости. Вывести очередь на экран.

6. Сформировать стек – пирамиду слов: на вершине – самое длинное.

7. Дан массив данных о работниках фирмы: ФИО и дата поступления на работу (месяц, год). Сформировать стек, в который попадут данные тех из них, кто на сегодняшний день проработал уже не менее 5 лет.

8. Дан список, содержащий числовые данные. Написать программу передвижения элемента на  $n$  позиций.

9. Дан список, содержащий 10 фамилий игроков футбольной команды. Разбить игроков на 2 группы по 5 человек. Во вторую группу попадает каждый 2-й человек.

10. Записать в стек все заглавные буквы некоторого текста. Вывести стек на экран.

11. Даны два списка. Создать новый список, который будет содержать элементы, общие для двух исходных списков.

12. Точку в пространстве можно представить тремя координатами  $X$ ,  $Y$ ,  $Z$ . Дан массив из  $N$  точек. Сформировать стек тех точек, которые лежат справа от плоскости  $X=6$ .

13. Для студентов КГУ, проживающих в общежитии, известно ФИО, номер этажа и номер сотового телефона. Студенту на  $k$ -ом этаже стало плохо. Сформировать стек из тех студентов, которые могли бы быть полезны в данной ситуации.

14. Даны два списка, содержащие фамилии студентов 2-х групп. Перевести  $L$  студентов из 1-й группы во вторую. Число пересчета -  $K$  (т.е. каждый  $K$ -й студент из 1-й группы переводится во 2-ю).

15. Точка плоскости может быть представлена двумя координатами  $X$  и  $Y$ . Дан массив, содержащий  $N$  точек. Сформировать очередь из точек, лежащих слева от оси ординат.

16. Дан текст, состоящий не менее чем из семи слов. Сформировать стек, в который попадут слова, содержащие в себе повторяющиеся буквы (молоко).

17. Дан список, содержащий перечень товаров, производимых концернами

SHARP и LG. Создать списки товаров, выпускаемых как одной, так и другой фирмой.

18. Даны два массива, содержащие перечень товаров и фамилии покупателей соответственно. Каждый N-й покупатель покупает M-й товар. Вывести список покупок.

19. Дан текст, состоящий не менее чем из 7 слов. Сформировать стек всех слов, длина которых меньше либо равна 5.

20. Даны два списка, содержащие фамилии футболистов основного состава команды и запасного. Произвести K замен.

21. Сформировать очередь из чисел некоторого диапазона [k1..k2]. В очередь попадают те числа, которые состоят из одинаковых цифр (11, 22, 33...).

22. Для каждого из N студентов группы известны ФИО и оценки по пяти дисциплинам. Сформировать очередь, в которую попадут те студенты, у которых количество положительных оценок больше, чем отрицательных.

23. Найти минимальный элемент в списке и вставить после него число, вводимое с клавиатуры.

24. Дан список слов. Заменить слова, начинающиеся с гласной на эти же слова, но записанные прописными буквами. Вывести список.

25. Дан текст, состоящий не менее чем из пяти слов. Сформировать стек из тех слов, в которых присутствует буква "Е".

26. Дан список, содержащий информацию об учениках некоторой школы. Сформировать новый список, который будет содержать сведения об учениках только десятых классов.

27. Дан массив, содержащий сведения о книгах: название, жанр, автор. Сформировать очередь, в которой книги будут отсортированы по жанру, т.е. сначала, например, вся классика, затем фантастика и т.д.

28. Для каждого работника известны ФИО, район, адрес (дом, улица, квартира). Сформировать очередь тех рабочих, которые живут в центре выбранного района (за центр считать улицы, номера домов которых от 50 до 150).

29. Дан список символов. Определить, чего в нем больше: букв или цифр. Добавить недостающие элементы (с помощью генератора случайных чисел) в конец списка, чтобы количество тех и других сравнялось.

30. Определить количество нулей в числовом списке. Добавить это значение (количества) в начало и конец списка.

31. Дан список чисел, отсортированных по возрастанию. Найти среднее арифметическое его элементов и вставить его в список, не нарушив упорядочения.

32. Дан список, содержащий данные о работниках фирмы: ФИО и адрес (улица, дом, квартира). Сформировать второй список из тех работников, которые живут на улице Красной, при этом удалить эти сведения из исходного списка.

## **Лабораторная работа №4. Разработка приложений с использованием динамических массивов**

### **1 Цель работы**

Получить практические навыки разработки программ с использованием динамических массивов и матриц. Изучить приемы передачи динамических массивов в качестве формальных параметров функций.

### **2 Порядок выполнения работы**

Получить задание для выполнения лабораторной работы согласно своему варианту. Разработать и отладить программу. Составить отчет о лабораторной работе и защитить его у преподавателя.

### **3 Содержание отчета**

- наименование и цель работы;
- задание на лабораторную работу согласно варианту;
- схема алгоритма, текст программы на алгоритмическом языке;
- результаты работы программы.

### **4 Краткая теория**

Традиционный способ работы с массивами предполагает, что массив объявляется с фиксированным количеством элементов, это количество становится известным программе на этапе ее компиляции и не изменяется до конца ее выполнения. Это статические массивы.

Однако во многих случаях заранее не известно, какое количество данных требуется обрабатывать программе, либо размер обрабатываемых данных часто меняется в процессе ее работы. Объявление статических массивов в этом случае приводит к тому, что значительная часть памяти тратится впустую.

Для решения данной проблемы и применяются динамические массивы, для работы с которыми используются указатели.

#### **4.1 Создание и удаление динамических массивов**

Как только память для массива выделена, имя массива воспринимается как указатель того типа, к которому отнесены элементы массива. Значением имени массива является адрес первого элемента массива. Таким образом, для любого массива соблюдается равенство:

`имя_массива == &имя_массива == &имя_массива[0]`

Для создания динамического массива служит операция **new**, которой

указывается тип элементов массива и их желаемое количество в квадратных скобках. При этом количество элементов может быть обычной переменной (не константой), вводимой, например, с клавиатуры. Перед этим сам массив описывается как указатель, тип которого соответствует типу элементов массива. Пример:

```
int* mas;  
mas = new int[10];
```

или: `int* mas = new int[10];`

Операция **new** возвращает адрес первого элемента массива, и это значение присваивается указателю **mas**.

Когда для создания массива используется операция **new**, следует употребить другую форму операции **delete**, которая указывает, что освобождается именно массив: `delete[] mas;`

Квадратные скобки свидетельствуют о том, что необходимо освободить целый массив, а не только элемент, на который указывает указатель (Ранние версии языка C++ могут не распознавать запись с квадратными скобками).

Так как имя массива есть указатель на его первый элемент, то к нему приемлемы все правила адресной арифметики, связанные с указателями. Если описан массив `int z[3]`, то `*z` аналогично `z[0]`, `*(z+1)` аналогично `z[1]` и т.д. А, учитывая коммутативность операции сложения, получаем, что:

```
z[1] == *(z+1) == *(1+z) == 1[z]
```

Т.е. все четыре способа получения доступа к элементам массива эквивалентны.

Матрица, в соответствии с синтаксисом языка, есть массив массивов, т.е. массив, элементами которого служат массивы. Поэтому выделение и освобождение памяти для динамических матриц должно выполняться отдельно для матриц целиком, которые являются массивом строк, и отдельно для каждой строки, которые состоят из массивов элементов:

```
int m=3,n=5;  
float** matr;//указатель для массива указателей  
matr=new float *[m];  
for (int i=0;i<m;i++)  
    matr[i]=new float[n];  
...  
for (i=0;i<m;i++)  
    delete matr[i];  
delete []matr;
```

По аналогии с массивами, можно получать доступ к элементам матриц через указатели:

```
matr[i][j] == (*(matr+i)+j) == *(matr[i]+j)
```

## 4.2 Передача матриц в качестве формальных параметров функций

Особенность и в некотором смысле недостаток языка C++ – несамоопределенность многомерных массивов, под которой понимается невозможность по имени массива (по указателю на массив) определить его размерность и размеры по каждому измерению. Эта особенность затрудняет использование матриц в качестве параметров функций.

Очевидное и неверное решение в этом случае – определить заголовок функции следующим образом:

```
void func(int x[][],int n) ...//неверно!!!
```

Здесь  $n$  – предполагаемый порядок квадратной матрицы; `int x[][]` – попытка определить матрицу с заранее неизвестными размерами. На такую попытку компилятор отвечает сообщением об ошибке.

При описании матрицы (или спецификации матрицы-параметра) неопределенным может быть только первый (левый) размер. Таким образом, примитивнейшее разрешение проблемы выглядит так:

```
void func(int x[][3],int n) ...
```

Примитивность данного решения состоит в том, что матрица-параметр передается с фиксированным вторым параметром, т.е. она может быть только с размерами 3 на 3.

Для избавления от фиксации количества элементов многомерного массива при передаче параметров используются следующие пути:

- подмена многомерного массива одномерным и имитация внутри функции доступа к многомерному массиву;
- использование вспомогательных массивов указателей на массив.

Однако эти способы достаточно громоздки и трудоемки. Поэтому самым оптимальным методом передачи матриц в качестве формальных параметров функций является использование указателей и динамических массивов: `void func(int** x,int m,int n) ...`

Сама матрица может формироваться в главной функции программы как совокупность одномерных динамических массивов строк матрицы и динамического массива указателей на эти массивы-строки.

## 5 Пример программы

Создать матрицу случайных чисел размером  $m \times n$ . Описать функцию подсчета нулевых элементов каждой строки матрицы и записи этих значений в одномерный массив.

В приведенной ниже программе продемонстрирована возможность передачи динамических массивов и матриц в качестве параметров функций, а также возврата их в качестве результата. Динамическое выделение памяти осуществляется внутри функций (хотя может происходить и внутри функции `main`), а освобождение памяти – в функции `main`.

```

#include <iostream.h>
#include <stdlib.h>
#include <time.h>
//формирование матрицы
int** MakeMatr(int m,int n)
{
    int** t; //создаем вспомогательную матрицу,
    t = new int*[m]; //выделяем ей память,
    for (int i=0;i<m;i++)
    {
        t[i] = new int[n];
        for (int j=0;j<n;j++)//заполняем случайными числами
            t[i][j] = rand()%10-5;
    }
    return t; //и возвращаем ее в главную функцию
}
//печать матрицы
void PrintMatr(int m,int n,int** matr)
{
    for (int i=0;i<m;i++)
    {
        for (int j=0;j<n;j++)
        {
            cout.width(4);
            cout<<matr[i][j];
        }
        cout<<endl;
    }
}
//формирование массива-результата
int* MakeArray(int m,int n,int** matr)
{
    int* arr = new int[m]; //вспомогательный массив
    for (int i=0;i<m;i++)
    {
        arr[i] = 0; //сначала каждый элемент массива = 0
        for (int j=0;j<n;j++)
            //ищем нули в очередной строке матрицы
            if (matr[i][j]==0)
                arr[i]++; //наращиваем значение массива
    }
    return arr; //возвращаем сформированный массив
}
void main()
{
    int M,N;
    cout<<"Введите размерности матрицы: ";
    cin>>M>>N;
    int** A; //описание матрицы
    A = MakeMatr(M,N);
    PrintMatr(M,N,A);
    int* B = MakeArray(M,N,A);
    cout<<"Результат:\n";
    for (int i=0;i<M;i++) cout<<B[i]<<' ';
    cout<<endl;
    for (i=0;i<M;i++) //освобождение памяти
        delete[] A[i];
}

```



```

delete[] A;
delete[] B;
}

```

## 6 Задания для самостоятельной работы

Во всех вариантах необходимо разработать программу, состоящую из нескольких функций. Массивы или матрицы необходимо передавать в функции в качестве параметров или возвращать в качестве результата. Массивы и матрицы создавать и удалять динамически.

1. Матрицу размером  $m \times n$  ( $m, n$  – четные) разбить на 2 матрицы размером  $m/2 \times n/2$ . Элементы из исходной матрицы брать в шахматном порядке.

2. Две матрицы размером  $m \times n$  объединить в одну размером  $2m \times 2n$ . Элементы в новую матрицу заносить в шахматном порядке.

3. Отсортировать столбцы матрицы по возрастанию их первых элементов.

4. Отсортировать строки матрицы по убыванию их среднеарифметических значений.

5. Даны две квадратные матрицы. Напечатать ту из них, которая имеет минимальный «след» (т.е. сумму элементов главной диагонали). Использовать функцию для нахождения следа матрицы и функцию печати матрицы.

6. Дана прямоугольная вещественная матрица. Проверить, упорядочены ли по неубыванию суммы элементов строк этой матрицы. Использовать функцию для нахождения суммы элементов строки матрицы.

7. Дана целочисленная матрица  $A(m, n)$ . Заменить нулями элементы матрицы, стоящие на пересечении строк и столбцов, содержащих минимальный и максимальный элементы.

8. Сформировать матрицу  $B(m, n)$ , элементами которой являются случайные числа, равномерно распределенные в интервале  $[-5, 5]$ . Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент матрицы оказался в правом нижнем углу.

9. Дана матрица  $N \times N$  ( $N$  – четное), в которой каждый элемент встречается 4 раза. Развернуть матрицу по строкам, удалить все повторяющиеся элементы и свернуть массив обратно по столбцам в матрицу  $[N/2 \times N/2]$ .

10. Найти седловые точки матрицы (седловой точкой называется элемент, являющийся минимальным в строке и максимальным в столбце).

11. В двумерном массиве  $X(m, n)$  все числа различны. В каждой строке находится минимальный элемент, затем среди этих чисел находится максимальное. Напечатать индексы (номер строки и номер столбца) этого элемента.

12. Даны целые числа  $a$  и  $b$  ( $a < b$ ). Сформировать матрицу  $X(10, 10)$ , элементами которой являются случайные числа, равномерно распределенные на отрезке  $[a, b]$ . Найти в матрице строку с минимальным элементом и поме-

нять ее местами с первой строкой.

13. Найти наименьшее среди тех элементов матрицы  $X$ , которые не являются элементами одномерного массива  $Y$ .

14. Дана целочисленная матрица  $A(N, M)$  ( $N, M \leq 10$ ). Построить по ней целочисленный массив  $B$ , присвоив его  $k$ -му элементу значение 1, если  $k$ -я строка матрицы  $A$  симметрична (т.е. первый элемент равен последнему, второй - предпоследнему и т.д.), и 0 – в противном случае.

15. В данной квадратичной целочисленной матрице размера  $10 \times 10$  определить количество столбцов, у которых полусумма минимального и максимального элемента столбца превышает среднее арифметическое значение всех элементов матрицы.

16. Дана матрица  $W(10, 10)$ . Составить программу вычисления средне-квадратического значения элементов и максимального отклонения от этого значения для элементов на главной диагонали.

17. Дана матрица  $A(10, 10)$ . Найти среднеарифметическое каждого из столбцов и количество столбцов, у которых среднеарифметическое больше элемента побочной диагонали.

18. В заданной матрице поменять местами ее минимальный и максимальный элементы.

19. Перемножить две матрицы:  $A(k, m)$  и  $B(m, n)$ .

20. Дана целочисленная матрица  $A(N, N)$ . Построить по ней целочисленный массив  $B$ , присвоив его  $k$ -му элементу значение 1, если в  $k$ -й строке матрицы  $A$  элементы главной и побочной диагоналей совпадают, и 0 – в противном случае.

21. Дана матрица размером  $m \times n$ , заполненная случайным образом. Определить, есть ли в данной матрице строка, в которой ровно два нечетных элемента.

22. В каждой строке заполненной случайным образом матрицы размером  $m \times n$  поменять местами последний элемент и минимальный по модулю.

23. Дан двумерный массив размерностью  $m \times n$ , заполненный целыми числами с клавиатуры. Сформировать одномерный массив, каждый элемент которого равен количеству отрицательных элементов, кратных 3, соответствующей строки.

24. Определить, есть ли в данной матрице столбец, состоящий только из четных элементов.

25. Дана матрица, заполненная случайным образом. Сформировать одномерный массив, элементами которого могут быть 1 или 0 в зависимости от следующего условия: 1 – сумма элементов строки больше вводимого числа  $X$ , 0 – иначе.

26. Даны две матрицы одинаковых размерностей. Сформировать одномерный массив, элементами которого могут быть 1 или 0 в зависимости от следующего условия: 1 – сумма элементов столбца первой матрицы больше

аналогичной суммы для второй матрицы, 0 – иначе.

27. Даны две квадратные матрицы одинаковых размерностей. Сформировать одномерный массив, каждый элемент которого равен максимальному из соответствующих элементов главных диагоналей матриц.

28. Дана матрица  $A(m,n)$ . Сформировать одномерный массив  $B(m)$ , элементами которого являются суммы элементов  $i$ -ой строки, начиная с первого нулевого элемента и до конца строки матрицы (0 – если нулевой элемент отсутствует).

29. Дана матрица  $K(m,n)$ . Найти позицию первого элемента, встречающегося в матрице ровно два раза.

30. Дана матрица  $K(m,n)$ . Сформировать одномерный массив  $L(n)$ , элементами которого являются суммы элементов  $j$ -ого столбца, начиная с первого положительного элемента и до второго положительного элемента столбца матрицы (0 – если положительных элементов в столбце нет).

31. Дан двумерный массив размером  $m \times n$ , заполненный случайными числами. Определить, есть ли в данном массиве строка, содержащая больше четных элементов, чем нечетных.

32. Даны две матрицы размером  $m \times n$ . Поменять в них местами строки, в которых находится первый нулевой элемент. Если в какой-то из матриц нулевых элементов нет, выдать соответствующее сообщение.

# Лабораторная работа №5. Разработка приложений с использованием файлового ввода/вывода

## 1 Цель работы

Получить практические навыки разработки программ с использованием файлового ввода/вывода. Изучить приемы работы с текстовыми и двоичными файлами.

## 2 Порядок выполнения работы

Получить задание для выполнения лабораторной работы согласно своему варианту. Разработать и отладить программу. Составить отчет о лабораторной работе и защитить его у преподавателя.

## 3 Содержание отчета

- наименование и цель работы;
- задание на лабораторную работу согласно варианту;
- схема алгоритма, текст программы на алгоритмическом языке;
- результаты работы программы.

## 4 Краткая теория

При решении многих задач возникает проблема хранения и обработки достаточно большого объема данных. В таких случаях широко используют внешние устройства для записи и чтения информации. Связь с внешними источниками, приемниками и носителями информации в C++ осуществляется только с помощью файлов.

Традиционно под файлом понимается поименованная совокупность данных на внешнем носителе, однако в C++ этот термин трактуется более широко. **Файлом** здесь считается также **любое внешнее устройство**, по своему назначению являющееся источником или приемником информации, например, клавиатура, принтер, диск и т.д. Такое устройство принято называть логическим, поскольку учитывается только его главная функция, а не физические характеристики.

До начала операции ввода-вывода конкретному внешнему файлу должна быть поставлена в соответствие специальная переменная в программе.

### 4.1 Файловый ввод/вывод с помощью потоков

В языке C++ файл рассматривается как поток (stream), представляющий собой последовательность считываемых или записываемых байтов. При этом поток «не знает», что и в какой последовательности в него записано. Расшифровка смысла записанных последовательностей байтов лежит на про-

грамме.

Язык C++ работает с файловым вводом/выводом во многом так же, как он это делает со стандартным вводом/выводом. Для работы с файлами необходимо подключить заголовочный файл **fstream.h**. При этом автоматически включается и файл **iostream.h**, поэтому его не нужно подключать явно.

В C++ определены три класса файлового ввода/вывода:

- **ifstream** – входные файлы для чтения;
- **ofstream** – выходные файлы для записи;
- **fstream** – файлы для чтения и записи.

Предположим, что программе необходимо записать информацию в файл. Для этого необходимо сначала создать переменную (объект) типа **ofstream** для управления потоком вывода, например:

```
ofstream fout;
```

Имя объекта может быть любым допустимым именем C++.

Затем необходимо поставить в соответствие объекту определенный файл. Это можно сделать с помощью функции **open()**:

```
fout.open("test.dat");
```

Два этих этапа (создание объекта и связывание его с файлом) можно объединить в одном операторе:

```
ofstream fout("test.dat");
```

После этого объект **fout** можно использовать точно так же, как и **cout**:

```
int i=1,j=25;  
double a=25e6;  
char s[10]="строка";  
fout<<i<<' '<<j<<' '<<a<<' '<<s<<endl;
```

В этом коде с помощью одного оператора осуществляется запись в файл в текстовом виде данных разного типа: двух целочисленных переменных, действительного числа и строки. После того, как файл закроется, в нем будет записан текст «1 25 2.5e+07 строка».

Открытие файла таким способом позволяет создать новый файл, если файла с таким именем не существует. Если же такой файл уже есть, то до открытия для вывода этот файл урезается до нулевого размера и информация начинает выводиться в пустой файл.

Требования к чтению файла очень похожи на требования к записи:

1. Создать переменную (объект) типа **ifstream** для управления потоком ввода.

2. Поставить этот объект в соответствие определенному файлу.

3. Использовать объект так же, как **cin**.

Например:

```
ifstream fin;  
fin.open("test.txt");  
//или ifstream fin("test.txt");  
int x;
```

```
float f;
char st[20];
fin>>x>>f; //чтение из файла числовых данных
fin.getline(st,20); //чтение строки
```

При установлении связи между файлом и потоком могут возникнуть ошибки (например, при открытии файла для чтения нужного файла нет на диске). В этом случае значение объекта потока равно 0, и этот факт можно использовать для отслеживания таких ошибок:

```
if (!fin)
{ cout<<"Ошибка!!!\n";
  return;
}
```

Связь с файлом прекращается автоматически, если истекает срок действия объектов потока ввода/вывода, например, по окончании выполнения программы. Кроме того, связь с файлом можно прервать явно, воспользовавшись функцией `close()`:

```
fout.close();
fin.close();
```

При прерывании такой связи поток не ликвидируется, он просто отключается от файла. Этот поток можно вновь соединить с тем же самым или с другим файлом.

Если необходимо одновременно работать с несколькими открытыми файлами, то для каждого из них нужно создавать отдельный поток.

## 4.2 Дополнительные функции файлового ввода/вывода

Классы потоков ввода/вывода содержат множество полезных функций для работы с файлами. Ниже описаны некоторые из них.

```
filebuf* rdbuf();
```

Возвращает указатель на буфер, связанный с потоком.

```
int is_open();
```

Осуществляет проверку того, было ли успешным открытие файла. Возвращает нулевое значение в случае ошибки. Пример:

```
if (!fin.rdbuf()->is_open())
  cerr<<"Не удалось открыть файл...\n";
int eof();
```

Возвращает ненулевое значение, если достигнут конец файла:

```
while (!fin.eof()) fin>>x;
void clear(int=0);
```

Устанавливает состояние потока в ноль. Эту функцию необходимо вызывать, если работу с потоком нужно продолжать после возникновения таких ситуаций, как достижение конца файла, ошибка при обмене с потоком и т.п.

```
ostream& put(char);
```

Выводит в поток один символ. Допускает сцепленный вызов:

```
fout.put('A').put('\n');
ostream& write(const signed char*,int n);
```

```
ostream& write(const unsigned char*,int n);
```

Выводит в файл из символьного массива, на который указывает первый параметр, число символов, указанных вторым параметром. Например, оператор

`fout.write(s,5);` записывает в поток `fout` 5 символов из массива `s`. Причем эти символы никак не обрабатываются, а просто выводятся в качестве сырых байтов данных. Среди этих символов, например, может встретиться в любом месте нулевой символ, но он не будет рассматриваться как признак конца строки.

```
istream& read(signed char*,int);  
istream& read(unsigned char*,int);
```

Аналогичный метод для чтения данных в символьный массив также без всякой обработки.

```
int gcount();
```

Используется совместно с предыдущей функцией. Возвращает количество символов, действительно прочитанных последней операцией ввода.

```
int get();
```

Читает одиночный символ из указанного потока (даже если это символ-разделитель) и возвращает этот символ в качестве значения вызова функции. Если в потоке встретился признак конца файла, возвращает значение `EOF`. Данный вариант функции удобно использовать для поиска в файле какого-то ключевого символа. Например, цикл поиска в файле символа «\$» можно организовать следующим образом:

```
char c;  
while ((c=fin.get())!=EOF)  
    if (c=='$') break;  
if (c=='$')  
    ...  
istream& get(unsigned char&);  
istream& get(signed char&);
```

Вводит очередной символ из входного потока (даже если это символ-разделитель) и сохраняет его в символьном аргументе. Этот вариант функции возвращает 0, когда встречается признак конца файла; в остальных случаях возвращается ссылка на тот поток, для которого вызывалась функция. Предыдущий пример с использованием данного варианта функции `get()` можно было бы переписать так:

```
while (fin.get(c))  
    ...  
istream& get(signed char*,int,char='\n');  
istream& get(unsigned char*,int,char='\n');
```

Третий вариант функции `get()` принимает три параметра: символьный массив, максимальное число элементов и символ-ограничитель (по умолчанию символ перевода строки `'\n'`). Символы читаются из входного потока до тех пор, пока не достигается число символов, на 1 меньше указанного

максимального числа, или пока не считывается ограничитель. Затем для завершения введенной строки в символьный массив помещается нулевой символ. Ограничитель в символьный массив не помещается, а остается во входном потоке (он будет следующим считываемым символом). Таким образом, результатом второго подряд использования функции `get()` явится пустая строка, если только ограничитель не удалить из входного потока.

```
istream& getline(signed char*,int,char='\n');
istream& getline(unsigned char*,int,char='\n');
```

Действует подобно предыдущему (третьему) варианту функции `get()`, но удаляет символ-ограничитель из потока (т.е. читает этот символ и отбрасывает его); этот символ не сохраняется в символьном массиве. С помощью `getline()` можно следующим образом записать цикл чтения файла по строкам:

```
while (!fin.eof())
{   fin.getline(s,80);   ...   //обработка строки
}
```

### 4.3 Режимы файлов

Режим файла описывает, как используется файл: для чтения, для записи, для добавления и т.д. Когда поток связывается с файлом, а также при инициализации файлового потокового объекта именем файла или при работе с функцией `open()`, можно использовать и второй аргумент, задающий режим файла:

```
ifstream fin("file1.txt",ios::in);
ofstream fout;
fout.open("file2.dat",ios::app);
```

В C++ определено несколько констант, которыми можно воспользоваться для указания режима файла. Список этих констант и их значений приведен в таблице 5.1.

Таблица 5.1—Константы режимов файла

Константа	Значение
<code>ios::in</code>	Открыть файл для чтения
<code>ios::out</code>	Открыть файл для записи
<code>ios::ate</code>	Переместить указатель в конец файла после открытия
<code>ios::app</code>	Добавить информацию к концу файла
<code>ios::trunc</code>	Урезать файл, если он существует
<code>ios::nocreate</code>	Не открывать новый файл (для несуществующего файла функция <code>open()</code> выдаст ошибку)
<code>ios::noreplace</code>	Не открывать существующий файл (для существующего выходного файла, не имеющего режимов <code>ate</code> или <code>app</code> , выдаст ошибку)
<code>ios::binary</code>	Двоичный файл



Если при связывании потока с файлом необходимо указать одновременно несколько режимов, их следует перечислять через | (операция «побитовое ИЛИ»). Например, чтобы открыть файл для дозаписи данных, нужно использовать следующий оператор:

```
ofstream fout("myfile.dat",ios::out|ios::app);
```

По умолчанию, при связывании файла с потоком ввода используется константа `ios::in` (открыть для чтения), а при связывании с потоком вывода – `ios::out|ios::trunc` (открыть файл для записи и стереть его содержимое).

#### 4.4 Двоичные файлы

Когда данные сохраняются в файле, их можно сохранить в текстовой форме или двоичном формате. Текстовая форма означает, что все данные сохраняются как текст, даже числа. Например, сохранение значения - 2.324216e+07 в текстовой форме означает сохранение 13 символов, из которых состоит данное число. Двоичный формат означает, что число сохраняется во внутреннем представлении, т.е. вместо символов сохраняется 64-разрядное представление числа типа **double**. Для символа двоичное представление совпадает с его текстовым – двоичным представлением ASCII-кода (или его эквивалента) символа. Однако для чисел двоичное представление очень сильно отличается от их текстового представления (рис. 5.1).

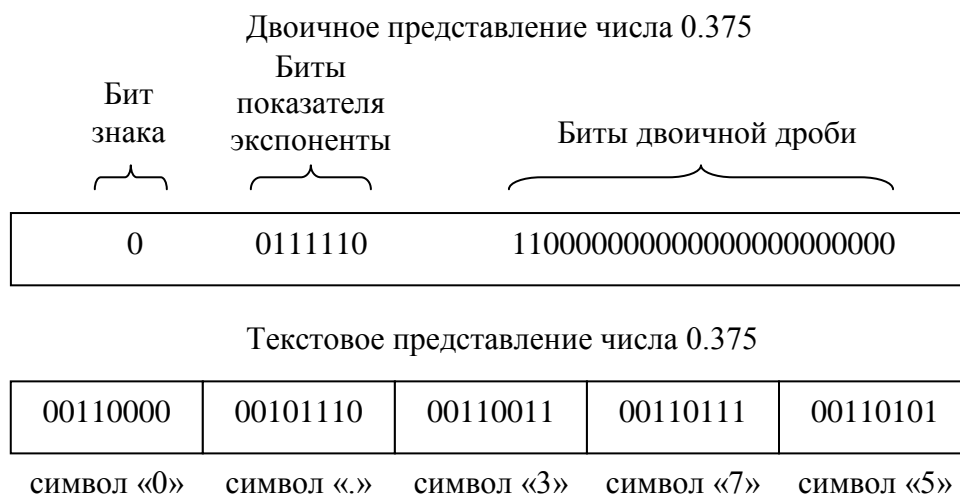


Рисунок 5.1 – Двоичное и текстовое представление числа с плавающей точкой

В двоичном формате числа сохраняются более точно, поскольку он позволяет сохранить точное внутреннее представление числа. Не происходит ошибок преобразования и округления. Сохранение данных в двоичном формате может происходить быстрее, поскольку при этом не происходит преобразования и данные можно сохранять большими блоками. Кроме того, двоичный формат обычно занимает меньше места.

Для работы с двоичными файлами используется режим `ios::binary`.

Чтобы сохранить данные в двоичной форме, можно воспользоваться функцией `write()`, которая копирует определенное число байтов из памяти в файл. Ранее указывалось, что эта функция используется для копирования текста, но она может копировать любой тип данных байт в байт, не производя преобразования. Например, если ей передать адрес переменной типа `long` и указать скопировать 4 байта, то данная функция скопирует 4 байта, «дословно» передав значение типа `long` и не производя его преобразования. Единственное неудобство заключается в том, что адрес переменной необходимо преобразовать к типу указатель-на-`char`. Чтобы узнать размер переменной в байтах, можно воспользоваться операцией `sizeof`. Пример:

```
long x=10L;
ofstream fout("file1.dat",ios::out|ios::binary);
fout.write((char*)&x,sizeof(x));
```

Чтобы прочесть информацию из двоичного файла, нужно использовать соответствующую функцию `read()` с объектом типа `ifstream`:

```
ifstream fin("file1.dat",ios::in|ios::binary);
fin.read((char*)&x,sizeof(x));
```

Данный блок кода копирует количество байтов `sizeof(x)` из файла в переменную `x`.

Подобным образом можно сохранять в файлы и читать из них и переменные более сложных типов, например, структуры.

## 4.5 Произвольный доступ к файлам

Произвольный доступ к файлам предоставляет возможность переместиться в любое место файла сразу, вместо последовательного передвижения по нему. Подход с произвольным доступом часто используется при обработке файлов баз данных. Этот подход проще реализовать, если файл состоит из набора записей одинакового типа (или хотя бы размера).

Для реализации «передвижения» по файлу существуют специальные функции: `seekg()` передвигает указатель ввода, `seekp()` – указатель вывода в определенную точку файла. Соответственно, `seekg()` можно использовать с объектом типа `ifstream`, а `seekp()` – с объектом типа `ofstream`.

Функция передвижения указателя ввода имеет следующие прототипы:

```
istream& seekg(long);
istream& seekg(long, seek_dir);
```

Первый прототип устанавливает указатель чтения входного потока на абсолютную позицию, заданную параметром. Эта позиция отстоит от начала файла на указанное количество байтов, т.е. значение позиции можно трактовать как смещение от начала файла, где первый байт имеет индекс 0. Поэтому оператор

```
fin.seekg(112);
```

передвигает файловый указатель на 112-й байт, который является реальным 113-м байтом файла.

Второй прототип перемещает указатель чтения входного потока на число байтов, заданное первым параметром. Второй параметр задает точку отсчета: значение `ios::beg` означает, что смещение отсчитывается от начала файла, `ios::cur` – от текущей позиции, `ios::end` – от конца файла.

Примеры вызова функции:

```
//30 байтов от начала файла
fin.seekg(30,ios::beg);
//один байт назад от текущей позиции
fin.seekg(-1,ios::cur);
//переход к концу файла
fin.seekg(0,ios::end);
```

Функция передвижения указателя вывода имеет следующие прототипы:

```
ostream& seekp(long);
ostream& seekp(long,seek_dir);
```

Принципы работы этой функции полностью идентичны предыдущей, за исключением того, что она работает с объектом потока вывода:

```
fout.seekp(20,ios::beg);
```

Если необходимо проверить текущую позицию файлового указателя, можно воспользоваться функцией `tellg()` для потока ввода и `tellp()` – для потока вывода. Каждый из них возвращает значение типа `long`, представляющее собой текущее смещение указателя от начала файла в байтах. Когда создается объект типа `fstream`, входной и выходной указатели передвигаются одновременно, поэтому в таком случае функции `tellg()` и `tellp()` возвращают одинаковое значение. Но если используется объект типа `ifstream` для управления потоком ввода и объект типа `ofstream` для управления потоком вывода, входной и выходной указатели передвигаются независимо друг от друга, и функции `tellg()` и `tellp()` могут возвращать разные значения.

## 5 Примеры работы с файлами

**5.1** Пример работы с текстовым файлом. Дан текстовый файл, нужно вычислить его длину в байтах и сохранить ее в конце файла. Кроме того, каждый раз при встрече символа новой строки требуется сохранить текущее смещение в конце файла. Например, если исходный файл имеет вид

```
abcd
efg
hi
j
```

то программа должна модифицировать этот файл следующим образом:

```
abcd
efg
hi
```

```
j
5 9 12 14 24
```

```
#include <fstream.h>
void main()
{    // открыть файл для ввода и дозаписи
    fstream inOut("copy.out",ios::in|ios::app);
    int cnt=0; // счетчик байтов
    char ch;
    //ставим указатель на начало файла
    inOut.seekg(0);
    //считываем данные из файла посимвольно
    while (inOut.get(ch))
    {    cout.put(ch);    //скопировать на экран
        cnt++;
        if (ch=='\n')    //если конец строки
        {    // запомнить текущую позицию
            int mark = inOut.tellg();
            inOut<<cnt<<' ';
            inOut.seekg(mark); // восстановить позицию
        }
    }
    inOut.clear(); //обнулить флаги состояния
    //вывести окончательное значение счетчика байтов
    inOut<<cnt<<endl;
    cout<<"[ "<<cnt<<" ]\n";
}
```

**5.2** Пример работы с двоичным файлом. Компонентами файла являются массивы из 10 элементов. Программа последовательно считывает из файла эти массивы, и определяет для массивов с четными номерами максимальное значение, с нечетными – минимальное.

```
#include <fstream.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
//количество элементов в массивах
const N=10;
//функция нахождения максимума и минимума в массиве
int MaxMin(int a[],char* Choice)
{    int M=a[0],i;
    if (strcmp(Choice,"Max")==0)
        for (i=1;i<N;i++)
            if (a[i]>M)    M=a[i];
    if (strcmp(Choice,"Min")==0)
        for (i=1;i<N;i++)
            if (a[i]<M)    M=a[i];
    return M;
}
```

```

void main()
{
    srand((unsigned)time(NULL));
    //открываем двоичный файл для записи
    ofstream fo("maxmin.dat",ios::out|ios::binary);
    int rec;
    int A[N];
    cout<<"Input the number of records: ";
    cin>>rec;
    //заполняем файл данными - массивами
    for (int j=1;j<=rec;j++)
    {
        for (int i=0;i<N;i++)  A[i] = rand()%100-rand()%50;
        fo.write((char*)A,sizeof(A));
    }
    //закрываем файл
    fo.close();
    //и открываем его для чтения
    ifstream fi("maxmin.dat",ios::in|ios::binary);
    //пока не конец файла, читаем очередной массив
    while (fi.read((char*)A,sizeof(A)))
    {
        //определяем номер прочитанного массива
        int pos = (fi.tellg()-1)/sizeof(A);
        //если номер четный, то выводим max, иначе min
        if (pos % 2 == 0)
            cout<<"Max of array "<<pos<<" = "<<
                MaxMin(A,"Max")<<endl;
        else
            cout<<"Min of array "<<pos<<" = "<<
                MaxMin(A,"Min")<<endl;
    }
    //очищаем поток ввода
    fi.clear();
    //перемещаем указатель на начало файла
    fi.seekg(0,ios::beg);
    //бесконечный цикл чтения из файла
    while (1)
    {
        //считать очередной массив
        fi.read((char*)A,sizeof(A));
        //если конец файла, то выходим из цикла
        if (fi.eof()) break;
        cout<<"Array "<<fi.tellg()/sizeof(A)-1<<": ";
        //печатаем массив
        for (int i=0;i<N;i++)    cout<<A[i]<<' ';
        cout<<endl;
    }
}

```

## 6 Задания для самостоятельной работы

### 6.1 Текстовые файлы

Примечание. Исходные текстовые файлы можно создавать с использованием

редакторов IDE BC++, FarManager, WinCommander (с сохранением файла в формате DOS). При использовании редакторов ОС Windows 9X/2000/XP (например, «Блокнот») возможно несовпадение кодировок кириллицы.

1. В файле data.txt записан некоторый текст. Написать программу, которая подсчитывает количество букв латинского и русского алфавита и вычисляет процентное отношение между ними.
2. Дано два текстовых файла. Написать функцию, которая поменяет самую длинную строку первого файла с самой короткой строкой второго файла и наоборот.
3. Описать функцию triangle, формирующую текстовый файл из 9 строк, в первой из которых – один символ '1', во второй – два символа '2', ..., в девятой – девять символов '9'.
4. В файле data.txt построчно записан некоторый текст. Найти строку максимальной длины и ее позицию.
5. Дан текстовый файл. Написать функцию, которая закодирует его, т.е. каждую строку текста переписшет в обратном порядке.
6. В файле data.txt записан некоторый текст построчно. В каждой строке отсортировать слова по возрастанию их длины.
7. Описать функцию line40, которая считывает из входного файла символы до первой точки и записывает их (без точки) в текстовый файл, формируя в нем строки по 40 символов (в последней строке символов может быть меньше).
8. Дан текстовый файл. Подсчитать количество знаков препинания (X) и ко-

личество слов (Y). Если выполняется условие  $\frac{Y}{X} \leq 6$ , то вывести сообщение о том, что текст в достаточной мере обогащен знаками препинания, иначе – недостаточно.

9. Описать функцию, которая подсчитывает количество пустых строк в текстовом файле.
10. Описать функцию для подсчета числа строк, начинающихся и заканчивающихся одним и тем же символом.
11. Описать функцию для подсчета числа строк, состоящих из одинаковых символов.
12. Дан текстовый файл. Написать функцию PersName(<имя\_файла>), которая удалит из текста все имена собственные.
13. Дан текстовый файл. Написать программу, которая, игнорируя исходное деление этого файла на строки, переформатирует его, разбивая на строки так, чтобы каждая строка оканчивалась точкой либо содержала ровно 60 символов, если среди них нет точки.
14. Описать функцию, которая преобразовывает исходный файл с выравниванием текста по левому краю в файл с выравниванием текста по правому

краю. Длину строки считать равной 80 символам.

15. Описать функцию `fromto`, переписывающую содержимое одного текстового файла в другой, но без пустых строк.

16. Дан текстовый файл. Написать функцию, которая слова, содержащие больше четырех гласных, заменит на цифры (1,2,3,4,...) по порядку.

17. В текстовом файле записана непустая последовательность целых чисел, разделенных пробелами. Описать функцию `positive`, записывающую в другой текстовый файл все положительные числа из исходного файла.

18. Описать функцию `lines`, которая построчно печатает содержимое непустого текстового файла, вставляя в начало каждой строки ее порядковый номер (он должен занимать 4 позиции) и пробел.

19. Дан текстовый файл, состоящий из  $N$  строк. Написать функцию `Trans(k1, k2)`, которая меняет местами две строки файла с номерами  $k1$  и  $k2$ .

20. Дан текстовый файл. Написать функцию `DoubleString(n)`, удваивающую в тексте каждую букву в строке с номером  $n$ .

21. Дан текстовый файл без разбиения на абзацы. Написать функцию, которая ищет строки, заканчивающиеся точками, и делает после них новый абзац. Абзацем считать отступ от края на 6 пробелов.

22. Дан текстовый файл. Написать функцию `replace(c1, c2)`, заменяющую в тексте все символы  $c1$  на символы  $c2$ .

23. Дан текстовый файл. Написать процедуру `firsts`, оставляющую в каждой строке исходного текста только первые буквы каждого слова.

24. Дано два текстовых файла. Написать функции, которая запишет в третий файл те строки, которые встречаются в этих двух файлах.

25. Дан текстовый файл. Поменять местами первую и  $N$ -ную строку ( $N$  вводится с клавиатуры).

26. Дан текстовый файл, строки которого состоят из заглавных и строчных букв. Записать текст в новый файл таким образом, чтобы заглавные буквы стали строчными и наоборот.

27. Дан текстовый файл. Написать функцию, которая проверяет, есть ли в нем слова, состоящие только из цифр, если есть – удалить их.

28. Файл содержит текст, записанный в виде криптограммы, в которой буквы истинного текста размещаются в позициях, кратных 3. Напечатать исходный текст.

29. Дан текст. Исключить в каждой строке символы, расположенные между скобками '(', ')'. Сами скобки тоже должны быть исключены. Предполагается, что внутри каждой пары скобок нет других скобок.

30. Дан текстовый файл из латинских букв. Написать функцию, которая подсчитывает общее количество гласных букв и количество каждой гласной в отдельности. Результаты записать в новый файл.

31. Дан текстовый файл. Вывести на экран среднюю строку (две строки, если четное количество строк).

32. Описать функцию, которая построчно переписывает в другой файл содержимое текстового файла, вставляя в начало и в конец каждой строки через пробел знак восклицания.

## 6.2 Двоичные файлы

Указание к работе. Двоичные файлы создавать программно с использованием отдельной функции.

1. Дан некоторый файл, компоненты которого являются целыми числами. Найти сумму компонент первой половины файла и произведение второй половины.
2. Файл содержит список лотерейных билетов с четырехзначными номерами. Выигрышными считаются те билеты, сумма первых 3 цифр которых равна 8. Найти все выигрышные билеты и записать их в новый файл.
3. Дан некоторый файл, компонентами которого являются структуры типа: день, месяц, год. Описать функцию, проверяющую по сегодняшней дате (введенной с клавиатуры), какая из дат в файле является ближайшей к сегодняшнему дню.
4. Дан символьный файл. Удалить из него все записи, расположенные между заглавными буквами латинского алфавита и сами заглавные буквы. Например: jsdgfjAd12jfhgjf8K6775jjjsdhjh6365. В файле останется последовательность jsdgfj6775jjjsdhjh6365.
5. Дан файл, компонентами которого являются целые числа. Отсортировать их по возрастанию и записать в результирующий файл, при этом между каждой их парой вставить число, большее предыдущего, но меньшее последующего. Например, если после сортировки имеем последовательность 1,4,9,20, тогда в результате получим 1,3,4,5,9,15,20. (Примечание: исходный файл заполнять так, чтобы в нем не оказалось одинаковых и рядом стоящих чисел).
6. Записать в новый файл все символы из некоторого символьного файла chrs.dat, не являющиеся буквами. Определить, сколько таких символов.
7. Дан некоторый файл, компоненты которого являются целыми числами. Подсчитать количество компонент, у которых значение совпадает с номером позиции. Например: 0 1 4 6 7 5. Таких компонент три: 0 (номер позиции=0), 1 (номер позиции=1) и 5 (номер позиции=5).
8. Дан некоторый файл, компоненты которого являются вещественными числами. Найти разность сумм первых трех и последних трех компонент файла.
9. В файле содержатся номера автомобилей, стоящих на стоянке (123, 467, 129 и др.) Написать функцию, которая выведет сначала список автомобилей с тремя одинаковыми цифрами, потом с двумя и в конце все остальные.
10. Дан файл, компонентами которого являются структуры типа: день, месяц, год. Удалить из файла все записи, следующие за некоторой датой, введенной с клавиатуры (если такой даты нет - то дописать ее в конец файла).



11. Некоторая упорядоченная последовательность целых чисел записана в файл. Разбить последовательность на две части и вывести на экран сначала вторую часть, затем первую. Обе части выводить в обратном порядке.
12. Дан файл, представляющий собой телефонную книжку некоторого молодого человека, т.е. только семизначные номера сотовых телефонов. У своей новой подруги он взял номер телефона и, естественно, пока дошел до дома, забыл его. Однако он помнит, что номер начинается на 497\*\*\*\*. Написать функцию, которая выведет на экран только те номера из телефонной книги, среди которых может быть номер новой подруги.
13. Дан файл, содержащий данные о студентах: ФИО студента и оценки по предметам: РПАЯВУ, высшая математика, политология, дискретная математика. Написать функцию, которая запишет в новый файл только тех студентов, средний бал сессии которых не меньше 4.
14. Даны два файла, компонентами которых являются вещественные числа. Описать функцию, проверяющую, равны ли файлы между собой.
15. Дан файл, компонентами которого являются структуры типа: день, месяц, год. Записать в другой файл даты, которые меньше некоторой даты, введенной с клавиатуры.
16. Дан некоторый файл, компоненты которого являются вещественными числами. Найти сумму максимального среди четных и минимального среди нечетных компонентов.
17. Дан символьный файл. Удалить из него все пробелы и специальные символы.
18. Файл содержит шестизначные номера проездных билетов. Написать функцию, которая запишет в новый файл только номера «счастливых» билетов. («Счастливыми» считать те билеты, сумма цифр первой половины которых равна сумме цифр второй половины).
19. Дан некоторый файл, компонентами которого являются структуры типа: день, месяц, год. Вывести на экран сначала все зимние даты, затем весенние и т.д.
20. Дан некоторый файл, компоненты которого являются целыми числами. Подсчитать количество нулей, стоящих на четных местах.
21. Записать в новый файл все символы из некоторого символьного файла chrs.dat, являющиеся цифрами. Проверить, все ли цифры из диапазона 0...9 попали в новый файл, вывести те, которые не попали.
22. Дан файл, содержащий данные о погоде: время года, дату, облачность и температуру. Студенты со своими преподавателями решили идти в поход. Проверить, удовлетворяют ли погодные условия данному мероприятию путем ввода интересующей даты. Если нет, то программа должна предложить две близлежащие даты с нормальной погодой. (Примечание: хорошей погодой считается: безоблачно, для зимы +5 и более градусов, для весны и осени +15 и более, для лета - +25)

23. Дан символьный файл. Вывести на экран максимальную последовательность цифр. Например: jsdghjd**12**jfhghf**86775**jjjsdhjh**6365**. На экран будет выведено: 86775, так как длина этого фрагмента с цифрами больше длины (12)=2 и больше длины (6365)=4.
24. Дан некоторый файл, компонентами которого являются вещественные числа. Найти сумму компонент первой половины файла и произведение второй половины. Найти среднее арифметическое компонент, стоящих на местах, кратных трем.
25. Дан файл file1.dat, компонентами которого являются структуры типа: фамилия, имя, отчество. Поля "имя" и "отчество" записаны в полном формате (например: Петров Петр Петрович). Переписать в файл file2.dat все записи по следующему образцу: Петров П.П.
26. Даны два символьных файла. Записать в новый файл все совпадающие компоненты исходных файлов.
27. Дан файл, содержащий данные о студентах: ФИО студента, оценки по пяти дисциплинам и номер телефона родителей. Написать функцию, которая запишет в новый файл те записи о студентах, для которых стоит позвонить родителям по поводу сдачи сессии, т.е. средний бал  $< 20$ .
28. Даны два целочисленных файла, компоненты которых упорядочены по возрастанию. Объединить эти файлы в третий, не нарушая упорядоченности.
29. Дан файл, содержащий данные о преподавателях двух кафедр: ФИО, предмет, который он ведет, и название кафедры. Написать функцию, которая выведет сначала сотрудников одной кафедры, затем другой. Список сотрудников каждой кафедры должен идти в алфавитном порядке.
30. Дан символьный файл. Вставить после каждой цифры пробел.
31. Дан символьный файл, содержащий пробелы. Удалить из файла все "однобуквенные" слова и лишние пробелы. Например: jkd fh h kjdfh j jf. В результате: jkd fh kjdfh jf.
32. Дан файл, содержащий данные о студентах, проживающих в общежитии: ФИО, номер курса, смену(1-3), в которую он должен дежурить на проходной общежития. Создать три файла: в первый записать дежурящих в первую смену, во второй – во вторую, в третий – в третью.

## Лабораторная работа №6. Разработка приложений в среде «Embarcadero RAD Studio 2010»

### 1 Цель работы

Научиться проектировать каркас простейшего приложения в интегрированной среде визуального программирования C++Builder. Получить практические навыки в написании и отладке программ в среде C++Builder.

### 2 Порядок выполнения работы

Внимательно изучить краткую теорию работы. Выполнить самостоятельно примеры и задания из краткой теории. Получить задание на выполнение лабораторной работы (раздел 6) согласно своему варианту. Разработать и отладить программу. Составить и защитить отчет о лабораторной работе у преподавателя.

### 3 Содержание отчета

- наименование и цель работы;
- задание на лабораторную работу согласно варианту;
- текст программы на алгоритмическом языке C++;
- результаты работы программы.

### 4 Краткая теория

#### 4.1 Интегрированная среда разработки программ C++Builder

Среда *C++Builder* визуально реализуется в виде нескольких одновременно раскрытых на экране монитора окон. Количество, расположение, размер и вид окон может меняться программистом в зависимости от его текущих нужд, что значительно повышает производительность работы.

Внешний вид среды *C++Builder* после запуска представлен рисунке 6.1.

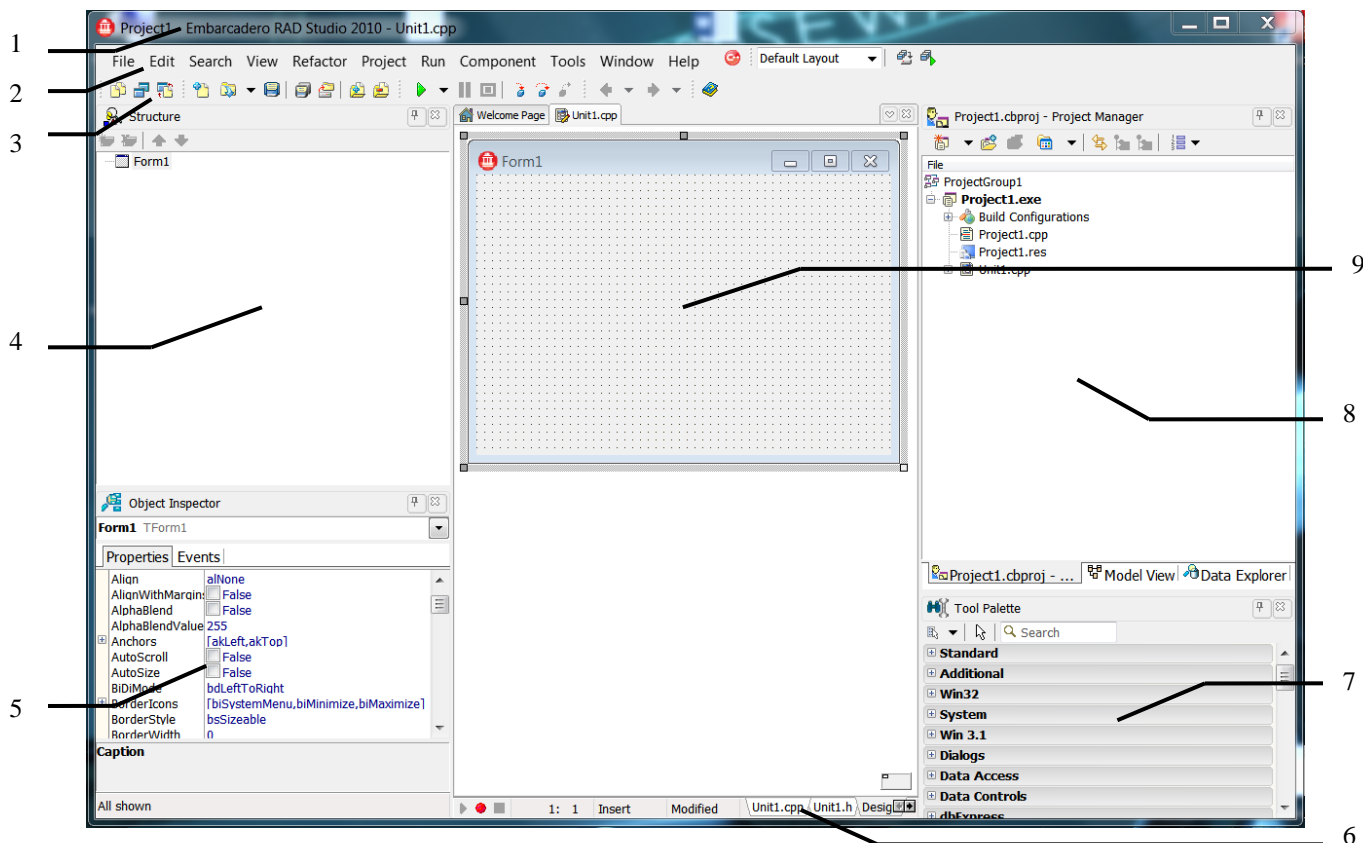


Рисунок 6.1 – Внешний вид среды C++Builder, где

1 - главное окно; 2 – основное меню; 3 – пиктограммы основного меню; 4 – окно дерева объектов; 5 - окно инспектора объектов; 6 – панель переключения окон; 7- палитра компонентов; 8 – окно файлов проекта; 9 – окно

**Главное окно** всегда присутствует на экране и предназначено для управления процессом создания программы. Основное меню содержит все необходимые средства для управления проектом. Пиктограммы облегчают доступ к наиболее часто применяемым командам основного меню. Через меню компонентов осуществляется доступ к набору стандартных сервисных программ среды **C++Builder**, которые описывают некоторый визуальный элемент (компонент), помещенный программистом в окно формы. Каждый компонент имеет определенный набор свойств (параметров), которые можно задавать (например, цвет, заголовок окна, надпись на кнопке, размер и тип шрифта и др.).

**Окно инспектора объектов** (вызывается с помощью клавиши **F11**) предназначено для изменения свойств выбранных компонентов и состоит из двух страниц. Страница **Properties** (Свойства) предназначена для изменения необходимых свойств компонента, страница **Events** (События) – для определения реакции компонента на то или иное событие (например, нажатие определенной клавиши или щелчок «мышью» по кнопке).

**Окно формы** представляет собой проект *Windows*-окна программы. В это окно в процессе написания программы помещаются необходимые компоненты. Причем при выполнении программы помещенные компоненты будут иметь тот же вид, что и на этапе проектирования.

**Окно текста** программы предназначено для просмотра, написания и редактирования текста программы. В системе *C++Builder* используется язык программирования *C++*. При первоначальной загрузке в окне текста программы находится текст, содержащий минимальный набор операторов для нормального функционирования пустой формы в качестве *Windows*-окна. При помещении некоторого компонента в окно формы текст программы автоматически дополняется описанием необходимых для его работы библиотек стандартных программ.

Программа в среде *C++Builder* составляется как описание алгоритмов, которые необходимо выполнить, если возникает определенное событие, связанное с формой (например щелчок «мыши» на кнопке – событие *OnClick*, создание формы – *OnCreate*). Для каждого обрабатываемого в форме события, с помощью страницы *Events* инспектора объектов в тексте программы организуется функция, между ключевыми символами {...} которой программист записывает на языке *C++* требуемый алгоритм.

Переключение между окном формы и окном текста программы осуществляется с помощью клавиши **F12**.

## 4.2 Структура программ C++Builder

Программа в *C++Builder* состоит из следующих файлов:

- файла проекта (файл с расширением *bpr*);
- файлов описания класса формы, с расширением *h*
- одного или нескольких файлов исходного текста (с расширением *cpp*);
- файлов с описанием окон формы (с расширением *dfm*).

В **файле проекта** находится информация о модулях, составляющих данный проект.

**Файл исходного текста** – программный модуль предназначенный для размещения текстов программ. В этом файле программист размещает текст программы, написанный на языке *C++*.





## 4.3 Пример написания программы

**Задание:** составить программу вычисления для заданных значений *x*, *y*, *z* арифметического выражения

$$u = tg^2(x + y) - e^{y-z} \sqrt{\cos x^2 + \sin z^2}.$$

Панель диалога программы организовать в виде, представленном на рисунке 6.2.

#### 4.3.1 Настройка формы

Пустая форма в правом верхнем углу имеет кнопки управления, которые предназначены: для свертывания формы в пиктограмму , для разворачивания формы на весь экран  и возвращения к исходному размеру  и для закрытия формы . С помощью мыши, «захватывая» одну из кромок формы или выделенную строку заголовка отрегулируйте нужные размеры формы и ее положение на экране.

#### 4.3.2 Изменение заголовка формы


Новая форма имеет одинаковые имя (*Name*) и заголовок (*Caption*) - *FORM1*. Имя формы менять не рекомендуется, т.к. оно входит в текст программы.

Для изменения заголовка вызовите окно инспектора объектов (*F11*) и щелкните кнопкой мыши на форме. В форме инспектора объектов найдите и щелкните мышью на *Properties – Caption*. В выделенном окне наберите “Лаб. раб. N1. Иванов А.А.”(без кавычек).

#### 4.3.3 Размещение строки ввода (TEdit)

При необходимости ввести из формы в программу или вывести на форму информацию, которая вмещается в одну строку, используют окно однострочного редактора текста, представляемого компонентом *TEdit*.

В данной программе с помощью однострочного редактора будут вводиться переменные *x,y,z* типа *float*.

Выберите в меню компонентов *Standard* пиктограмму , щелкните мышью в том месте формы, где вы хотите ее поставить. Вставьте три компонента *TEdit* в форму. Захватывая их “мышью” отрегулируйте размеры окон и их положение. Обратите внимание на то, что в тексте программы появились три новых одностипных переменных *Edit1*, *Edit2*, *Edit3*. В каждой из этих переменных в поле «*Text*» будет содержаться строка символов и отображаться в соответствующем окне *Edit*.


Так как численные значения переменных *x,y,z* имеют действительный тип для преобразования строковой записи числа, находящегося в переменной *Edit1->Text*, в действительное используется стандартная функция *StrToFloat*. Например, *x=StrToFloat(Edit1->Text)*, где *x* – переменная вещественного типа.

В случае если исходные данные имеют целочисленный тип, например *int*, то используется стандартная функция *StrToInt*. Например, *y=StrToInt(Edit1->Text)*, где *y* – переменная вещественного типа.


При этом в записи числа не должно быть пробелов, а действительное число пишется с десятичной запятой.

С помощью инспектора объектов установите шрифт и размер символов отражаемых в строке *Edit* (свойство *Font*).

#### 4.3.4 Размещение надписей (TLabel)

На форме (см. рисунок 6.2) имеются четыре пояснительные надписи. Для нанесения таких надписей на форму используется компонент *TLabel*. Выберите в палитре компонентов на странице *Standard* пиктограмму , щелкните на ней мышью. После этого в нужном месте формы щелкните мышью, появится надпись *Label1*. Прodelайте это для четырех надписей. Для каждой надписи, щелкнув на ней мышью, отрегулируйте размер и, изменив свойство *Caption* инспектора объектов, введите строку, например “*Введите значение X:*” (без кавычек), а также выберите размер символов (свойство *Font*).

#### 4.3.5 Размещение многострочного окна вывода (TMemo)

Для вывода результатов работы программы обычно используется текстовое окно, которое представлено компонентом (*TMemo*). Выберите в палитре компонентов на странице *Standard* пиктограмму  и поместите компонент *TMemo* на форму. С помощью мыши отрегулируйте его размеры и местоположение. Далее с помощью инспектора свойство «*ScrollBars*» установите в «*SSBoth*» в окне появятся вертикальная и горизонтальная полосы прокрутки.

Информация, которая отображается построчно в окно типа *TMemo*, находится в массиве строк *Memo1->Lines*. Каждая строка имеет тип *AnsiString*.

Для чистки окна используется метод *Memo1->Clear*. Для того чтобы добавить новую строку в окно, используется метод

*Memo->Lines->Add* (<переменная *muna AnsiString*>).

При необходимости вывода числа, хранящегося в переменной действительного или целого типа, его надо предварительно преобразовать к типу *AnsiString* и добавить в массив *Memo1->Lines*.

Например, если переменная *u=100* целого типа, то метод

*Memo1->Lines->Add*('Значение *u*='+*IntToStr(u)*)

сделает это, и в окне появится строка «Значение *u*=100».

Если переменная *u=-256,38666* - действительная, то при использовании метода

*Memo1->Lines->Add*('Значение *u*='+*FloatToStrF(u,ffFixed,8,2)*)

будет выведена строка «Значение *u*= -256.39». При этом под все число отводится восемь позиций, из которых две позиции занимает его дробная часть.

В том случае, если число строк в массиве *Memo1* превышает размер окна, то для просмотра всех строк используется вертикальная полоса прокрутки. Если длина строки *Memo1* превосходит количество символов в строке окна, то в окне отображается только начало строки. Для просмотра всей строки используется горизонтальная полоса прокрутки.

#### 4.3.6 Написание кода обработки события показа формы (*FormShow*)

При запуске программы, перед отображением формы, возникает событие «При открытии формы» (*OnShow*).

Создадим функцию–обработчик этого события, которая заносит начальные значения переменных *x*, *y*, *z* в соответствующие окна *TEdit*, а в окне *TMemo* помещает строку с указанием номера группы и фамилию студента.


Для этого необходимо дважды щелкнуть мышью на любом свободном месте формы. В инспекторе свойств, переключиться на закладку «*Events*» и два раза щелкнуть мышкой по полю «*OnShow*»

На экране появится текст, в котором автоматически внесен заголовок функции-обработчика события открытия формы:

```
void __fastcall TForm1::FormShow(TObject *Sender)
{
}
```

Между {...} вставляется текст программы (см. пример, расположенный ниже).

#### 4.3.7 Написание кода обработки события нажатия кнопки (*ButtonClick*)

Поместите на форму кнопку, которая описывается компонентом *TButton*. Для этого необходимо выбрать в палитре компонентов на странице *Standart* пиктограмму . С помощью инспектора объектов измените заголовок (*Caption*) – *Button1* на слово “*Выполнить*” (без кавычек) или любое другое по желанию. Отрегулируйте положение и размер кнопки.


После этого два раза щелкните мышью на кнопке, появится текст программы, дополненной заголовком функции-обработчика события нажатия кнопки (*void \_\_fastcall TForm1::Button1Click(TObject \*Sender)*).

Наберите текст этой функции, приведенный в примере.

#### 4.3.8 Запуск и работа с программой

Запустить программу можно, выбрав команду *Run* в главном меню



**Run**, или клавишу **F9**, или пиктограмму . При этом происходит трансляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с расширением *exe*. На экране появляется активная форма программы (рисунок 6.2).

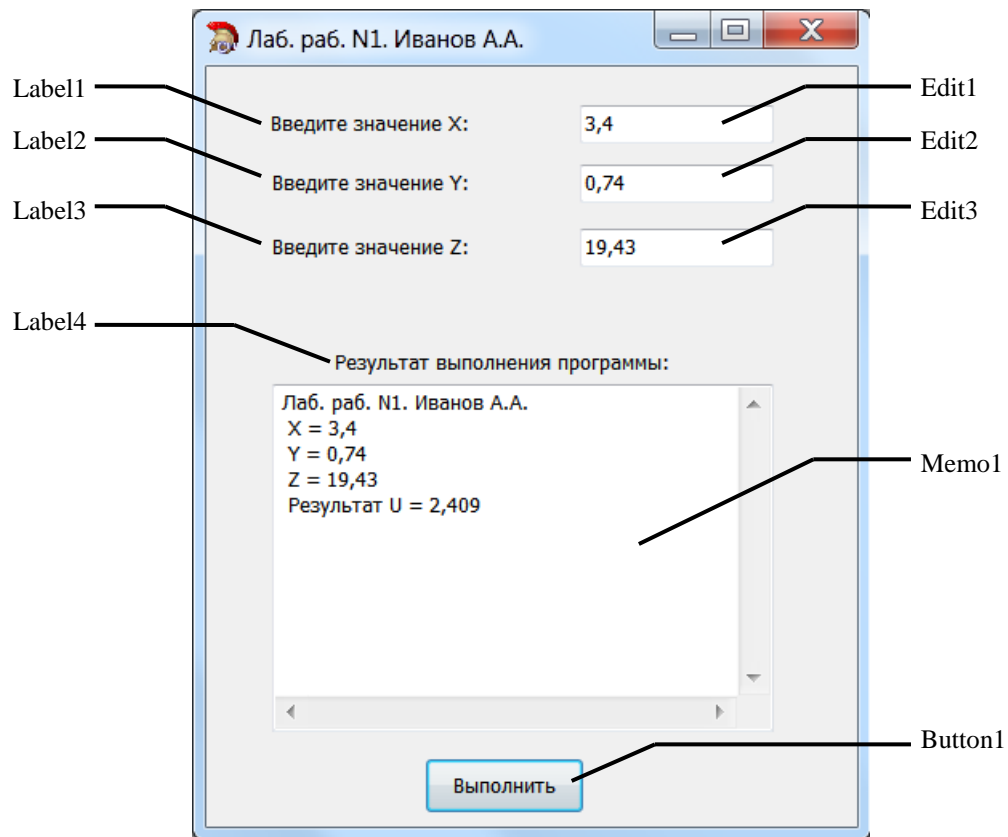



Рисунок 6.2 - Активная форма

Работа с программой происходит следующим образом. Нажмите (щелкните мышью) кнопку “**Выполнить**”. В окне *Memo1* появляется результат. Измените исходные значения *x*, *y*, *z* в окнах *Edit* и снова нажмите кнопку “**Выполнить**” - появятся новые результаты.

Завершить работу программы можно нажав *ProgramReset* в главном меню **Run**, или кнопку  на форме.

Текст программы имеет вид:

```
//-----
#include <math.h>
void __fastcall TForm1::FormShow(TObject *Sender)
{Edit1->Text="3,4";           // Начальное значение X
 Edit2->Text="0,74";          // Начальное значение Y
 Edit3->Text="19,43";         // Начальное значение Z
 Memo1->Clear();              // Очистка окна редактора Memo1
 // Вывод строки в многострочный редактор Memo1
 Memo1->Lines->Add("Лаб. раб. N1. Иванов А.А.");
```

```

}
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{ float x,y,z,a,b,c,u;
  x=StrToFloat(Edit1->Text);          // Считывается значение X
  Memo1->Lines->Add(" X = "+Edit1->Text); //Вывод X в окно Memo1
  y=StrToFloat(Edit2->Text);          // Считывается значение Y
  Memo1->Lines->Add(" Y = "+Edit2->Text); //Вывод Y в окно Memo1
  z=StrToFloat(Edit3->Text);          // Считывается значение Z
  Memo1->Lines->Add(" Z = "+Edit3->Text); //Вывод Z в окно Memo1
  // Вычисляем арифметическое выражение
  a=pow(sin(x+y),2)/pow(cos(x+y),2);
  b=exp(y-z);
  c=sqrt(cos(x*x)+sin(z*z));
  u=a-b*c;
  // Выводим результат в окно Memo1
  Memo1->Lines->Add(" Результат U = "
                  +FloatToStrF(u,ffFixed,8,3));
}

```

## 5 Сетка строк - компонент *TStringGrid*

При работе с массивами ввод и вывод информации на экран удобно организовывать в виде таблиц. Компонент *TStringGrid* предназначен для отображения информации в виде двумерной таблицы, каждая ячейка которой представляет собой окно однострочного редактора (аналогично окну *TEdit*).

Доступ к информации осуществляется с помощью свойства

*Cells[int ACol;int ARow]: AnsiString,*

где *ACol*, *ARow* - индексы элемента двумерного массива по столбцам и строкам соответственно. Свойства *ColCount* и *RowCount* устанавливают количество столбцов и строк в таблице, а свойства *FixedCols* и *FixedRows* задают количество столбцов и строк фиксированной зоны. Фиксированная зона выделена другим цветом, и в нее запрещен ввод информации с клавиатуры.

### 5.1 Пример выполнения задания

**Задание:** создать программу для определения вектора  $\vec{Y} = A * \vec{B}$ , где *A* - квадратная матрица размерностью  $N \times N$ , а *Y*, *B* – одномерные массивы размерностью *N*. Элементы вектора *Y* определяются по формуле  $Y_i = \sum_{j=1}^N A_{ij} \cdot B_j$ .

Значения *N* вводить в компонент *TEdit*, *A* и *B* - в компонент *TStringGrid*. Результат, после нажатия кнопки типа *TButton*, вывести в компонент *TStringGrid*.

Панель диалога приведена на рисунке 6.3.

#### 5.1.1 Настройка компонента *TStringGrid*

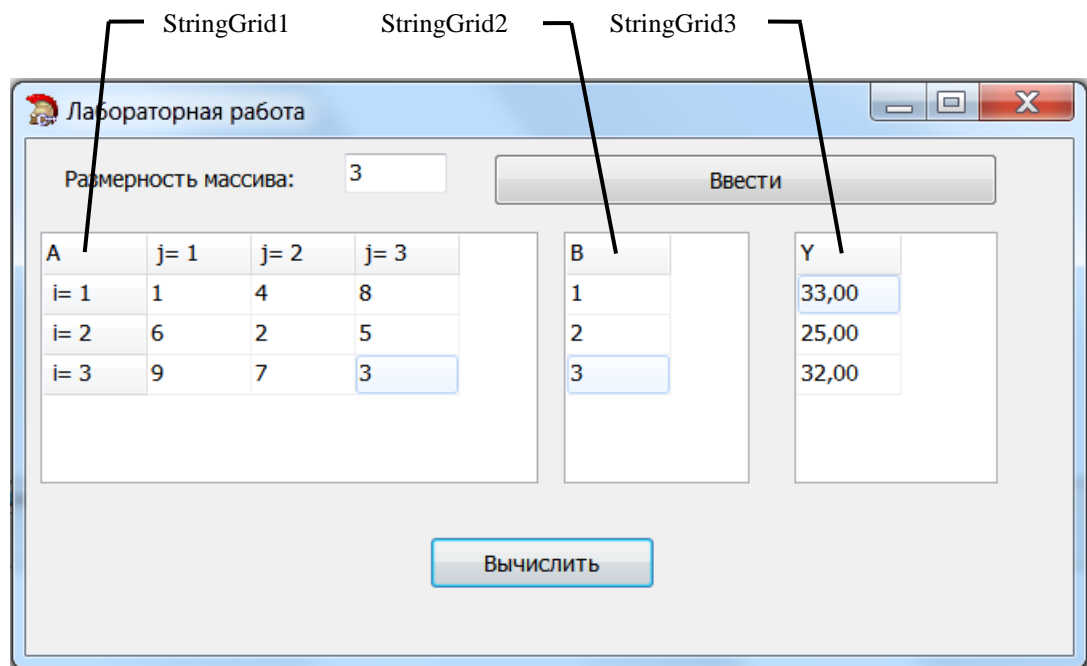



Рисунок 6.3 – Приложение с использованием сетки строк (*StringGrid*)

Для установки компонента *TStringGrid* на форму необходимо на странице **Additional** палитры компонентов щелкнуть мышью по пиктограмме . После этого щелкните мышью в нужном месте формы.

Захватывая края компонента, отрегулируйте его размер. В инспекторе объектов значения свойств *ColCount* и *RowCount* установите 2 (два столбца и две строки), а *FixedCols* и *FixedRows* установите 1 (один столбец и одна строка с фиксированной зоной).

Т.к. компоненты *StringGrid2* и *StringGrid3* имеют только один столбец, то у них: *ColCount=1*, *RowCount=2*, *FixedCols=0* и *FixedRows=1*.

По умолчанию в компонент *TStringGrid* запрещен ввод информации с клавиатуры, поэтому необходимо свойство *Options->goEditing* для компонентов *StringGrid1* и *StringGrid2* установить в положение *true*.

Текст программы приведен ниже.

```
#include <math.h>
int A[100][100], B[100], Y[100];      // Объявление массивов
int N;

void __fastcall TForm1::FormShow(TObject *Sender)
{
    N=3;                               // Размерность массива
    Edit1->Text=FloatToStr(N);
    //Задание числа строк и столбцов в таблицах
    StringGrid1->ColCount=N+1;
    StringGrid1->RowCount=N+1;
    StringGrid2->RowCount=N+1;
    StringGrid3->RowCount=N+1;
    // Ввод в левую верхнюю ячейку таблицы названия массива
```

```

StringGrid1->Cells[0][0]="Массив А:";
StringGrid2->Cells[0][0]="Массив В:";
StringGrid3->Cells[0][0]="Массив Y:";
//Заполнение верхнего и левого столбцов поясняющими подписями}
for (int i=1;i<=N; i++)
{ StringGrid1->Cells[0][i]=" i= "+IntToStr(i);
  StringGrid1->Cells[i][0]=" j= "+IntToStr(i);
}
}
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{ N=StrToInt(Edit1->Text);
  // Задание числа строк и столбцов в таблицах
StringGrid1->ColCount=N+1;
StringGrid1->RowCount=N+1;
StringGrid2->RowCount=N+1;
StringGrid3->RowCount=N+1;
// Заполнение верхнего и левого столбцов поясняющими подписями
for (int i=1; i<=10; i++)
{StringGrid1->Cells[0][i]=" i= "+IntToStr(i);
  StringGrid1->Cells[i][0]=" j= "+IntToStr(i);
}
}
//-----
void __fastcall TForm1::Button2Click(TObject *Sender)
{ int i,j;
  float s;
  // Заполнение массива А элементами из таблицы StringGrid1
for (i=1; i<=N; i++)
  for (j=1; j<=N; j++)
    A[i][j]=StrToFloat(StringGrid1->Cells[j][i]);
//Заполнение массива В элементами из таблицы StringGrid2
for (i=1; i<=N; i++)
  B[i]=StrToFloat(StringGrid2->Cells[0][i]);
// Умножение массива А на массив В
for (i=1; i<=N; i++)
{ s=0;
  for (j=1; j<=N; j++)
    s=s+A[i][j]*B[j];
  Y[i]=s;
  // Вывод результата в таблицу StringGrid3
  StringGrid3->Cells[0][i]=FloatToStrF(Y[i],ffFixed,6,2);
}
}

```

## 6 Варианты заданий для самостоятельного решения

Во всех заданиях по теме «**Массивы**» скалярные переменные вводить с помощью компонента **TEdit** с соответствующим пояснением в виде компонента **TLabel**. Скалярный результат выводить в виде компонента **TLabel**.

Массивы представлять на форме в виде компонентов *TStringGrid*, в которых 0-й столбец и 0-ю строку использовать для отображения индексов массивов. Вычисления выполнять, после нажатия кнопки типа *TButton*.

1. Задана матрица размером  $N \times M$ . Получить массив  $B$ , присвоив его  $k$ -му элементу значение 0, если все элементы  $k$ -го столбца матрицы нулевые, и значение 1 в противном случае.

2. Задана матрица размером  $N \times M$ . Получить массив  $B$ , присвоив его  $k$ -му элементу значение 1, если элементы  $k$ -й строки матрицы упорядочены по убыванию, и значение 0 в противном случае.

3. Задана матрица размером  $N \times M$ . Получить массив  $B$ , присвоив его  $k$ -му элементу значение 1, если  $k$ -я строка матрицы симметрична, и значение 0 в противном случае.

4. Задана матрица размером  $N \times M$ . Определить  $k$  – количество «особых» элементов матрицы, считая элемент «особым», если он больше суммы остальных элементов своего столбца.

5. Задана матрица размером  $N \times M$ . Определить  $k$  – количество «особых» элементов матрицы, считая элемент «особым», если в его строке слева от него находятся элементы, меньшие его, а справа – большие.

6. Задана символьная матрица размером  $N \times M$ . Определить  $k$  - количество различных элементов матрицы (т.е. повторяющиеся элементы считать один раз).

7. Дана матрица размером  $N \times M$ . Упорядочить ее строки по неубыванию их первых элементов.

8. Дана матрица размером  $N \times M$ . Упорядочить ее строки по неубыванию суммы их элементов.

9. Дана матрица размером  $N \times M$ . Упорядочить ее строки по неубыванию их наибольших элементов.

10. Определить, является ли заданная квадратная матрица  $n$ -го порядка симметричной относительно побочной диагонали.

11. Для матрицы размером  $N \times M$  вывести на экран все ее седловые точки. Элемент матрицы называется седловой точкой, если он является наименьшим в своей строке и одновременно наибольшим в своем столбце или, наоборот.

12. В матрице  $n$ -го порядка переставить строки так, чтобы на главной диагонали матрицы были расположены элементы, наибольшие по абсолютной величине.

13. В матрице  $n$ -го порядка найти максимальный среди элементов, лежащих ниже побочной диагонали, и минимальный среди элементов, лежащих выше главной диагонали.

14. В матрице размером  $N \times M$  поменять местами строку, содержащую элемент с наибольшим значением со строкой, содержащей элемент с наименьшим значением.

15. Из матрицы  $n$ -го порядка получить матрицу порядка  $n-1$  путем удаления из исходной матрицы строки и столбца, на пересечении которых расположен элемент с наибольшим по модулю значением.
16. Дан массив из  $k$  символов. Вывести на экран сначала все цифры, входящие в него, а затем все остальные символы, сохраняя при этом взаимное расположение символов в каждой из этих двух групп.
17. Дан массив, содержащий от 1 до  $k$  символов, за которым следует точка. Вывести этот текст в обратном порядке.
18. Дан непустой массив из цифр. Вывести на экран цифру, наиболее часто встречающуюся в этом массиве.
19. Отсортировать элементы массива  $X$  по возрастанию.
20. Элементы массива  $X$  расположить в обратном порядке.
21. Элементы массива  $X$  циклически сдвинуть на  $k$  позиций влево.
22. Элементы массива  $X$  циклически сдвинуть на  $n$  позиций вправо.
23. Преобразовать массив  $X$  по следующему правилу: все отрицательные элементы массива перенести в начало, а все остальные – в конец, сохраняя исходное взаимное расположение, как среди отрицательных, так и среди остальных элементов.
24. Элементы каждого из массивов  $X$  и  $Y$  упорядочены по неубыванию. Объединить элементы этих двух массивов в один массив  $Z$  так, чтобы они снова оказались упорядоченными по неубыванию.
25. Дан массив из  $k$  символов. Определить, симметричен ли он, т.е. читается ли он одинаково слева направо и справа налево.
26. Дано два массива. Найти наименьшее среди тех элементов первого массива, которые не входят во второй массив.
27. Определить количество инверсий в этом массиве  $X$  (т.е. таких пар элементов, в которых большее число находится слева от меньшего:  $x_i > x_j$  при  $i < j$ ).
28. Дан массив из строчных латинских букв. Вывести на экран в алфавитном порядке все буквы, которые входят в этот текст по одному разу.
29. Вывести на экран заданный массив из  $k$  символов, удалив из него повторные вхождения каждого символа.
30. Определить сколько различных символов входит в заданный текст, содержащий не более  $k$  символов и оканчивающийся точкой (в сам текст точка не входит).

## Лабораторная работа №7. Программирование с использованием компонент работы со строками

### 1 Цель работы

Изучить правила работы с компонентами *TListBox* и *TComboBox*. Получить практические навыки в написании и отладке программ работы со строками.

### 2 Порядок выполнения работы

Внимательно изучить краткую теорию работы. Выполнить самостоятельно примеры и задания из краткой теории. Получить задание на выполнение лабораторной работы (раздел 6) согласно своему варианту. Разработать и отладить программу. Составить и защитить отчет о лабораторной работе у преподавателя.

### 3 Содержание отчета

- наименование и цель работы;
- задание на лабораторную работу согласно варианту;
- текст программы на алгоритмическом языке *C++*;
- результаты работы программы.

### 4 Краткая теория

#### 4.1 Типы данных для работы со строками

Строковые типы данных *string* и *wstring* доступны при подключении библиотеки работы со строками:

```
#include <string>
```

Строковый тип данных *AnsiString* доступен при подключении библиотеки

```
#include <dstring.h >
```

##### 4.1.1 Длинная строка типа *string* и *AnsiString*

При работе с этим типом данных память выделяется по мере необходимости (динамически) и может занимать всю доступную программе память. Вначале компилятор выделяет для переменной 4 байта, в которых размещается номер ячейки памяти, начиная с которой будет располагаться символьная строка. На этапе выполнения программа определяет необходимую длину цепочки символов и обращается к ядру операционной системы с требованием выделить необходимую память.

Процедуры и функции для работы с короткими и длинными строками представлены в Приложении 4.

#### 4.1.2 Широкая строка типа *wstring*

Данный тип введен для обеспечения совместимости с компонентами, основанными на *OLE*-технологии. От типа *string* отличается только тем, что для представления каждого символа используется не один, а два байта.

#### 4.2 Компонент *TListBox*

Компонент *TListBox* представляет собой список, элементы которого выбираются при помощи клавиатуры или мыши. Список элементов задается свойством *Items*, через методы *Add*, *Delete* и *Insert* которого используются для добавления, удаления и вставки строк соответственно. Объект *Items* (*TString*) хранит строки, находящиеся в списке. Для определения номера выделенного элемента используется свойство *ItemIndex*.

#### 4.3 Компонент *TComboBox*

Комбинированный список *TComboBox* представляет собой комбинацию списка *TListBox* и однострочного редактора *TEdit*, поэтому практически все свойства заимствованы у этих компонентов.

Для работы с окном редактирования используется свойство *Text*, как в *TEdit*, а для работы со списком выбора – свойство *Items* как в *TListBox*. Существует пять модификаций компонента, определяемых его свойством *Style*. В модификации *csSimple* список всегда раскрыт, в остальных он раскрывается после нажатия кнопки справа от редактора.

#### 4.4 Компонент *TBitBtn*

Компонент *TBitBtn* расположен на странице *Additonal* палитры компонентов и представляет собой разновидность стандартной кнопки *TButton*. Его отличительная особенность – наличие растрового изображения на поверхности кнопки, которое определяется свойством *Glyph*. Кроме того, имеется свойство *Kind*, которое задает одну из 11 стандартных разновидностей кнопок. Нажатие любой из них, кроме *bkCustom* и *bkHelp* закрывает модальное окно и возвращает в программу результат *mr\*\*\** (например *bkOk* возвращает *mrOk*). Кнопка *bkClose* закрывает главное окно и завершает работу программы.

#### 4.5. Обработка событий

Обо всех происходящих в системе событиях, таких как создание формы, нажатие кнопки мыши или клавиатуры и т.д., ядро *Windows* информирует окна путем послышки соответствующих сообщений. Среда *C++Builder* позволяет принимать и обрабатывать большинство таких сообщений. Каждый



компонент содержит обработчики сообщений на странице *Events* инспектора объектов.

Для создания обработчика события необходимо раскрыть список компонентов в верхней части окна инспектора объектов и выбрать необходимый компонент. Затем, на странице *Events*, нажатием левой клавиши мыши, выбрать обработчик и дважды щелкнуть по его правой (белой) части. В ответ *C++Builder* активизирует окно текста программы и покажет заготовку процедуры обработки выбранного события.

Каждый компонент имеет свой набор обработчиков событий, однако некоторые из них присущи большинству компонентов. Наиболее часто применяемые события представлены в таблице 7.1.

Таблица 7.1 – Наиболее распространенные события у компонентов

Событие	Описание события
<i>OnActivate</i>	Форма получает это событие при активации
<i>OnCreate</i>	Возникает при создании формы (компонент <i>TForm</i> ). В обработчике данного события следует задавать действия, которые должны происходить в момент создания формы, например установка начальных значений
<i>OnKeyPress</i>	Возникает при нажатии кнопки на клавиатуре. Параметр <i>Key</i> имеет тип <i>Char</i> и содержит <i>ASCII</i> -код нажатой клавиши (клавиша <i>Enter</i> клавиатуры имеет код <i>13</i> , клавиша <i>Esc</i> - <i>27</i> и т.д.). Обычно это событие используется в том случае, когда необходима реакция на нажатие одной из клавиш
<i>OnKeyDown</i>	Возникает при нажатии клавиши на клавиатуре. Обработчик этого события получает информацию о нажатой клавише и состоянии клавиш <i>Shift</i> , <i>Alt</i> и <i>Ctlr</i> , а также о нажатой кнопке мыши. Информация о клавише передается параметром <i>Key</i> , который имеет тип <i>Word</i>
<i>OnKeyUp</i>	Является парным событием для <i>OnKeyDown</i> и возникает при отпускании ранее нажатой клавиши
<i>OnClick</i>	Возникает при нажатии кнопки мыши в области компонента
<i>OnDblClick</i>	Возникает при двойном нажатии кнопки мыши в области компонента

## 5 Порядок выполнения индивидуального задания на примере

**Задание:** написать программу подсчета числа слов в произвольной строке. В качестве разделителя может быть любое число пробелов. Для ввода строк и работы с ними использовать *TComboBox*. Ввод строки заканчивать нажатием клавиши *Enter*. Для выхода из программы использовать кнопку *Close*.

Панель диалога будет иметь вид, представленный на рисунке 7.1.

Текст программы приведен ниже в разделе 5.1.

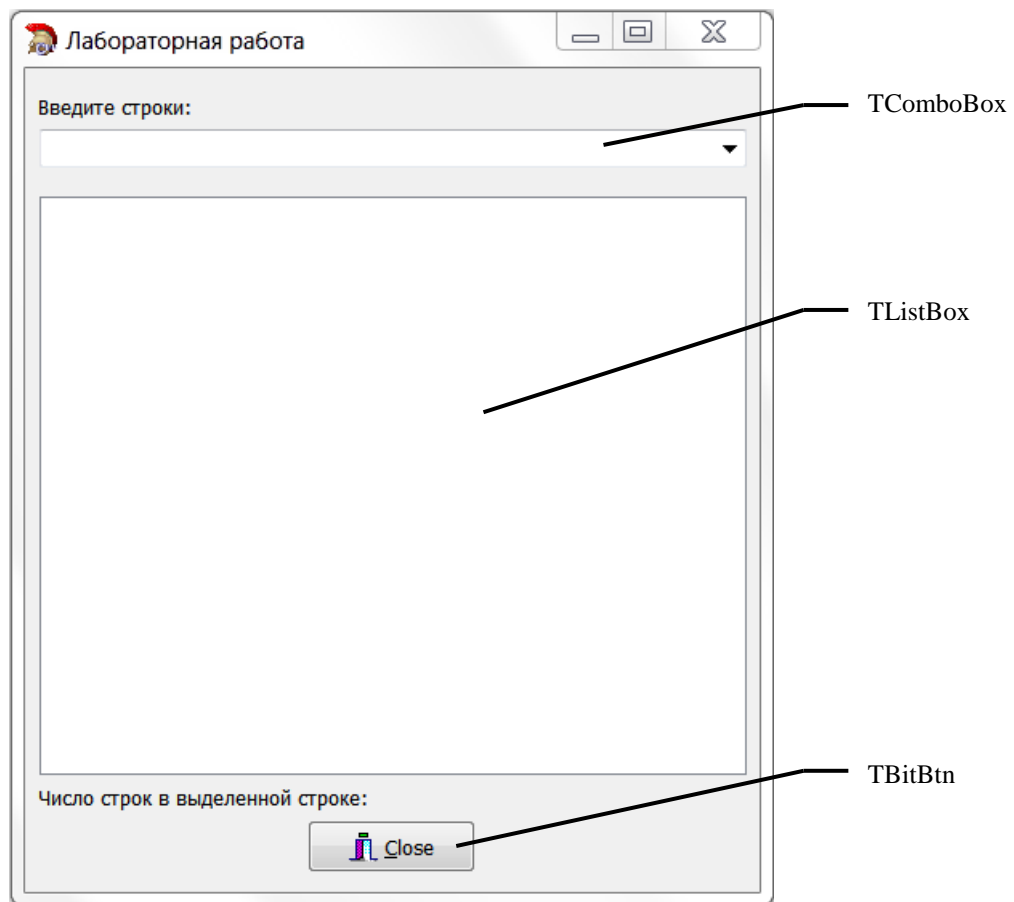


Рисунок 7.1 – Внешний вид приложения

## 5.1 Текст программы

```
#include <dstring.h>
void __fastcall TForm1::FormActivate(TObject *Sender)
{
    ComboBox1->SetFocus(); // Передача фокуса ComboBox1
}
//-----
void __fastcall TForm1::ComboBox1KeyPress(TObject *Sender, char
&Key)
{ if (Key==13)
    { // Если нажата клавиша Enter, то...
// Строка из окна редактирования
    ListBox1->Items->Add(ComboBox1->Text);
    ComboBox1->Items->Add(ComboBox1->Text);
    ComboBox1->Text=""; // Очистка окна редактирования
    }
}
//-----
```

```

void __fastcall TForm1::ListBox1Click(TObject *Sender)
{ int n=0; // Содержит число слов
  int ind=0;
  int nst=ListBox1->ItemIndex; //Определение номера выбранной строки
  // Занесение выбранной строки в переменную st
  AnsiString st=ListBox1->Items->Strings[nst];
  for (int i=1; i<=st.Length(); i++)
  { // Просмотр всех символов строки st
    switch (ind)
    {case 0: if (st[i]!=32)
      { // Если встретился символ после пробела
        ind=1;
        n++; // Число слов увеличивается на единицу
        break;
      }
      case 1: if (st[i]==32) ind=0; //Если встретился пробел
//после символов
        break;
      }
    }
  }
  // Вывод числа слов в Label3
  Label2->Caption="Число слов в выделенной строке:
"+IntToStr(n) ;
}

```

## 6 Задания для самостоятельной работы

Во всех заданиях исходные данные вводить с помощью компонента *TEdit* в компонент *TListBox* либо с помощью свойства *Text* в свойство *Items* компонента *TComboBox*. Скалярный результат выводить с помощью компонента *TLabel*. Ввод строки заканчивать нажатием клавиши *Enter*. Для выхода из программы использовать кнопку *Close*. Для расчетов вводить несколько различных строк.

1. Дана строка, состоящая из групп нулей и единиц. Каждая группа отделяется от другой одним или несколькими пробелами. Найти количество групп с пятью символами.

2. Дана строка, состоящая из групп нулей и единиц. Найти и вывести на экран самую короткую группу.

3. Дана строка, состоящая из групп нулей и единиц. Подсчитать количество символов в самой длинной группе.

4. Дана строка, состоящая из групп нулей и единиц. Найти и вывести на экран группы с четным количеством символов.

5. Дана строка, состоящая из групп нулей и единиц. Подсчитать количество единиц в группах с нечетным количеством символов.

6. Дана строка, состоящая из букв, цифр, запятых, точек, знаков "+" и "-". Выделить подстроку, которая соответствует записи целого числа (т.е.

начинается со знака “+” или “-” и внутри подстроки нет букв, запятых и точек).

7. Дана строка символов, состоящая из букв, цифр, запятых, точек, знаков “+” и “-”. Выделить подстроку, которая соответствует записи вещественного числа с фиксированной точкой

8. Дана строка символов, состоящая из букв, цифр, запятых, точек, знаков “+” и “-”. Выделить подстроку, которая соответствует записи вещественного числа с плавающей точкой

9. Дана строка символов, состоящая из произвольных десятичных цифр, разделенных пробелами. Вывести на экран числа этой строки в порядке возрастания их значений.

10. Дана строка символов, состоящая из произвольных десятичных цифр, разделенных пробелами. Вывести четные числа этой строки.

11. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Вывести на экран слова этого текста в порядке, соответствующем латинскому алфавиту.

12. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Вывести на экран порядковый номер слова, накрывающего  $k$ -ю позицию (если на  $k$ -ю позицию попадает пробел, то номер предыдущего слова).

13. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Разбить исходную строку на две подстроки, причем первая длиной  $k$ -символов (если на  $k$ -ю позицию попадает слово, то его следует отнести ко второй строке, дополнив первую пробелами до  $k$ -позиций).

14. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами.. Вывести на экран порядковый номер слова максимальной длины и номер позиции строки с которой оно начинается.

15. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Вывести на экран порядковый номер слова минимальной длины и количество символов в этом слове.

16. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. В каждом слове заменить первую букву на прописную.

17. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Удалить первые  $k$  слов из строки, сдвинув на их место последующие слова строки.

18. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами  $i$ - и  $j$ -е слова.

19. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами первую и последнюю буквы каждого слова.

20. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Заменить буквы латинского алфавита на соответствующие им буквы русского алфавита.

21. Дана строка символов  $S_1 S_2 \dots S_m$ , в которой могут встречаться цифры, пробелы, буква «Е» и знаки “+”, “-“. Известно, что первый символ  $S_1$  является цифрой. Из данной строки выделить подстроки, разделенные пробелами. Определить, является ли первая подстрока числом. Если да, то выяснить: целое или вещественное число, положительное или отрицательное.

22. Дана строка символов, содержащая некоторый текст на русском языке. Разработать программу форматирования этого текста, т.е. его разбиения на отдельные строки (по  $k$  символов в каждой строке) и выравнивания по правой границе путем вставки между отдельными словами необходимого количества пробелов.

23. Дана строка символов, содержащая некоторый текст на русском языке. Заменить буквы русского алфавита на соответствующие им буквы латинского алфавита.

24. Дана строка символов, содержащая некоторый текст. Разработать программу, которая определяет, является ли данный текст палиндромом, т.е. читается ли он слева направо так же, как и справа налево (например, «А роза упала на лапу Азора»).

25. Составить программу, которая читает построчно текст другой программы (ввести с клавиатуры) на языке C++, обнаруживает комментарии и выводит их на экран.

26. Составить программу, которая читает построчно текст другой программы (ввести с клавиатуры) на языке C++, подсчитывает количество ключевых пар символов «{» и «}» и выводит на экран соответствующее сообщение.

27. Разработать программу, которая заданное целое число от 1 до 1999 выводит на экран римскими цифрами.

28. Дан текст из заглавных латинских букв, за которым следует пробел. Определить, является ли этот текст правильной записью римскими цифрами целого числа от 1 до 999, и, если является, вывести на экран это число арабскими цифрами (в десятичной системе).

29. Дан текст из  $k$  символов. Вывести на экран только строчные русские буквы, входящие в этот текст.

30. Дан текст из  $k$  символов. Вывести на экран в алфавитном порядке все различные прописные русские буквы, входящие в этот текст.

## Лабораторная работа №8. Разработка приложений с использованием диалогов для выбора файлов и главного меню

### 1 Цель работы

Изучить правила работы с компонентами *TOpenDialog*, *TSaveDialog* и *TMainMenu*. Получить практические навыки в написании и отладке программ с использованием диалогов для выбора файлов, а также применение главного меню в приложениях.

### 2 Порядок выполнения работы

Внимательно изучить краткую теорию работы. Выполнить самостоятельно примеры задания из раздела 5. Получить задание на выполнение лабораторной работы (раздел 6) согласно своему варианту. Разработать и отладить программу. Составить и защитить отчет о лабораторной работе у преподавателя.

### 3 Содержание отчета

- наименование и цель работы;
- задание на лабораторную работу согласно варианту;
- текст программы на алгоритмическом языке C++;
- результаты работы программы.

### 4 Краткая теория

#### 4.1 Компоненты *TOpenDialog* и *TSaveDialog*

Компоненты *TOpenDialog* и *TSaveDialog* находятся на странице *Dialogs* палитры компонентов. Все компоненты этой страницы являются не визуальными, т.е. не видны в момент работы программы. Поэтому их можно разместить в любом удобном месте формы. Оба рассматриваемых компонента имеют идентичные свойства и отличаются только внешним видом.


Диалоговое окно указанных компонентов открывается после вызова метода *Execute*, который возвращает *true*, если пользователь выбрал файл и нажал «Ок», и *false*, если пользователь отказался от выбора. Например.

```
if (OpenDialog1->Execute)
{ // код, выполняющийся после выборе файла пользователем
}
```

В случае успешного завершения диалога имя выбранного файла поиска содержится в свойстве *FileName*. Для фильтрации файлов, отображаемых в окне просмотра, используется свойство *Filter*, а для задания расширения файла, в случае, если оно не задано пользователем, – свойство *DefaultExt*. Если необходимо изменить заголовок диалогового окна, используется свойство *Title*.

## 4.2 Использование компонента TMainMenu

Компонент предназначен для организации главного меню в окнах приложения.

Для создания меню, необходимо поместить компонент *TMainMenu* «» в любое место формы.

Для заполнения командами, необходимо щелкнуть два раза по компоненту *TMainMenu* и ввести в поле *Caption* надписи необходимых команд. Кроме того, можно задать «горячую» клавишу при помощи свойства «ShortCut», добавить пиктограмму в полнее *Bitmap*, установить доступность пункта меню в поле *Enabled* и другие свойства (рис. 8.1);

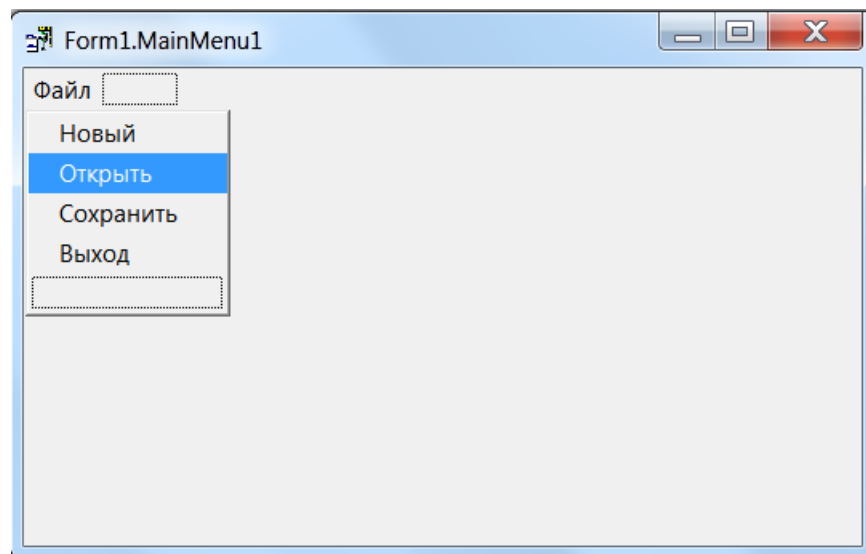


Рисунок 8.1 – Окно создания главного меню

Для создания обработчика пункт меню, необходимо в главном меню окна Вашего приложения щелкнуть на необходимый пункт меню, после чего, C++Builder создаст функцию-обработчика щелчка по данному пункту меню:

```
void __fastcall TForm1::N2Click(TObject *Sender)
{
}
```

## 5 Порядок выполнения задания

**Задание:** написать программу, вводящую в файл или читающую из файла ведомость абитуриентов, сдавших вступительные экзамены. Каждая запись должна содержать фамилию, а также оценки по математике, русскому языку, информатике.

### 5.1 Настройка компонентов *TOpenDialog* и *TSaveDialog*

Для установки компонентов *TOpenDialog* и *TSaveDialog* на форму необходимо на странице **Dialogs** палитры компонентов щелкнуть мышью со-

ответственно по пиктограммам  или  и поставить их в любое свободное место формы.

Установка фильтра производится следующим образом. Выбрав соответствующий компонент, дважды щелкнуть по правой части свойства **Filter** инспектора объектов. Появится окно **Filter Editor**, в левой части которого записывается текст, характеризующий соответствующий фильтр, а в правой части – маска.

Для **OpenDialog1** установите значения маски, как показано на рисунке 8.2. Фильтр **\*.dat** означает, что будут видны все файлы с расширением **dat**, а фильтр **\*.\*** - что будут видны все файлы (с любым именем и с любым расширением).

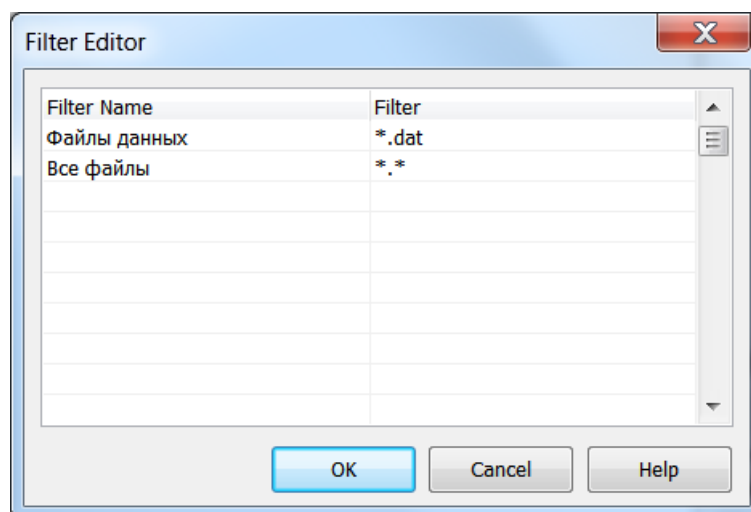


Рисунок 8.2 – Установка свойства Filter

Для того, чтобы файл автоматически записывался с расширением **dat**, в свойстве **DefaultExt** необходимо записать требуемое расширение - **dat**.

Аналогичным образом настраивается **SaveDialog1** (расширение **dat**).

## 5.2 Работа с программой

После запуска программы на выполнение на экране появится диалоговое окно программы (рис. 8.3), содержащую главное меню и панель с кнопками быстрого доступа к функциям, а также таблицу для ввода данных.

Кнопка «**Новый**» создает новую таблицу–ведомость. Открыть ранее созданный позволяет нажатие кнопки «**Открыть**». Кнопка «**Добавить**» добавляет строку в таблицу, а «**Удалить**» удаляет выбранную. Кнопка «**Сохранить**» позволяет сохранить данные в файл.

Текст программы приведен в разделе 5.3.



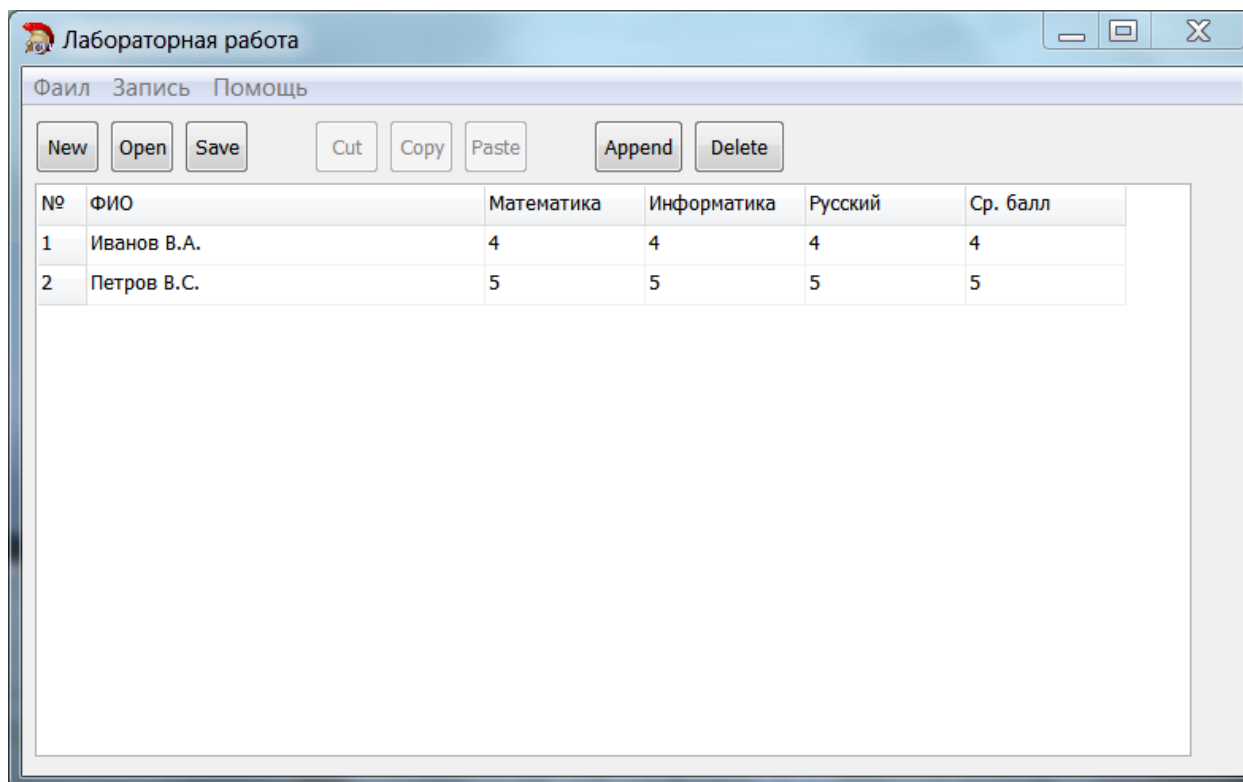


Рисунок 8.3 – Внешний вид программы

## 5.2 Текст программы

```
//-----
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{}
//-----
void __fastcall TForm1::N15Click(TObject *Sender)
{
    Application->Terminate();
}
//-----
void __fastcall TForm1::FileSaveUpdate(TObject *Sender)
{
    TAction* pa = dynamic_cast<TAction*>(Sender);
    if (pa) pa->Enabled=(StringGrid1->Visible);
}
//-----
void __fastcall TForm1::FileNewUpdate(TObject *Sender)
{
    // функция, управляющая доступностью контролирующих элементов
}
```

```

    TAction* pa = dynamic_cast<TAction*>(Sender);
    if (pa) pa->Enabled=true;
}
//-----
void __fastcall TForm1::FileNewExecute(TObject *Sender)
{ // новый файл
    StringGrid1->Visible=true;
    StringGrid1->RowCount=1;
    StringGrid1->ColWidths[0]=30;StringGrid1->ColWidths[1]=250;
    StringGrid1->ColWidths[2]=80; StringGrid1->ColWidths[3]=80;
    StringGrid1->ColWidths[4]=80; StringGrid1->ColWidths[5]=80;
    StringGrid1->Cells[0][0]="№";
    StringGrid1->Cells[1][0]="ФИО";
    StringGrid1->Cells[2][0]="Математика";
    StringGrid1->Cells[3][0]="Информатика";
    StringGrid1->Cells[4][0]="Русский";
    StringGrid1->Cells[5][0]="Ср. балл";
    AppendExecute(Sender);
    StringGrid1->FixedRows=1;
}
//-----
void __fastcall TForm1::AppendExecute(TObject *Sender)
{// Добавляет новую строку
    StringGrid1->RowCount++;
    StringGrid1->Cells[0][StringGrid1->RowCount-1]=IntToStr(StringGrid1->RowCount-1);
    for (int i=1; i<=5; i++)
        StringGrid1->Cells[i][StringGrid1->RowCount-1]="";
}
//-----
void __fastcall TForm1::DeleteUpdate(TObject *Sender)
{// функция, управляющая доступностью кнопки "удалить строку"
    Delete->Enabled=(StringGrid1->RowCount>2)*(StringGrid1->Visible);
}
//-----
void __fastcall TForm1::AppendUpdate(TObject *Sender)
{// функция, управляющая доступностью кнопки "добавить строку"
    Append->Enabled=(StringGrid1->Visible);}
//-----
void __fastcall TForm1::DeleteExecute(TObject *Sender)
{// Удаление строки и перенос выше всех других строк
    int k=StringGrid1->Row;
    if (k>0)
    {
        for (int i=k+1; i<StringGrid1->RowCount; i++)
            for (int j=0; j<StringGrid1->ColCount; j++)
                if (j==0) StringGrid1->Cells[0][i]=IntToStr(i);
                else
                    StringGrid1->Cells[j][i-1]=StringGrid1->Cells[j][i];
    }
}

```

```

        StringGrid1->RowCount--;
    }
}
//-----
void __fastcall TForm1::OpenFileExecute(TObject *Sender)
{ // Чтение из файла
    int myfile, row, col, lenStr;
    char *pszBuffer;
    if (OpenDialog1->Execute())
    {try
        { myfile = FileOpen(OpenDialog1->FileName, fmOpenRead);
          Form1->FileNewExecute(Sender);
          // Читаем из файла количество колонок и строк таблицы
          FileRead(myfile, &col, sizeof(StringGrid1->ColCount));
          FileRead(myfile, &row, sizeof(StringGrid1->RowCount));
          StringGrid1->ColCount=col; StringGrid1->RowCount=row;
          // теперь содержимое таблицы
          for (int i=0; i<col; i++)
              for (int j=0; j<row; j++)
              { // Читаем длину записанной строки
                FileRead(myfile, &lenStr, sizeof(lenStr));
                pszBuffer = new char[lenStr+1]; // создаем буфер для чтения
                // читаем саму строку
                FileRead(myfile, pszBuffer, lenStr);
                StringGrid1->Cells[i][j]=AnsiString(pszBuffer, lenStr);
                delete [] pszBuffer; // удаляем буфер для чтения
              }
            }
        catch(...)
        {
            Application->MessageBox("Ошибка чтения файла...", "Ошибка
файла", IDOK);
        }
    }
}
//-----
void __fastcall TForm1::FileSaveExecute(TObject *Sender)
{ // Запись в файл
    if (SaveDialog1->Execute())
    { int iLength;
      int myfile=FileCreate(SaveDialog1->FileName);
      // Записываем в выходной файл количество колонок и строк таблицы
      FileWrite(myfile, (char*)&(StringGrid1->ColCount),
sizeof(StringGrid1->ColCount));
      FileWrite(myfile, (char*)&(StringGrid1->RowCount),
sizeof(StringGrid1->RowCount));
      // теперь содержимое таблицы
      for (int x=0; x<StringGrid1->ColCount; x++)
          for (int y=0; y<StringGrid1->RowCount; y++)
              { // Записываем длину строки

```

```

        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(myfile, (char*)&iLength, sizeof(iLength));
        AnsiString temp = StringGrid1->Cells[x][y];
//Копируем текст ячейки во временную переменную
        FileWrite(myfile, temp.c_str(),
                StringGrid1->Cells[x][y].Length());
    }
    FileClose(myfile);
}
}
//-----
void __fastcall TForm1::HelpAboutExecute(TObject *Sender)
{
    ShowMessage("Типовая программа ЛР №8 (С) 2006 Яхонтов С.А.");
}

```

## 6 Варианты заданий для самостоятельной работы

В программе предусмотреть сохранение вводимых данных в файле и возможность чтения из ранее сохраненного файла. Результаты выводить в окно просмотра и в текстовый файл.

1. В магазине формируется список лиц, записавшихся на покупку товара повышенного спроса. Каждая запись этого списка содержит: порядковый номер, Ф.И.О., домашний адрес покупателя и дату постановки на учет. Удалить из списка все повторные записи, проверяя Ф.И.О. и домашний адрес.
2. Список товаров, имеющихся на складе, включает в себя наименование товара, количество единиц товара, цену единицы и дату поступления товара на склад. Вывести в алфавитном порядке список товаров, хранящихся больше месяца, стоимость которых превышает 1000 000 руб.
3. Для получения места в общежитии формируется список студентов, который включает Ф.И.О. студента, группу, средний балл, доход на члена семьи. Общежитие в первую очередь предоставляется тем, у кого доход на члена семьи меньше двух минимальных зарплат, затем остальным в порядке уменьшения среднего балла. Вывести список очередности предоставления мест в общежитии.
4. В справочной автовокзала хранится расписание движения автобусов. Для каждого рейса указаны его номер, тип автобуса, пункт назначения, время отправления и прибытия. Вывести информацию о рейсах, которыми можно воспользоваться для прибытия в пункт назначения раньше заданного времени.
5. На междугородной АТС информация о разговорах содержит дату разговора, код и название города, время разговора, тариф, номер телефона в

- этом городе и номер телефона абонента. Вывести по каждому городу общее время разговоров с ним и сумму.
6. Информация о сотрудниках фирмы включает: Ф.И.О., табельный номер, количество проработанных часов за месяц, почасовой тариф. Рабочее время свыше 144 часов считается сверхурочным и оплачивается в двойном размере. Вывести размер заработной платы каждого сотрудника фирмы за вычетом подоходного налога, который составляет 12% от суммы заработка.
  7. Информация об участниках спортивных соревнований содержит: наименование страны, название команды, Ф.И.О. игрока, игровой номер, возраст, рост, вес. Вывести информацию о самой молодой, рослой и легкой команде.
  8. Для книг, хранящихся в библиотеке, задаются: регистрационный номер книги, автор, название, год издания, издательство, количество страниц. Вывести список книг с фамилиями авторов в алфавитном порядке, изданных после заданного года.
  9. Различные цехи завода выпускают продукцию нескольких наименований. Сведения о выпущенной продукции включают: наименование, количество, номер цеха. Для заданного цеха необходимо вывести количество выпущенных изделий по каждому наименованию в порядке убывания количества.
  10. Информация о сотрудниках предприятия содержит: Ф.И.О., номер отдела, должность, дату начала работы. Вывести списки сотрудников по отделам в порядке убывания стажа.
  11. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит: Ф.И.О., адрес, оценки. Определить количество абитуриентов, проживающих в г. Минске и сдавших экзамены со средним баллом не ниже 4.5, вывести их фамилии в алфавитном порядке.
  12. В справочной аэропорта хранится расписание вылета самолетов на следующие сутки. Для каждого рейса указаны: номер рейса, тип самолета, пункт назначения, время вылета. Вывести все номера рейсов, типы самолетов и времена вылета для заданного пункта назначения в порядке возрастания времени вылета.
  13. У администратора железнодорожных касс хранится информация о свободных местах в поездах дальнего следования на ближайшую неделю в следующем виде: дата выезда, пункт назначения, время отправления, число свободных мест. Оргкомитет международной конференции обращается к администратору с просьбой зарезервировать  $m$  мест до города  $N$  на  $k$ -й день недели с временем отправления поезда не позднее  $t$  часов вечера. Вывести время отправления или сообщение о невозможности выполнить заказ в полном объеме.

14. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит: Ф.И.О. абитуриента, оценки. Определить средний балл по университету и вывести список абитуриентов, средний балл которых выше среднего балла по университету. Первыми в списке должны идти студенты, сдавшие все экзамены на 5.
15. В радиоателье хранятся квитанции о сданной в ремонт радиоаппаратуре. Каждая квитанция содержит следующую информацию: наименование группы изделий(телевизор, радиоприемник и т. п.),марку изделия, дату приемки в ремонт, состояние готовности заказа (выполнен, не выполнен). Вывести информацию о состоянии заказов на текущие сутки по группам изделий.
16. Разработать программу формирования ведомости об успеваемости студентов. Каждая запись этой ведомости должна содержать: номер группы, Ф.И.О. студента, оценки за последнюю сессию. Вывести списки студентов по группам. В каждой группе Ф.И.О. студентов должны быть расположены в порядке убывания среднего балла.
17. В исполкоме формируется список учета нуждающихся в улучшении жилищных условий. Каждая запись этого списка содержит: порядковый номер, Ф.И.О., величину жилплощади на одного члена семьи и дату постановления на учет. По заданному количеству квартир, выделяемых по данному списку в течение года, вывести весь список с указанием ожидаемого года получения квартиры.
18. Имеется список женихов и список невест. Каждая запись списка содержит пол, имя, возраст, рост, вес, а также требования к партнеру: наименьший и наибольший возраст, наименьший и наибольший вес, наименьший и наибольший рост. Объединить эти списки в список пар с учетом требований к партнерам без повторений женихов и невест.
19. В библиотеке имеется список книг. Каждая запись этого списка содержит: фамилии авторов, название книги, год издания. Вывести информацию о книгах, в названии которых встречается некоторое ключевое слово (ввести с клавиатуры).
20. В магазине имеется список поступивших в продажу автомобилей. Каждая запись этого списка содержит: марку автомобиля, стоимость, расход топлива на 100 км, надежность (число лет безотказной работы), комфортность (отличная, хорошая, удовлетворительная). Вывести перечень автомобилей, удовлетворяющих требованиям покупателя, которые вводятся с клавиатуры в виде некоторого интервала допустимых значений.
21. Каждая запись списка вакантных рабочих мест содержит: наименование организации, должность, квалификацию (разряд или образование), стаж работы по специальности, заработную плату, наличие социального страхования (да/нет), продолжительность ежегодного оплачиваемого отпуска. Вывести список рабочих мест в соответствии с требованиями клиента.

22. В технической службе аэропорта имеется справочник, содержащий записи следующей структуры: тип самолета, год выпуска, расход горючего на 1000 км. Для определения потребности в горючем техническая служба запрашивает расписание полетов. Каждая запись расписания содержит следующую информацию: номер рейса, пункт назначения, дальность полета. Вывести суммарное количество горючего, необходимое для обеспечения полетов на следующие сутки.
23. Для участия в конкурсе на замещение вакантной должности сотрудника фирмы желающие подают следующую информацию: Ф.И.О., год рождения, образование(среднее, специальное, высшее), знание иностранных языков(английский, немецкий, французский, владею свободно, читаю и перевожу со словарем), владение компьютером, стаж работы, наличие рекомендаций. Вывести список претендентов в соответствии с требованиями руководства фирмы.
24. При постановке на учет в ГИБДД автолюбители указывают следующие данные: марка автомобиля, год выпуска, номер двигателя, номер кузова, цвет, номерной знак, Ф.И.О и адрес владельца. Вывести список автомобилей, проходящих техосмотр в текущем году, сгруппированных по маркам автомобилей. Учесть, что если текущий год четный, техосмотр проходят автомобили с четными номерами двигателей, иначе – с нечетными номерами.
25. Для участия в конкурсе исполнителей необходимо заполнить следующую анкету: Ф.И.О., год рождения, название страны, класс музыкального инструмента (гитара, фортепиано, скрипка, виолончель). Вывести список самых молодых лауреатов конкурса по классам инструментов в порядке занятых мест.
26. Список группы студентов содержит следующую информацию: Ф.И.О., рост и вес. Вывести Ф.И.О. студентов, рост и вес которых чаще всего встречаются в списке.
27. Список группы студентов содержит следующую информацию: Ф.И.О., рост и вес. Вывести Ф.И.О. студентов, рост и вес которых являются в списке уникальными.

## Лабораторная работа №9. Разработка приложений с использованием средств для отображения графической информации

### 1 Цель работы

Изучить возможности построения графиков с помощью компонента отображения графической информации *TChart*. Получить практические навыки в написании и отладке программ построения на экране графика заданной функции.

### 2 Порядок выполнения работы

Внимательно изучить краткую теорию работы. Выполнить самостоятельно примеры задания из раздела 5. Получить задание на выполнение лабораторной работы (раздел 6) согласно своему варианту. Разработать и отладить программу. Составить и защитить отчет о лабораторной работе у преподавателя.

### 3 Содержание отчета

- наименование и цель работы;
- задание на лабораторную работу согласно варианту;
- текст программы на алгоритмическом языке *C++*;
- результаты работы программы.

### 4 Краткая теория

#### 4.1 Построение графика с помощью компонента *TChart*

Обычно результаты расчетов представляются в виде графиков и диаграмм. Система *C++Builder* имеет мощный пакет стандартных программ вывода на экран и редактирования графической информации, который реализуется с помощью визуально отображаемого на форме компонента *TChart* (рисунк 9.1).

Построение графика (диаграммы) производится после вычисления таблицы значений функции  $y=f(x)$  на интервале  $[Xmin, Xmax]$  с заданным шагом. Полученная таблица передается в специальный двумерный массив *Series<sub>k</sub>* (*k* – номер графика, например, *Series1*) компонента *TChart* с помощью метода *Add*.

Компонент *TChart* осуществляет всю работу по отображению графиков, переданных в объект *Series<sub>k</sub>*: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде всевозможных графиков или диаграмм. При необходимости, с помощью встроенного редактора *EditingChart* компоненту *TChart*



передаются данные о толщине, стиле и цвете линий, параметрах шрифта подписей, шагах разметки координатной сетки и другие настройки.

В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам компонента **TChart**. Так, например, свойство **Chart1->BottomAxis** содержит значение максимального предела нижней оси графика и при его изменении во время работы программы автоматически изменяется изображение графика (см. нижеприведенную программу).

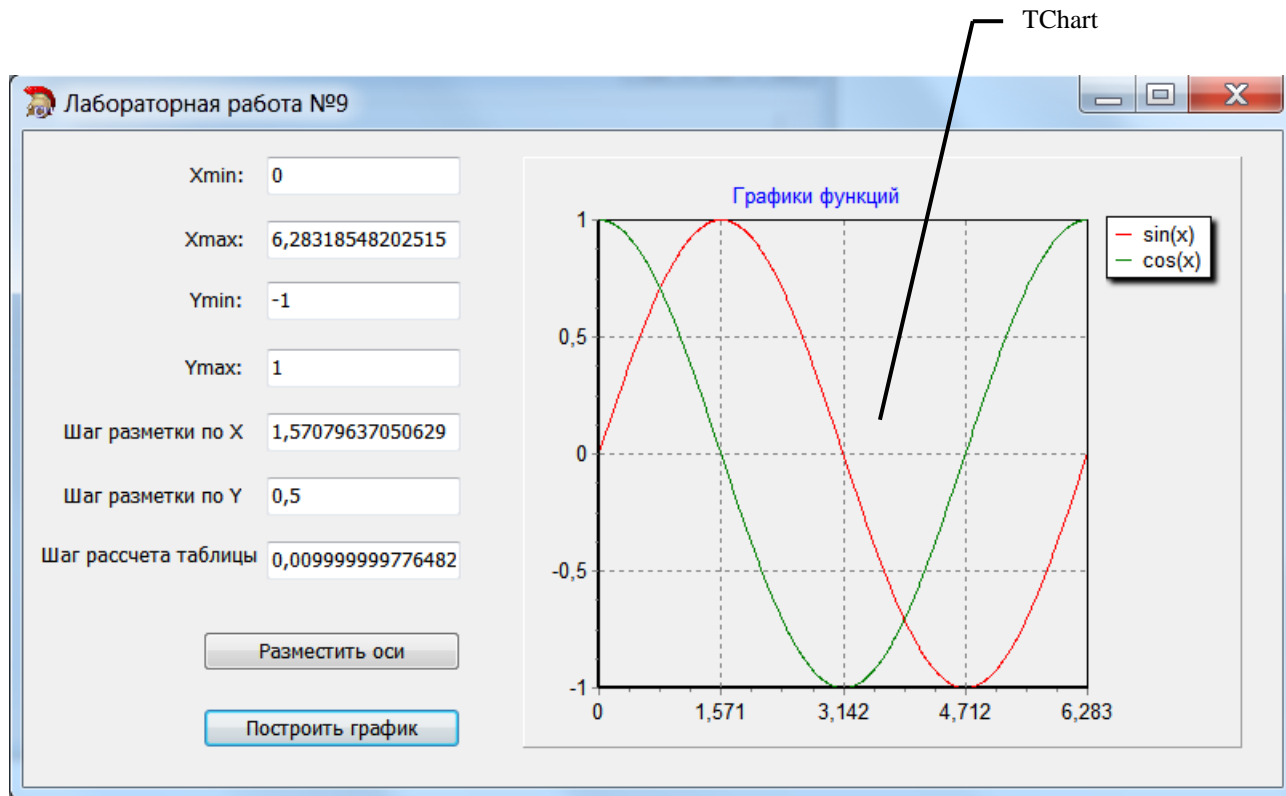



Рисунок 9.1 – Использование компонента TChart

## 5 Пример написания программы

**Задание:** составить программу, отображающую графики функций  $\sin(x)$  и  $\cos(x)$  на интервале  $[Xmin, Xmax]$ . Предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

### 5.1 Настройка формы

Панель диалога программы организуется в виде, представленном на рисунке 9.1.

Для ввода исходных данных используются окна редактирования **TEdit**. Компонент **TChart** вводится в форму путем нажатия пиктограммы «» в палитре компонентов на странице **TeeChart Std**.

### 5.2 Работа с компонентом TChart

Для изменения параметров компонента **TChart** необходимо дважды щелкнуть по нему мышью в окне формы. Появится окно редактирования **EditingChart1** (рисунок 9.2).

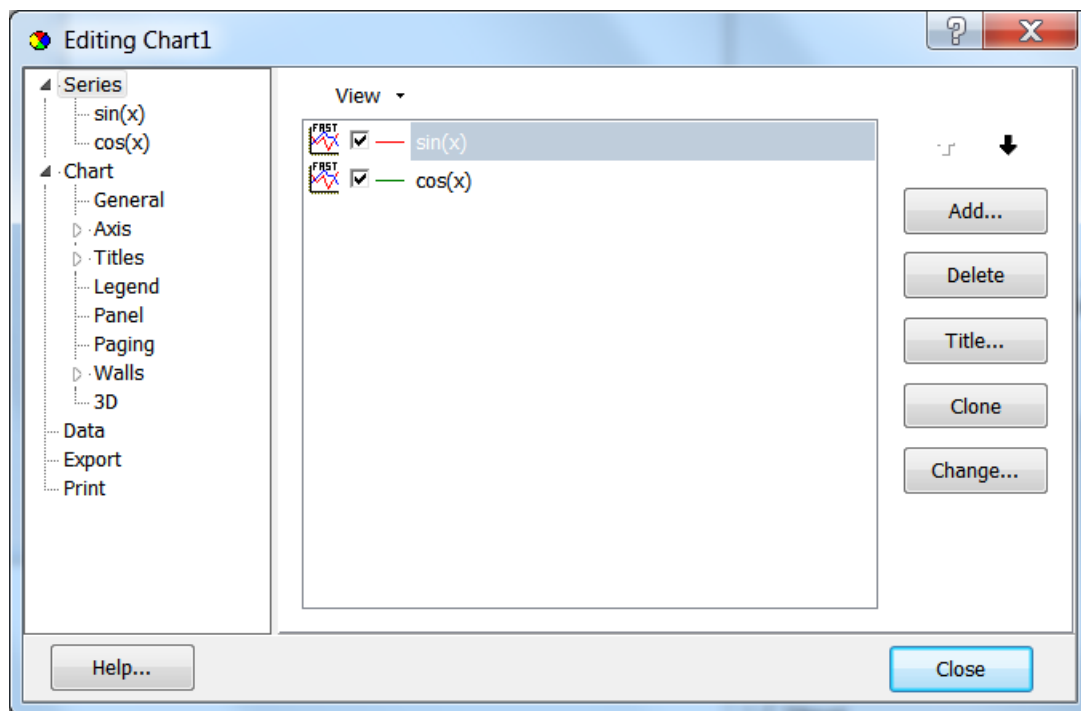


Рисунок 9.2 – Окно редактирования EditingChart1

Для создания нового объекта **Series1** щелкнуть по кнопке **Add** на странице **Series**. В появившемся диалоговом окне **TeeChart Gallery** выбрать пиктограмму с надписью **Line** (график выводится в виде линий). Если нет необходимости представления графика в трехмерном виде, отключить независимый переключатель **3D**.

После нажатия на кнопку **OK** появится новая серия с название **Series1**. Для изменения названия нажать кнопку **Title**. В появившемся однострочном редакторе набрать имя отображаемой функции – «**sin(x)**». Аналогичным образом создать объект **Series2** для функции **cos(x)**.

Для изменения надписи над графиком на странице **Titles** в многострочном редакторе набрать: «**Графики функций**».

Для разметки осей необходимо выбрать страницу **Axis** и научиться устанавливать параметры настройки осей.

Нажимая различные кнопки меню, познакомиться с другими возможностями **EditingChart**.

### 5.3 Написание кода обработки события открытия формы

В данном месте программы устанавливаются начальные пределы и шаг разметки координатных осей.

Когда свойство *Chart1->BottomAxis->Automatic* имеет значения *false*, автоматическая установка параметров осей не работает.

#### 5.4 Написание кода обработки событий нажатия на кнопки

Функция *TForm1::Button1Click* обрабатывает нажатие кнопки «*Установить оси*».

Функция *TForm1::Button2Click* обрабатывает нажатие кнопки «*Построить график*».

Для добавления координат точек из таблицы в двумерный массив объекта *Series1* используется функция

*int Series1->AddXY(AXValue;AYValue;AXLabel;AColor)* ,

где *AXValue*, *AYValue* – координаты точки по осям *X* и *Y*;

*AXLabel* – текстовая надпись добавленной точки;

*AColor* задает цвет линий (если равен *clTeeColor*, то принимается цвет, определенный при проектировании формы).

#### 5.5 Текст программы

```
#include <math.h>
float   Xmin,Xmax,Ymin,Ymax,Hx,Hу,h;
void __fastcall TForm1::FormShow(TObject *Sender)
{ // Установка начальных параметров координатных осей
  Xmin=0;  Xmax=2*M_PI;  Ymin=-1;  Ymax=1;
  Hx=M_PI/2;  Hу=0.5;
  h=0.01; // Установка шага расчета таблицы
  // Вывод данных в окна однострочных редакторов
  Edit1->Text=FloatToStr(Xmin);
  Edit2->Text=FloatToStr(Xmax);
  Edit3->Text=FloatToStr(Ymin);
  Edit4->Text=FloatToStr(Ymax);
  Edit5->Text=FloatToStr(Hx);
  Edit6->Text=FloatToStr(Hу);
  Edit7->Text=FloatToStr(h);
  // Отключение автоматического определения
  // параметров нижней оси
  Chart1->BottomAxis->Automatic=false;
  // Установка левой границы нижней оси
  Chart1->BottomAxis->Minimum=Xmin;
  // Установка правой границы нижней оси
  Chart1->BottomAxis->Maximum=Xmax;
  // Отключение автоматического
  // определения параметров левой оси
  Chart1->LeftAxis->Automatic=false;
  // Установка нижней границы левой оси
  Chart1->LeftAxis->Minimum=Ymin;
  // Установка верхней границы левой оси
  Chart1->LeftAxis->Maximum=Ymax;
```

```

// Установка шага разметки по нижней оси
Chart1->BottomAxis->Increment=Hx;
// Установка шага разметки по левой оси
Chart1->LeftAxis->Increment=Hy;
}
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{ //Чтение данных из окон однострочных редакторов
  Xmin=StrToFloat(Edit1->Text);
  Xmax=StrToFloat(Edit2->Text);
  Ymin=StrToFloat(Edit3->Text);
  Ymax=StrToFloat(Edit4->Text);
  Hx=StrToFloat(Edit5->Text);
  Hy=StrToFloat(Edit6->Text);
// Установка левой границы нижней оси
Chart1->BottomAxis->Minimum=Xmin;
// Установка правой границы нижней оси
Chart1->BottomAxis->Maximum=Xmax;
// Установка нижней границы левой оси
Chart1->LeftAxis->Minimum=Ymin;
// Установка верхней границы левой оси
Chart1->LeftAxis->Maximum=Ymax;
// Установка шага разметки по нижней оси
Chart1->BottomAxis->Increment=Hx;
// Установка шага разметки по левой оси
Chart1->LeftAxis->Increment=Hy;
}
//-----
void __fastcall TForm1::Button2Click(TObject *Sender)
{ float x,y1,y2;
  // Очистка графиков
  Series1->Clear();
  Series2->Clear();
  Xmin=StrToFloat(Edit1->Text);
  Xmax=StrToFloat(Edit2->Text);
  h=StrToFloat(Edit7->Text); // Шаг расчета таблицы для графика
  x=Xmin; // Начальное значение по оси X
  while (x<Xmax)
  { y1=sin(x); // Расчет функции
    Series1->AddXY(x,y1,"",clTeeColor); //Вывод точки на график
    y2=cos(x); // Расчет функции
    Series2->AddXY(x,y2,"",clTeeColor); //Вывод точки на график
    x=x+h; // Увеличение значения X на величину шага
  }
}

```

## 6 Варианты заданий для самостоятельной работы

Постройте графики функций для соответствующих вариантов. Таблицу данных получить изменяя параметр  $X$  с шагом  $h$ . Ввод исходных данных организовать через окно **TEdit**. Самостоятельно выбрать удобные параметры настройки.

1.  $t = \frac{2\cos\left(x - \frac{\pi}{6}\right)}{0.5 + \sin^2 y} \left(1 + \frac{z^2}{3 - z^2/5}\right)$ . При  $y = -1.22$ ,  $z = 3.5 \times 10^{-2}$ ,  $14 \leq x \leq 28$ ,  $\Delta x = 2$ .

2.  $u = \frac{\sqrt[3]{8 + |x - y|^2 + 1}}{x^2 + y^2 + 2} - e^{|x-y|} (tg^2 z + 1)^x$ .

При  $-4 \leq x \leq 8$ ,  $\Delta x = 0.5$ ,  $y = 0.75 \times 10^{-4}$ ,  $z = 0.845 \times 10^2$

3.  $v = \frac{1 + \sin^2(x + y)}{\left|x - \frac{2y}{1 + x^2 y^2}\right|} x^{|y|} + \cos^2\left(\arctg \frac{1}{z}\right)$ .

При  $0.37 \leq x \leq 2.8$ ,  $\Delta x = .7$ ,  $y = -0.825$ ,  $z = 0.16 \times 10^2$ .

4.  $w = |\cos x - \cos y|^{(1 + 2 \sin^2 y)} \left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4}\right)$ .

При  $1 \leq x \leq 8$ ,  $\Delta x = 2$ ,  $y = -0.875$ ,  $z = -0.475 \times 10^{-3}$ .

5.  $\alpha = \ln\left(y^{-\sqrt{|x|}}\right) \left(x - \frac{y}{2}\right) + \sin^2 \arctg(z)$ .

При  $-14 \leq x \leq 4.8$ ,  $\Delta x = .6$ ,  $y = 4.642 \times 10^{-2}$ ,  $z = 20.001 \times 10^2$ .

6.  $\beta = \sqrt{10(\sqrt[3]{x} + x^{y+2})} (\arcsin^2 z - |x - y|)$ .

При  $x = 16.55 \times 10^{-3}$ ,  $y = -2.75$ ,  $z = 0.15$   $\beta = -38.902$ .

7.  $\gamma = 5 \arctg(x) - \frac{1}{4} \arccos(x) \frac{x + 3|x - y| + x^2}{|x - y|z + x^2}$ .

При  $x = 0.1722$ ,  $y = 6.33$ ,  $z = 3.25 \times 10^{-4}$   $\gamma = -172.025$ .

8.  $\varphi = \frac{e^{|x-y|} |x - y|^{x+y}}{\arctg(x) + \arctg(z)} + \sqrt[3]{x^6 + \ln^2 y}$ .

При  $x = -2.235 \times 10^{-2}$ ,  $y = 2.23$ ,  $z = 15.221$   $\varphi = 39.374$ .

$$9. \psi = \left| x^{\frac{y}{x}} - \sqrt[3]{\frac{y}{x}} \right| + (y-x) \frac{\cos y - \frac{z}{(y-x)}}{1 + (y-x)^2}.$$

При  $x=1.825 \times 10^2$ ,  $y=18.225$ ,  $z=-3.298 \times 10^{-2}$   $\psi=1.2131$ .

$$10. a = 2^{-x} \sqrt{x + 4\sqrt{y}} \sqrt[3]{e^{x-1/\sin z}}.$$

При  $x=3.981 \times 10^{-2}$ ,  $y=-1.625 \times 10^3$ ,  $z=0.512$   $a=1.26185$ .

$$11. b = y^{\sqrt[3]{|x|}} + \cos^3(y) \frac{|x-y| \left( 1 + \frac{\sin^2 z}{\sqrt{x+y}} \right)}{e^{|x-y|} + \frac{x}{2}}.$$

При  $x=6.251$ ,  $y=0.827$ ,  $z=25.001$   $b=0.7121$ .

$$12. c = 2^{(y^x)} + (3^x)^y - \frac{y \left( \arctg z - \frac{\pi}{6} \right)}{|x| + \frac{1}{y^2 + 1}}.$$

При  $x=3.251$ ,  $y=0.325$ ,  $z=0.466 \times 10^{-4}$   $c=4.025$ .

$$13. f = \frac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x-y|(\sin^2 z + tgz)}.$$

При  $x=17.421$ ,  $y=10.365 \times 10^{-3}$ ,  $z=0.828 \times 10^5$   $f=0.33056$ .

$$14. g = \frac{y^{x+1}}{\sqrt[3]{|y-2|} + 3} + \frac{x + \frac{y}{2}}{2|x+y|} (x+1)^{-1/\sin z}.$$

При  $x=12.3 \times 10^{-1}$ ,  $y=15.4$ ,  $z=0.252 \times 10^3$   $g=82.8257$ .

$$15. h = \frac{x^{y+1} + e^{y-1}}{1 + x|y - tgz|} \left( 1 + |y-x| \right) + \frac{|y-x|^2}{2} - \frac{|y-x|^3}{3}.$$

При  $x=2.444$ ,  $y=0.869 \times 10^{-2}$ ,  $z=-0.13 \times 10^3$   $h=-0.49871$ .

$$16. y = \frac{x^{\cos x}}{|x + e^x| + tgx}, \text{ при } 0.4 \leq x \leq 6, \Delta x = 0.7.$$

$$17. y = \frac{2 \cos(x - \pi/6)}{1/2 + \sin^2(1 + \frac{x^2}{3})}$$

При  $0 \leq x \leq 1$ ,  $\Delta x = 0.1$ .

$$18. y = x \sin x^3 + x + \frac{x^2}{2!} + \frac{x^3}{3!}$$

При  $1 \leq x \leq 3$ ,  $\Delta x = 0.2$ .

$$19. y = \left| \cos^2 x \right| \frac{x - \frac{2}{x-2}}{1 + (1-x)^2}$$

При  $-1 \leq x \leq 1$ ,  $\Delta x = 0.2$ .

$$20. y = e^{-2x} \sin(2x+1) - \sqrt{|x+2|}$$

При  $-0.5 \leq x \leq 4.5$ ,  $\Delta x = 0.5$ .

$$21. y = \sqrt{x^2 + 5} - 10 \sin^3(x+1)$$

При  $-2 \leq x \leq 8$ ,  $\Delta x = 1$ .

$$22. y = e^{-x} \frac{\operatorname{tg} x^2 + x^3}{x \cos^2 2x}$$

При  $0.05 \leq x \leq 0.55$ ,  $\Delta x = 0.05$ .

$$23. y = \frac{x^2(x+1)}{2} - \sin^2 \sqrt{x+2}$$

При  $-1 \leq x \leq 1$ ,  $\Delta x = 0.2$ .

$$24. y = \frac{x^2 + 1}{\sin 3x} + \frac{\sqrt{x/2}}{\cos 3x}$$

При  $0.2 \leq x \leq 2.2$ ,  $\Delta x = 0.2$ .

$$25. y = \sin^3(x^2 + \pi/3)^2 - \sqrt{\frac{x}{2}}$$

При  $2 \leq x \leq 12$ ,  $\Delta x = 1$ .

$$26. y = 10^{-x} \sqrt{|\sin 2x + \cos x^2|}$$

При  $0 \leq x \leq 3$ ,  $\Delta x = 0.3$ .

$$27. y = \frac{e^{-\frac{x}{2}} \operatorname{tg}^2 3x}{1 + |\cos x^3|}$$

При  $-2.3 \leq x \leq -1.8$ ,  $\Delta x = 0.05$ .

$$28. y = 2 \operatorname{tg}^2 x - \frac{1}{\frac{1}{2} \sin^2 \frac{x}{2}}$$

При  $2.4 \leq x \leq 3.4$ ,  $\Delta x = 0.1$ .

$$29. y = \ln(1+x^2) + \sin^2\left(\frac{x}{3}\right)$$

При  $-2 \leq x \leq 1$ ,  $\Delta x = 0.3$ .

$$30. y = \frac{2^{3x} + 10^{-x} \cos(x + \pi/3)}{x \sin x}$$

При  $1.2 \leq x \leq 2.2$ ,  $\Delta x = 0.1$ .

## **Лабораторная работа №10. Разработка приложений с использованием доступа к таблицам баз данных**

### **1 Цель работы**

Изучить возможности разработки приложений для доступа к таблица базы данных. Получить практические навыки в написании и отладке программ для работы с таблицами базы данных.

### **2 Порядок выполнения работы**

Внимательно изучить краткую теорию работы. Выполнить самостоятельно примеры задания из раздела 4 и 5. Получить задание на выполнение лабораторной работы (раздел 6) согласно своему варианту. Разработать и отладить программу. Составить и защитить отчет о лабораторной работе у преподавателя.

### **3 Содержание отчета**

- наименование и цель работы;
- задание на лабораторную работу согласно варианту;
- текст программы на алгоритмическом языке C++;
- результаты работы программы.

### **4 Краткая теория**

#### **4.1 Основы реляционных баз данных**

Когда возникает потребность хранить, периодически обновлять и анализировать большие объёмы структурированных данных, то можно использовать картотеку, хранящую записи для каждого студента, таблицу с перечнем всех студентов и предметов и т.д. Однако на практике для таких целей чаще всего используются базы данных (БД). Информация в БД хранится в табличном виде. БД — это, прежде всего, набор таблиц, хотя, в базу данных могут входить так же процедуры и ряд других объектов. Таблица БД представляет собой обычную двумерную таблицу с характеристиками (атрибутами) какого-то множества объектов. Таблица имеет имя - идентификатор, по которому на неё можно сослаться. Например, таблица для хранения информации о студентах может быть представлена в следующем виде (табл. 10.1).



**Таблица 10.1 – Данные о студентах**

№ зач. книжки	Группа	Фамилия	Имя	Отчество	Год рождения	Пол	Предмет	Оценка
Zach	Group	Fam	Nam	Par	Year_b	Sex	Pred	Oz
115261	A962	Иванов	Егор	Петрович	1976	М	Физика	4
234582	Э973	Петров	Иван	Сидорович	1978	М	Физика	5
365484	M992	Сидоров	Иван	Петрович	1979	М	Химия	3
498759	A971	Егорова	Инна	Егоровна	1977	Ж	ТАУ	4
...	...	...	...	...	...	...	...	...

Столбцы таблицы соответствуют тем или иным характеристикам объектов - **полям**.

Каждое поле характеризуется **именем и типом** хранящихся данных. **Имя поля** - это идентификатор, который используется в программах для манипуляции данными. Он строится по тем же правилам, как любой идентификатор, т.е. пишется латинскими буквами, состоит из одного слова и т.д. Таким образом, имя - это не то, что отображается на экране или в отчёте заголовка столбца (это отображение можно писать по-русски), а идентификатор, соответствующий этому заголовку. Например, для таблицы введём для последующих ссылок имена полей **Zach, Group, Fam, Nam, Par, Year\_b, Sex, Pred, Oz**, соответствующие указанным в ней заголовкам полей.

**Тип поля** характеризует тип хранящихся в поле данных. Это могут быть строки, числа, булевы значения, большие тексты, изображения и т.п.

Каждая строка таблицы соответствует одному из объектов. Она называется **записью** и содержит значения всех полей, характеризующих данный объект.

При построении таблиц БД важно обеспечить непротиворечивость информации. Это делается введением **ключевых полей**, обеспечивающих уникальность каждой записи. Ключевым может быть одно или несколько полей. В приведённом примере можно было бы сделать ключевыми совокупность полей **Fam, Nam** и **Par**. Но в этом случае нельзя было бы заносить в таблицу сведения о полных однофамильцах, у которых совпадают фамилия, имя и отчество. Поэтому целесообразнее использовать поле **Zach** – номер зачётной книжки, которое можно сделать ключевым, поскольку номер зачётной книжки не может быть одинаковым у двух студентов.

В каждый момент времени есть некоторая текущая запись, с которой ведётся работа. Записи в таблице базы данных физически могут располагаться без какого-либо порядка, просто в последовательности их ввода (появления новых студентов). Но когда данные таблицы предъявляются пользователю, они должны быть упорядочены. Для упорядочения данных в БД, так же, как и для упорядочения данных в массивах, используется понятие **индекса**.

Индекс показывает, в какой последовательности будет просматриваться таблица. Часто индексы хранятся отдельно от файла с данными. Для быстрой сортировки обычно индексы выносят в отдельный индексный файл с тем же названием, но с другим расширением. В таком файле содержатся только первичные ключи и описание очередности записей. Основной же файл при этом остается неупорядоченным. Т. к. индексный файл намного меньше файла объектов, сортировка по индексу происходит гораздо быстрее, чем сортировка основного файла (рис. 10.1).

Индексы могут быть *первичными и вторичными*. Например, первичным индексом могут служить поля, отмеченные при создании таблицы как *ключевые*. А вторичные индексы могут создаваться для других полей как при создании таблицы, так и впоследствии. Вторичным индексам присваиваются идентификаторы, по которым их можно использовать.

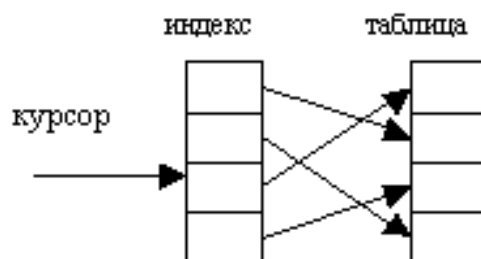


Рисунок 10.1 – Перемещение курсора по индексу

Если индекс включает в себя несколько полей, то упорядочение БД сначала осуществляется по первому полю, а для записей, имеющих одинаковое значение первого поля, по второму полю. Например, таблицу о студентах можно проиндексировать по группам, а внутри каждой группы по алфавиту.

## 4.2 Создание таблиц при помощи Microsoft Office Access

*Microsoft Office Access* – приложение предназначенное для работы с базами данных, которое входит в пакет продуктов *Microsoft Office*.

Создаем пустую базу данных. При создании, утилита предлагает выбрать адрес сохранения и имя БД (вводим имя, например, *stud.accdb*).

Затем необходимо перейти в режим “конструктор”. и удалить поле “код”, для упрощения таблицы и предотвращения конфликтов с программой, в которой будет использоваться эта таблица.

Для примера БД студентов учебного заведения структура файла таблицы может быть представлена в виде (рис. 10.1).

[illegible]

Рисунок 10.1 – Диалог конструктора

После задания структуры таблицы, её сохранение производится выбором кнопки “Сохранить”. В диалоге указывается имя файла таблицы (например, *stud*).

### 4.3 Разработка приложения для работы с таблицей данных

Для разработки системы управления базой данных можно воспользоваться как стандартными СУБД (*Access, Paradox, dBase* и т.д.), так и разработать самостоятельное приложение для работы с БД.

Разработка самостоятельного приложения для работы с БД имеет ряд преимуществ по сравнению с использованием готовых пакетов:

- ресурсы ОС задействованы только для тех задач, которые будут реализованы в СУБД;
- полученные программные модули очень компактны;
- обеспечивается более высокое быстродействие вследствие использования только необходимых для решения задачи операций с БД;

- многообразие реализаций максимально удобного интерфейса пользователя с СУБД;
- возможность интегрироваться в любые программные приложения;
- осуществление расчётных задач сложности, которая не может быть обеспечена применением готовых СУБД.

Для программирования баз данных используются следующие вкладки палитры компонент: *Data Access*, *Data Control* и *dbGo* (рис. 10.1 - 10.3).

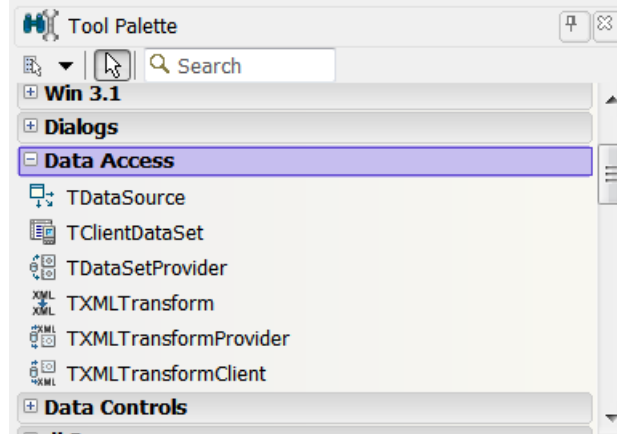


Рисунок 10.1 – Вкладка *Data Access*

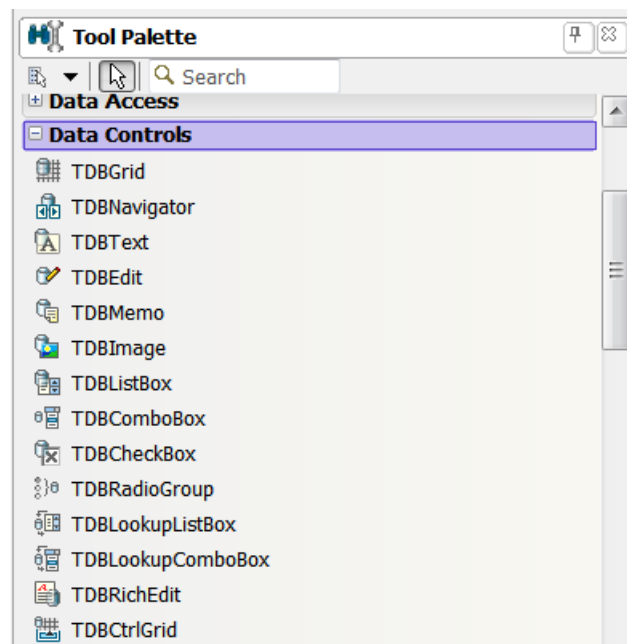


Рисунок 10.2 – Вкладка *Data Control*

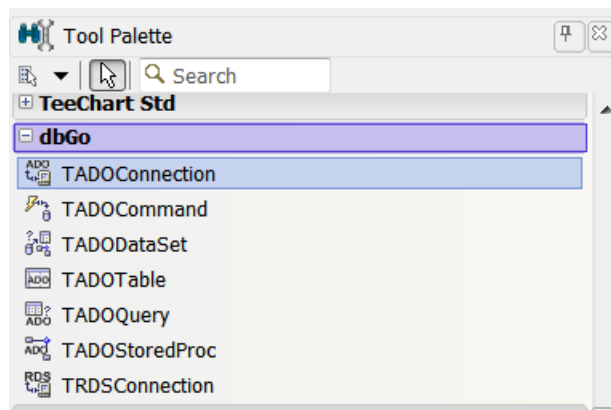


Рисунок 10.3 – Вкладка *dbGo*

Вкладка *dbGo* содержит компоненты:

- *ADOTable* - универсальный способ подключения к базам данных.

Вкладка *Data Access* содержит компоненты:

- *DataSource* - связывает компоненты отображения информации с компонентом *ADOTable*.

Вкладка *Data Control* содержит компоненты:

- *DBGrid* - таблица для отображения и редактирования записей базы данных.
- *DBNavigator* - панель для управления данными (удаление, добавление, перемещение и т. д.)

Компоненты вкладки *Data Access* и *dbGo* являются служебными и относятся к невидимым (т. е. отображаются на стадии разработки приложения и не отображаются в процессе выполнения), а *Data Control* к отображаемым как в процессе разработки, так и в процессе выполнения.

После переноса компонент на форму производится настройка свойств объектов с использованием *Object Inspector*.

Основные свойства объекта *ADOTable*:

- *Name* - имя объекта, используемое в программе (является указателем, заданным в соответствии с шаблоном компоненты);
- *ConnectionString* - это строка, которая содержит информацию, необходимую для подключения к источнику данных;
- *TableName* - имя файла таблицы (выбирается из списка, если подключение к базе данных через *ConnectionString* прошло успешно);
- *Active* - позволяет отображать данные из таблицы в процессе разработки и автоматически открывает файл при выполнении (когда установлено *true*).

Служебная компонента *DataSource* выполняет роль связующего звена и имеет следующие свойства:

- *Name* - имя объекта, используемое в программе (является указателем, заданным в соответствии с шаблоном компоненты);

- ***DataSet*** - имя таблицы, связанной с данным объектом (выбирается из списка).

Объект ***ADOTable*** позволяет приложению работать с таблицей, но для отображения данных на экране необходимо использовать компоненту ***DBGrid***.

Основные свойства объекта ***DBGrid***:

- ***Name*** - имя объекта, используемое в программе (является указателем, заданным в соответствии с шаблоном компоненты);
- ***DataSource*** – имя источника данных, посредством которого осуществляется связь объекта ***DBGrid*** с объектом ***ADOTable*** (выбирается из списка).

Объект ***DBGrid*** позволяет не только отображать данные, но и управлять ими (перемещаться по записям, добавлять и редактировать записи и т.д.). Однако ряд действий эффективней выполнять с использованием объекта ***DBNavigator***.

Основные свойства объекта ***DBNavigator***:

- ***Name*** – имя объекта (указатель);
- ***DataSource*** – имя источника данных, посредством которого осуществляется управление компонентой ***ADOTable***.

На рис. 10.4 показана структурная схема взаимодействия компонент приложения для работы с базой данных:

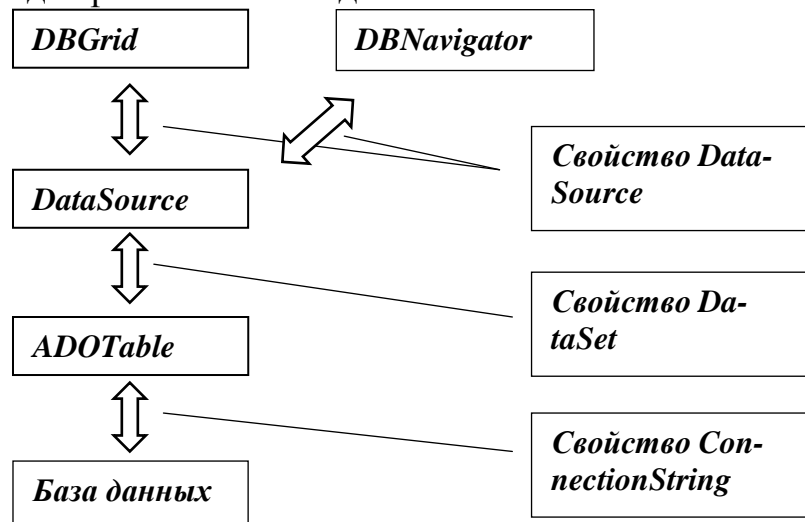


Рисунок 10.4 – Взаимодействие компонент приложения

Через меню **Structure** можно изменять свойства каждого столбца, а при помощи клавиши **Shift** можно изменить общие свойства всех столбцов сразу (например **DisplayWidth** – ширины).

После размещения на форме всех необходимых компонент и настройки их свойств получим макет приложения, показанный на рис. 10.5.

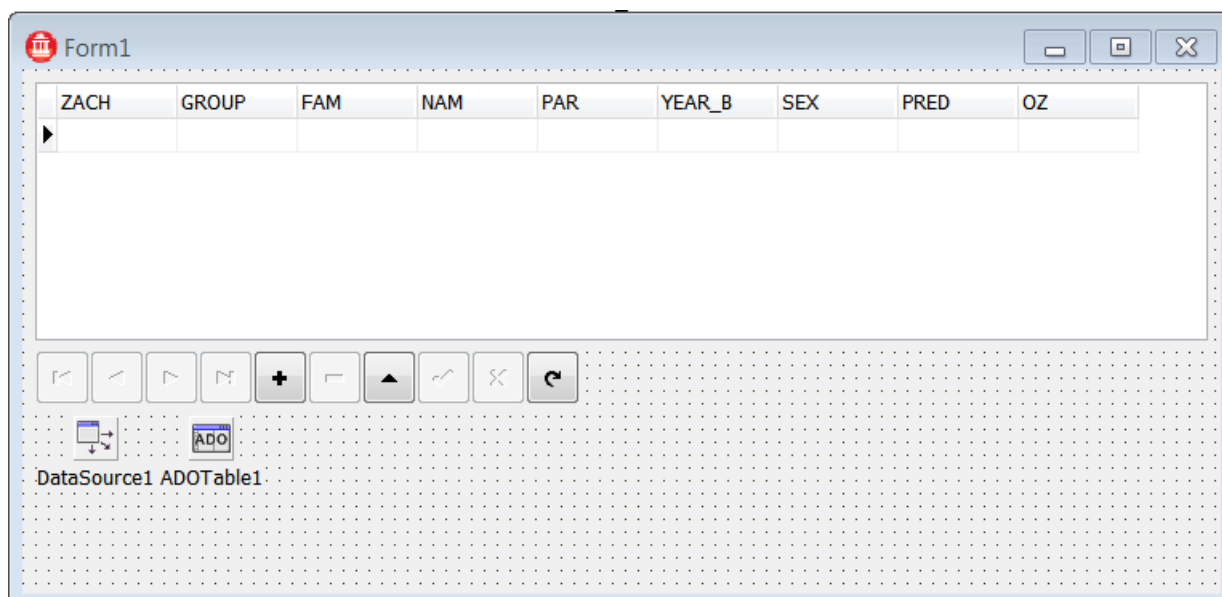


Рисунок 10.5 – Макет формы приложения для работы с таблицей *stud*

После компиляции и запуска приложения возможна работа с исходной однотабличной БД – ввод, сохранение, удаление и редактирование данных, перемещение по записям таблицы. Вводимая в поля таблицы информация должна соответствовать указанному при создании таблицы типу поля и не превышать длину поля (в этом случае ввод символов за границей длины поля будет невозможен). Значение свойств компонентов необходимо установить в соответствии с таблицей 10.3.

**Таблица 10.3 – Свойства компонент приложения**

Компонента	Свойство	Значение
ADOTable	Name	Table1
	TableName	Stud.dbf
	Active	True
DataSource	Name	DataSource1
	DataSet	Table1
DBGrid	Name	DBGrid1
	DataSource	DataSource1
DBNavigator	Name	DBNavigator1
	DataSource	DataSource1

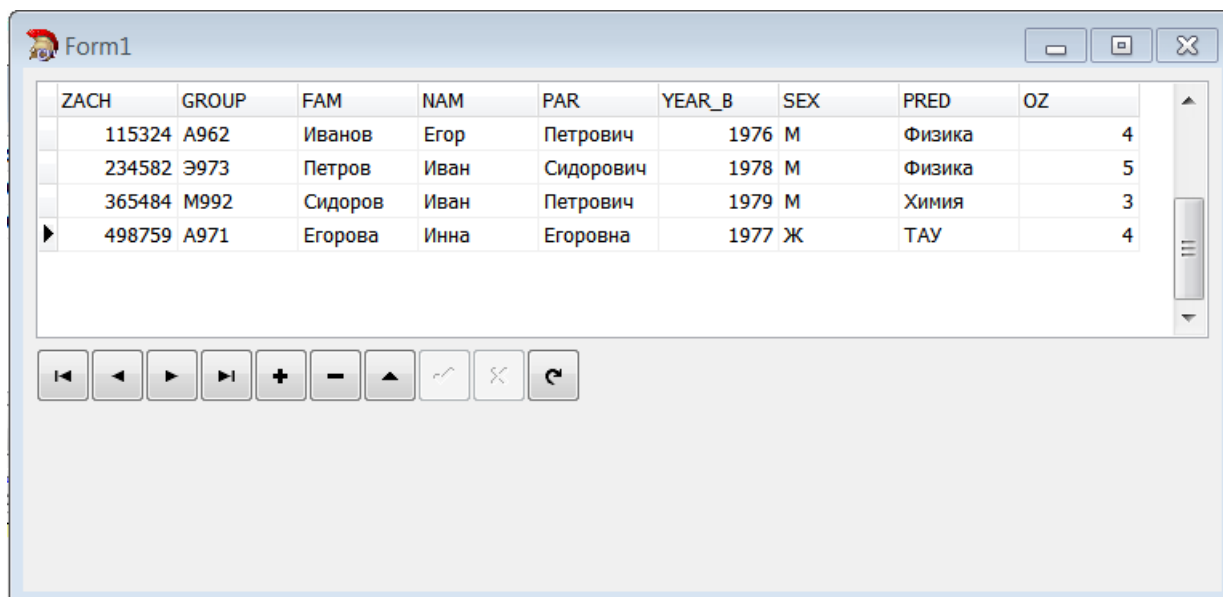


Рисунок 10.6 – Внешний вид работающего приложения

#### 4.4. Обработка информации однотабличной базы данных

Помимо хранения, редактирования, обновления и анализа данных в базе данных (БД), на практике часто возникает задача осуществления разного рода расчётов и вычислений с использованием хранящейся в БД информации. Например, для базы данных, хранящей информацию о студентах учебного заведения, может возникнуть задача расчёта средней оценки студента по конкретному предмету или расчёта успеваемости группы. Поскольку требуемые расчёты могут быть довольно сложные, необходимо использование программных пакетов разработки приложений, объединяющих в себе как удобные средства доступа к БД различных форматов, так и гибкий язык программирования.

Визуальная навигация и управление данными в Borland C++ Builder производится с помощью компонент просмотра (**DBGrid**) и управления (**DBNavigator**), которые позволяют перемещаться по записям, удалять, добавлять или модифицировать их и т.д.

Программная навигация по записям таблицы возможна использованием соответствующих методов. Компонента **ADOTable** основана на базовом классе **TADOTable**, в который инкапсулированы следующие методы навигации:

- **Next()** - перемещение указателя на следующую запись;
- **Prior()** - перемещение указателя на предыдущую запись;
- **First()** - перемещение указателя на первую запись таблицы;
- **Last()** - перемещение указателя на последнюю запись таблицы.



Например, если создан объект *ADOTable*, свойством *Name* которого является значение *ADOTable1*, то оператор *ADOTable1->First()* переместит указатель на первую запись.

Перечисленные методы позволяют последовательно перемещать указатель записи по таблице. Для определения последней записи таблицы используется ее свойство *Eof*, которое принимает значение *true*, когда указатель установлен на последней записи.

Фрагмент программы показывает, как организовать последовательный просмотр всех записей таблицы.

```
ADOTable1->First();           //перемещение на начало таблицы
while (ADOTable1->Eof!=true)  //проверка на конец файла
{
    ADOTable1->Next();         //перемещение на следующую запись
}
```

Для программного управления записями таблицы в класс *TADOTable* инкапсулированы следующие методы управления:

- *Delete()* - удаление текущей записи с позиционированием указателя на следующую запись;
- *Insert()* - добавление пустой записи в таблицу;
- *Edit()* – перевод таблицы в режим редактирования;
- *Post()* – сохранение изменений в таблице.

Например, фрагмент программы позволяет удалить последнюю запись таблицы.

```
ADOTable1->Last();
ADOTable1->Delete();
```

Перечисленные методы позволяют оперировать с целыми записями таблицы.

При обработке данных таблицы часто требуется программно получить значение определенного поля записи или записать в него новое значение. Например, в БД студентов учебного заведения необходимо поменять оценку в одной записи, не изменяя значения остальных полей. Подобные операции с полями таблицы требуют создания объектов, указывающих на соответствующие поля.

Для создания подобных объектов нужно выполнить следующую последовательность действий:

- нажать правой клавишей мыши на соответствующем объекте *ADOTable*;
- в выведенном меню выбрать опцию *Fields Editor*, в результате чего на экране появится диалог редактирования полей (рис. 10.7);
- нажать правой клавишей мыши в области диалога редактирования полей;
- в выведенном меню выбрать опцию *Add Fields*, в результате чего на экране появится соответствующий диалог;

- добавить необходимые поля выбором кнопки **Ok**. В результате диалог редактирования полей должен содержать список созданных объектов, свойства которых отражены в окне **Object Inspector**. Каждый из созданных объектов, указывающих на соответствующие поля, имеет свойство **Name**. Значение этого свойства является имя объекта (указатель). Имя указателя по умолчанию формируется из имени объекта **ADOTable** и имени поля.

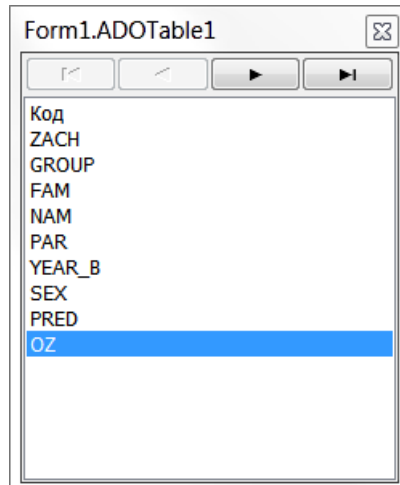


Рисунок 10.7 – Окно **Fields Editor**

Например, для таблицы с именем *stud.dbf*, содержащей фамилии студентов и их оценки с именами полей **FAM** и **OZ**, для полей **FAM** и **OZ** будут созданы объекты с именами **ADOTable1FAM** и **ADOTable1OZ**.

Созданные объекты позволяют оперировать с таблицей на уровне полей. Значение поля содержит свойство **Value**. Следовательно, обозначения **ADOTable1FAM->Value**, **ADOTable1OZ->Value** позволяют получать или модифицировать значения полей **FAM** и **OZ** текущей записи.

Фрагмент программы показывает, как вычислить среднюю оценку всех студентов. Для отображения на экране вычисленных данных можно, например, использовать объекты **Label** (текстовая метка). Свойство **Caption** объекта **Label** отвечает за надпись на форме приложения. Для функции **sprintf()** необходимо подключить библиотеку **stdio.h**.

```
float soz=0;           //переменная для средней оценки
int n=0;               //счётчик записей
ADOTable1->First();
while (ADOTable1->Eof!=true)
{
    soz=soz+ADOTable1OZ->Value; //накопление суммы
    n++;                       //инкремент счётчика записей
    ADOTable1->Next();          //переход на след. запись
}
if (n!=0)                //проверка отсутствия записей
soz = soz/n;              //расчёт среднего
char buf[50];
sprintf(buf, "Средняя оценка: %.2f", soz) ;
```

```
Label1->Caption = buf;
```

Кроме ввода данных в таблицу, как правило, в программу требуется вводить другие исходные данные. Для ввода исходных данных в программу используются объекты *Edit*. Объект *Edit* имеет свойство *Text*, которое является данным типа *AnsiString* и содержит значение, введенное в поле ввода на форме приложения.

Например, если имя объекта *Edit1*, то фрагмент программы позволяет рассчитать оценку не всех студентов, а конкретного студента, фамилия которого введена в поле ввода объекта *Edit1*.

```
float soz = 0;
int n=0;
ADOTable1->First();
while(ADOTable1->Eof!=true)
{
// проверка совпадения поля FAM и текста в Edit1
if(ADOTable1FAM->Value == Edit1->Text)
{
    soz=soz+ADOTable1OZ->Value;
    n++;
}
    ADOTable1->Next();
}
if(n!=0)
    soz=soz/n;
char buf[50];
sprintf(buf, "Средняя оценка: %.2f", soz);
Label1->Caption = buf;
```

Тип *AnsiString* позволяет непосредственно сравнивать строки с использованием логической операции «==». Ввод числовых значений осуществляется аналогичным образом с дальнейшим последовательным преобразованием значения типа *AnsiString* к типу символьной строки, а затем к числовому.

Например, фрагмент программы позволяет преобразовать значение переменной *Edit2->Text* типа *AnsiString* в символьную строку с использованием метода *c\_str()*.

```
float x=atof(Edit2->ADOText.c_str());
```

Доступ к методу осуществляется через «.» поскольку данное *Edit2->ADOText* является переменной, а не указателем. Ряд методов в *Borland C++ Builder* специально предусмотрен для преобразования типов:

- *FloatToStr()* - преобразование вещественного числа в строку;
- *IntToStr()* - преобразование целого числа в строку;
- *StrToFloat()* - преобразование строки в вещественное число;

– *StrToInt()* - преобразование строки в целое число.

Приведём вид формы приложения (рис. 10.8) примера работы с таблицей *stud*, расчёта средней оценки студента, фамилия которого вводится в поле ввода.

The screenshot shows a Windows form titled 'Form1'. It contains a table with the following data:

ZACH	GROUP	FAM	NAM	PAR	YEAR_B	SEX	PRED	OZ
115324	A962	Иванов	Егор	Петрович	1976	М	Физика	4
234582	Э973	Петров	Иван	Сидорович	1978	М	Физика	5
365484	M992	Сидоров	Иван	Петрович	1979	М	Химия	3
498759	A971	Егорова	Инна	Егоровна	1977	Ж	ТАУ	4

Below the table is a toolbar with various navigation and editing icons. To the right of the toolbar is a text input field labeled 'Введите имя студента' (Enter student name) with a text box containing 'Edit1'. Below this is a label 'Label1'. At the bottom left is a button labeled 'Расчет средней оценки' (Calculate average grade). The data source is indicated as 'DataSource1 ADOTable1'.

Рисунок 10.8 – Внешний вид формы приложения

На рис. 10.9 показан вид работающего Windows-приложения после отладки и компиляции проекта, запуска приложения, ввода в поле ввода фамилии «Петров» и нажатия кнопки «Расчёт средней оценки».

The screenshot shows the same Windows form 'Form1' after execution. The table data remains the same. The text input field 'Введите имя студента' now contains the text 'Сидоров'. Below the input field, the text 'Средняя оценка: 3.00' is displayed. The button 'Расчет средней оценки' is now highlighted in blue, indicating it was just clicked. The data source is still 'DataSource1 ADOTable1'.

Рисунок 10.9 – Внешний вид работающего приложения

## 5 Варианты заданий для самостоятельной работы

1. **Картотека Интерпола.** Данные по зарегистрированному преступнику: фамилия, имя, кличка, рост, цвет волос и глаз, особые приметы, гражданство место и дата рождения, последнее место жительства, знание языков, преступная профессия, последнее дело. Выборка по фамилии, преступной профессии.

2. **Бюро знакомств.** База потенциальных женихов и невест: пол, регистрационный номер, дата регистрации, сведения о себе, требования к партнеру. Выбор подмножества подходящих кандидатур, подготовка встреч.

3. **Биржа труда.** База безработных: анкетные данные, профессия, образование, место и должность последней работы, причина увольнения, семейное положение, жилищные условия, контактные координаты, требования к будущей работе. База вакансий: фирма, должность, условия труда и оплаты, жилищные условия, требования к специалисту. Поиск и вариантов; формирование объявлений для печати.

4. **Касса аэрофлота.** Расписание: номер рейса, маршрут, пункты промежуточной посадки, время отправления, дни полета. Количество свободных мест на каждом рейсе. Выбор ближайшего рейса до заданного пункта (при наличии свободных мест), оформление заданного числа билетов по согласованию с пассажиром (с уменьшением числа свободных мест), оформление посадочной ведомости.

5. **Магазин с одним продавцом.** Компьютер вместо кассового аппарата. База наличия товаров: наименование, единица измерения, цена единицы, количество, дата последнего завоза. Оформление покупки: выписка чека, корректировка базы. Инвентаризация остатков товара с вычислением суммарной стоимости.

6. **Отдел кадров.** База данных о сотрудниках фирмы: паспортные данные, образование, специальность, подразделение, должность, оклад, даты поступления в фирму. Выбор и печать приказов для увольнения лиц пенсионного и предпенсионного возраста.

7. **Склад.** База товаров, хранящихся на складе: наименование, единица измерения, цена единицы, количество, дата последнего завоза. Регистрация поступления товара (формирование приходной накладной) и отгрузки (расходная накладная). Вывод инвентарной ведомости.

8. **Касса автовокзала.** Расписание автобусов: номер рейса, конечный и промежуточный пункты, время отправления. Количество свободных мест на каждом рейсе. Выбор ближайшего рейса до заданного пункта (при наличии свободных мест), оформление билетов, оформление посадочной ведомости.

9. **Справочник лекаря.** База болезней: название, симптомы, процедуры, перечень рекомендуемых лекарств с указанием требуемого количества. Формирование рецепта после осмотра больного.

10. **Зачисление абитуриентов.** База абитуриентов: анкетные данные, совокупность оценок на вступительных экзаменах, готовность учиться на договорной основе. Выбор для зачисления заданного количества абитуриентов; формирование для собеседования списка тех, кто набрал предельный проходной балл, но не может платить за образование.

11. **Обмен жилья.** База предложений по обмену: район, площадь, планировка и т. д.; требования к вариантам обмена. Выбор подходящих вариантов, печать объявлений.

12. **Сбербанк.** Сведения о вкладчиках банка: номер лицевого счета, категория вклада, паспортные данные, текущая сумма вклада, дата последней операции. Операции приема и выдачи любой суммы, автоматическое начисление процентов.

13. **Ломбард.** База хранимых товаров и недвижимости: анкетные данные клиента, наименование товара, оценочная стоимость; сумма, выданная под залог, дата сдачи, срок хранения. Операции приема товара, возврата, продажи по истечении срока хранения.

14. **Справочник работника ГИБДД.** Марка, цвет, заводской и бортовой номера, дата выпуска, особенности конструкции и окраски, дата последнего техосмотра транспортного средства (автомобиля, мотоцикла, прицепа и т.д.), паспортные данные владельца. Выбор транспортных средств по произвольному шаблону. Формирование приглашений на техосмотр в соответствии со сроком.

15. **Справочник фаната.** База спортсменов: анкетные и антропологические данные, гражданство, происхождение, вид спорта, клуб или команда, данные о личном рекорде или победах и так далее. Поиск рекордсмена в заданном виде спорта.

16. **Автосалон.** База автомобилей: марка, год выпуска, технические характеристики, особенности исполнения, техническое состояние, запрашиваемая цена. Автоматизация подбора вариантов для покупателя, формирование заявки для поставщиков и перегонщиков.

17. **База данных** «Табель рабочего времени»: ФИО, количество отработанных часов по дням, итог за месяц. Печать ведомости.

18. **Видеосалон** – сведения о дисках, ФИО взявшего, срок выдачи. Подбор имеющихся дисков, печать просроченных.

19. **Информационный киоск** – список товара, его характеристики, наличие на складе, печать остатков, прайс-листа, характеристик.

20. **Картотека библиотеки** – сведения о книгах, кто взял и на какой срок, печать просроченных.

21. **Журнал регистрации курсовых(контрольных) работ** – ФИО студента, факультет, группа, тема курсовой, предмет, дата и номер регистрации, печать списка, печать этикетки регистрации.

22. **База данных «Секретарь»** - список дела, дата и время, ФИО сотрудника для вызова, печать списка дел, печать вызова.

23. **БД медицинские учреждения города** – наименование, адрес нахождения, количество врачей всего, наличие специалистов (лор, хирург, стоматолог, ортопед, невропатолог и т.д.). Печать списка, выбор по адресу, печать наличие врачей в выбранном учреждении.

24. **БД наличие лекарств в аптеках города.**

25. **БД справочная по товарам и услугам** – наименование товара или услуги, фирма, адрес и телефоны фирмы, выбор товара или услуги, печать адресов и телефонов фирм, предлагающих данную услуги или товар.

26. **БД «Учет компьютеров на фирме»** - сведения о компьютерах (материнская плата, видеокарта, монитор, HDD, CDD, FDD, принтер, сканер, модем, сетевая карта, операционная система, год выпуска, срок гарантии), место установки. Печать реестра компьютеров, характеристики выбранного, печать списка у которых закончилась гарантия.

27. **Карточка трЗ** – сведения о файлах (место расположения, артист, произведение, альбом, год выпуска, жанр). Печать реестра файлов, печать произведений по жанру, формирование и печать плейлиста.

28. **График дежурства преподавателей в общежитии** – ФИО преподавателя, дата дежурства. Печать списка дежурств, печать дежурного на заданную дату.

29. **Система тестирования** – база содержит вопросы и ответы типа да или нет. Провести тестирование, поставить оценку, печать вопросов, печать оценки.

30. **Журнал регистрации вызовов таксопарка** – информация о вызовах (район вызова, телефон клиента, куда поедет, время вызова, номер автомотора). Печать свободных вызовов, печать реестра вызовов.

## ЛИТЕРАТУРА

1. С/С++. Программирование на языке высокого уровня / Т. А. Павловская. – СПб.: Питер, 2002. – 464 с.
2. С/С++. Структурное программирование: Практикум / Т. А. Павловская, Ю.А.Щупак. – СПб.: Питер, 2003. – 240 с.:ил.
3. Архангельский А.Я. Программирование в С++ Builder 6. – М.Ж ЗАО «Издательство БИНОМ», 2005. – 1152 с.
4. Грегори К. Использование Visual С++ 6. Специальное издание.: Пер. с англ. – М.; СПб.; К.: Издательский дом “Вильямс”, 1999. – 864 с.: ил.– Парал. тит. англ., уч. пос.
5. Подбельский В.В. Язык С++: Учебное пособие. – 5-е изд. -М.: Финансы и статистика, 2003. -560 с.
6. Подбельский В.В., Фомин С.С. Программирование на языке СИ: Учебное пособие. – М: Финансы и статистика, 1998. – 600 с.:ил.
7. Пол И. Объектно-ориентированное программирование с использованием Си++: Пер. с англ. – Киев: НИПФ ДиаСофт Лтд, 1995. – 480 с.
8. Прата С. Язык программирования С++. Лекции и упражнения.
9. Сван Т. Освоение Borland С++ 4.5: Практический курс. -Киев: Диалектика, 1996. -544 с.
10. Скляров В.А. Язык С++ и объектно-ориентированное программирование. – Мн.: Выш. шк., 1997. – 478 с.
11. Страуструп Б. Язык программирования Си++: Пер. с англ. – М.: Радио и связь, 1991. – 352 с.
12. Черносвитов А. Visual С++ и MFC. Курс MCSD для профессионалов – СПб: Издательство «Питер», 2000. – 544 с.: ил.
13. Элджер Дж. С++: библиотека программиста – СПб: ЗАО Издательство «Питер», 1999. – 320 с.: ил.