

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет информатики  
и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

# **ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ (ЯЗЫК C/C++). ЛАБОРАТОРНЫЙ ПРАКТИКУМ**

В двух частях

Часть 1

*Рекомендовано УМО по образованию в области информатики  
и радиоэлектроники в качестве учебно-методического пособия  
для всех специальностей I ступени высшего образования,  
закрепленных за УМО*

Минск БГУИР 2017

УДК 004.42(076.5)  
ББК 32.972.1я73  
О-75

**А в т о р ы:**

С. А. Беспалов, И. Е. Зайцева, Т. М. Кривоносова, Т. А. Рак, В. Л. Смирнов,  
О. О. Шатилова, В. П. Шестакович

**Р е ц е н з е н т ы:**

кафедра экономической информатики учреждения образования  
«Белорусский государственный экономический университет»  
(протокол №3 от 26.10.2015);

доцент кафедры прикладной информатики учреждения образования  
«Белорусский государственный аграрный технический университет»,  
кандидат технических наук, доцент А. И. Шакирин

**Основы** алгоритмизации и программирования (язык C/C++). Лабо-  
О-75 торный практикум. В 2 ч. Ч. 1 : учеб.-метод. пособие / С. А. Беспалов [и др.]. – Минск : БГУИР, 2017. – 71 с. : ил.  
ISBN 978-985-543-242-6 (ч. 1).

Состоит из 9 тем, содержащих теоретические сведения, примеры выполнения задач и индивидуальные задания.

Изложены краткие теоретические сведения по основам алгоритмического языка C/C++ и программированию в современных средах программирования, приведены примеры реализации алгоритмов в консольном (*Visual, Builder*) и оконном (*Builder*) приложениях.

УДК 004.42(076.5)  
ББК 32.972.1я73

ISBN 978-985-543-242-6 (ч. 1)  
ISBN 978-985-543-241-9

© УО «Белорусский государственный  
университет информатики  
и радиоэлектроники», 2017

## СОДЕРЖАНИЕ

<b>ТЕМА №1. ЛИНЕЙНЫЙ ВЫЧИСЛИТЕЛЬНЫЙ ПРОЦЕСС</b>	<b>4</b>
1.1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	4
1.2. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ В СРЕДЕ BUILDER	8
1.3. СОЗДАНИЕ КОНСОЛЬНОГО ПРИЛОЖЕНИЯ	11
1.4. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ	13
1.5. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ	15
<b>ТЕМА №2. РЕАЛИЗАЦИЯ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ</b>	<b>19</b>
2.1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	19
2.2. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ В СРЕДЕ BUILDER	20
2.3. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ	21
2.4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ	23
<b>ТЕМА №3. РЕАЛИЗАЦИЯ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ</b>	<b>27</b>
3.1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	27
3.2. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ	28
3.3. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ	29
<b>ТЕМА №4. ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ</b>	<b>32</b>
4.1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	32
4.2. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ В СРЕДЕ BUILDER	32
4.3. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ	33
4.4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ	34
<b>ТЕМА №5. ОБРАБОТКА ДВУХМЕРНЫХ ДИНАМИЧЕСКИХ МАССИВОВ</b>	<b>37</b>
5.1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	37
5.2. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ	39
5.3. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ	40
<b>ТЕМА №6. ИСПОЛЬЗОВАНИЕ СТРОК</b>	<b>43</b>
6.1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	43
6.2. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ В СРЕДЕ BUILDER	44
6.3. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ	45
6.4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ	46
<b>ТЕМА №7. ОБРАБОТКА СТРУКТУР С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВ</b>	<b>50</b>
7.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	50
7.2. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ В СРЕДЕ BUILDER	52
7.3. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ	52
7.4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ	55
<b>ТЕМА №8. ФУНКЦИИ ПОЛЬЗОВАТЕЛЯ</b>	<b>58</b>
8.1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	58
8.2. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ	60
8.3. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ	60
<b>ТЕМА №9. ОТОБРАЖЕНИЕ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ</b>	<b>63</b>
9.1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	63
9.2. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ	65
9.3. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ	66
<b>ПРИЛОЖЕНИЕ 1. ОСНОВНЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ</b>	<b>67</b>
<b>ПРИЛОЖЕНИЕ 2. ОПИСАНИЕ ОБЩИХ СТРУКТУР ФАЙЛОВ ПРОЕКТА</b>	<b>68</b>
<b>ЛИТЕРАТУРА</b>	<b>70</b>

## Тема №1. Линейный вычислительный процесс

**Цель работы:** изучить правила составления текстов программ, научиться реализовывать линейные алгоритмы; написать и отладить программу, реализующую линейный алгоритм.

### 1.1. Общие теоретические сведения

Алфавит языка C/C++ состоит из прописных и строчных букв латинского алфавита, арабских цифр и специальных символов, смысл и правила использования которых будут рассматриваться далее.

В языке C применяются данные двух категорий: простые (скалярные) и сложные (составные).

К базовым типам данных относятся целый (*int*), вещественный (*float*, *double*) и символьный (*char*) типы. В свою очередь данные целого типа могут быть короткими (*short*) или длинными (*long*), со знаком (*signed*) или беззнаковыми (*unsigned*). Атрибут *long* может использоваться и с типом *double* – длинное вещественное.

К сложным типам данных относятся массивы, структуры (*struct*), объединения (*union*) и перечисления (*enum*).

В языке C++ используются данные типа *bool* – логические, принимающие значения *true* (1) – истина – и *false* (0) – ложь, а также *AnsiString* (*String*) – строковый тип данных (см. подразд. 7.2).

#### Запись самоопределенных констант

Тип данных	Общий формат записи	Примеры
Десятичные целые	$\pm n$	22   -15   176   -1925
Вещественные с фиксированной десятичной точкой	$\pm n.m$	1.0   -3.125   -0.001 .56   3.
Вещественные с плавающей точкой	$\pm n.mE\pm p$ смысл записи $\pm n,m \cdot 10^{\pm p}$	1.01E-10   0.12537e+4 -5.25e3
Символьные	' * '	'A'   'x'   '0'   '<'
Строковые	" ***** "	"Minsk"   "Press any key"
* – любой символ, набранный на клавиатуре.		

#### Декларация объектов

Все объекты (переменные, массивы и т. д.), с которыми работает программа, необходимо декларировать (объявлять). В декларации объектам присваиваются идентификаторы, т. е. имена, которые могут включать латинские буквы, символ нижнего подчеркивания « » и цифры, причем первым символом имени не может быть цифра.

**Внимание!** В языке C/C++ строчные и прописные буквы имеют различные коды, т. е. *PI*, *Pi* и *pi* – различные идентификаторы.

При декларации объектам можно задавать начальные значения (инициализировать), например:

```
int    k = 10, m = 3, n;  
double c = -1.3, w = -10.23, s;
```

Принято использовать в именах переменных строчные буквы, а в именованных константах – прописные, например:

```
const double PI = 3.1415926;  
double pi = 3.1415926;
```

Комментарий – любая последовательность символов, начинающаяся парой символов /\* и заканчивающаяся парой символов \*/ или начинающаяся // и продолжающаяся до конца текущей строки.

### ***Директивы препроцессора***

Перед компиляцией программы с помощью директив препроцессора выполняется предварительная обработка текста программы.

Директивы начинаются с символа #; за которым следует наименование операции препроцессора. Чаще всего используются директивы ***include*** и ***define***.

Директива ***#include*** используется для подключения к программе заголовочных файлов с декларацией стандартных библиотечных функций, например:

```
#include <stdio.h> – стандартные функции ввода-вывода;  
#include <conio.h> – функции работы с консолью;  
#include <math.h> – математические функции.
```

Директива ***#define*** (определить) создает макроконстанту и ее действие распространяется на весь файл, например:

```
#define    PI    3.1415927
```

В ходе препроцессорной обработки идентификатор *PI* везде заменяется указанным значением 3.1415927.

***Операции языка C/C++*** (арифметические: +, -, \*, /, %) и наиболее часто используемые ***стандартные математические функции*** описаны в прил. 1.

При использовании деления надо помнить, что при ***делении целого числа на целое*** остаток отбрасывается, таким образом, ***7/4 будет равно 1***. Если же надо получить вещественное число и не отбрасывать остаток, делимое или делитель надо преобразовать к вещественному типу.

```
Например, для переменных    int i = 7, n;    и    double x;  
x = i / 4;                    результат    x = 1, т. к. int делится на int;  
x = i / 4.;                  результат    x = 1.75, т. к. int делится на double;  
x = (double) i / 4;          результат    x = 1.75, т. к. double делится на int;  
n = 7. / 4.;                 n = 1, т. к. double результат присваивается переменной  
                             типа int.
```

***Операция присваивания*** имеет полную и сокращенную формы записи.

***Полная форма:*** ***Имя\_переменной = Выражение;***

Операция выполняется справа налево, т. е. сначала вычисляется ***Выражение***, а затем его результат присваивается указанной переменной, например:

- 1)  $y = (x + 2) / (3 * x) - 5;$  – присваивается значение выражения;
- 2)  $x = y = z = 0;$  – последовательно  $z$ ,  $y$  и  $x$  присваиваются значения 0;
- 3)  $z = (x = y) * 5;$  – сначала переменной  $x$  присваивается значение  $y$ , далее вычисляется выражение  $x * 5$ , и результат присваивается переменной  $z$ .

**Сокращенная форма:** *Имя\_переменной Операция= Выражение;*

Здесь *Операция* – одна из арифметических операций (+, –, \*, /, %), например:

$s += 7;$  ( $s = s + 7;$ )      или       $y *= x + 3;$  ( $y = y * (x + 3);$ );

Сокращенная форма применяется, когда переменная используется в обеих частях ее полной формы.

В языке C/C++ существуют унарные операции **инкремента** (++) и **декремента** (--), т. е. увеличения или уменьшения значения переменной на 1. Операции могут быть **префиксные** (++*i* и --*i*) и **постфиксные** (*i*++ и *i*--). При использовании в выражении операции в префиксной форме сначала выполняется сама операция (изменяется значение *i*) и только потом вычисляется выражение. В постфиксной форме операция применяется после вычисления выражения, например, для значений  $b = 7$  и  $n = 1$  будут получены следующие результаты:

- 1)  $c = b * ++n;$  (порядок выполнения:  $n = n + 1$ ,  $c = b * n$ , т. е.  $c = 14$ );
- 2)  $c = b * n++;$  (в этом случае:  $c = b * n$ ,  $n = n + 1$ , т. е.  $c = 7$ ).

### **Интегрированная среда разработчика C++ Builder**

Среда *Builder* визуально реализуется в виде нескольких окон, одновременно раскрытых на экране монитора. Количество, расположение, размер и вид окон может меняться пользователем в зависимости от поставленной задачи. Меню среды *Builder* может иметь вид, представленный на рис. 1.1.

**Главное окно** предназначено для управления процессом создания программы. Основное меню содержит необходимые средства для управления проектом. Пиктограммы облегчают доступ к командам основного меню. Через меню компонент осуществляется выбор стандартных сервисных программ среды, которые описывают некоторый визуальный элемент (компоненту), помещенный в окно формы. Каждая компонента имеет набор свойств, задаваемых пользователем, например заголовок окна, надпись на кнопке и т. п.

**Окно инспектора объектов** (вызывается нажатием клавиши **F11**) предназначено для изменения свойств выбранных компонент и состоит из двух страниц: *Properties* (свойства) – для изменения свойств компоненты, *Events* (события) – для определения реакции компоненты на то или иное событие, например нажатие определенной клавиши или щелчок кнопкой мыши.

**Окно формы** представляет собой проект *Windows*-окна программы, в которое помещаются необходимые компоненты для решения поставленной задачи, причем при выполнении программы помещенные компоненты будут иметь тот же вид, что и на этапе проектирования.

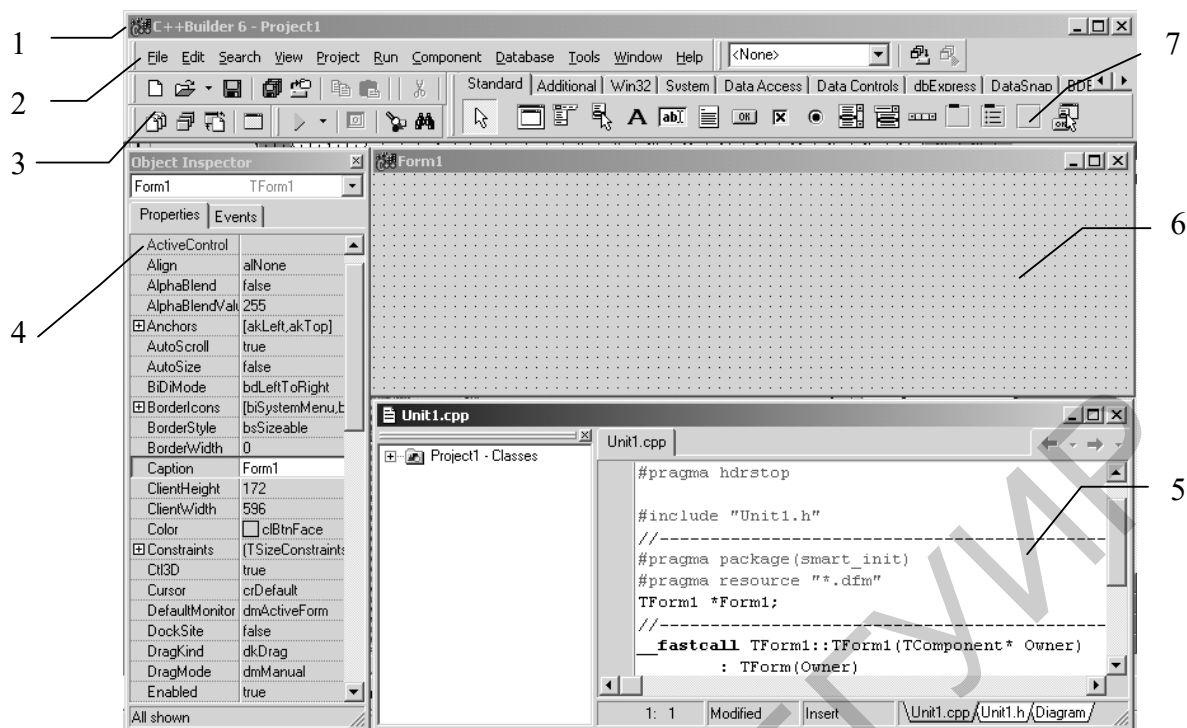


Рис. 1.1:

- 1 – главное окно; 2 – основное меню; 3 – пиктограммы основного меню;  
4 – окно инспектора объектов; 5 – окно текста программы; 6 – окно формы;  
7 – меню компонент

**Окно текста** (имя *Unit1.cpp*) предназначено для написания, редактирования и просмотра текста программы. При первоначальной загрузке в окне текста программы находится текст, содержащий минимальный набор операторов для нормального функционирования пустой формы в качестве *Windows*-окна.

Программа в среде *Builder* состоит из функций, которые необходимо выполнить, если возникает определенное событие, связанное с формой, например, щелчок кнопкой мыши – событие *OnClick*, создание формы – *OnCreate*.

Для каждого события формы с помощью страницы *Events* инспектора объектов или двойным щелчком кнопкой мыши по выбранной компоненте в текст программы вставляется функция-обработчик, в которую между символами { } записываются нужные действия.

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

### Структура программы C++ Builder

Программа в *Builder* состоит из множества функций, объединяемых в один проект с помощью файла проекта *Project1.bpr*, который создается автоматически, обрабатывается средой *Builder* и не предназначен для редактирования.

При создании консольного или оконного приложений системой программирования автоматически формируется текстовый файл с именем *Unit1.cpp*.

В оконном приложении объявления классов, переменных (объектов) и функций-обработчиков (методов) находятся в заголовочном файле, имеющем





то же имя, что и текстовый файл, только с расширением *.h*. Описание окна формы находится в файле с расширением *.dfm*. Файл проекта может быть только один, файлов с другими расширениями может быть несколько.

**Внимание!** Для того чтобы перенести проект на другой компьютер, необходимо переписать все файлы с расширениями: *\*.bpr, \*.h, \*.cpp, \*.dfm*.

Общий вид структур следующих файлов: текст программы (*Unit1.cpp*), проект (*Project1.cpp*), заголовочный (*Unit1.h*) – приведен в прил. 2.

## 1.2. Создание оконного приложения в среде Builder


### Настройка формы

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

Для изменения заголовка после вызова окна инспектора объектов (*F11*) выбирается свойство *Caption*, и в выделенном окошке вместо стандартного текста **Form1** набирается нужный текст, например «Л.р. 1. Гр.610101 Иванова А.».

**Внимание!** Свойства *Name* (имя) и *Caption* (заголовок) у компонент совпадают, но имя менять не рекомендуется, т. к. оно входит в текст программы.

### Компонента Edit

Если необходимо ввести из формы в программу или вывести на форму информацию, которая вмещается в одну строку, используют окно однострочного редактора текста, представляемого компонентой **Edit**, для чего в меню компонент *Standard* выбираем пиктограмму  и щелчком кнопки мыши устанавливаем ее в нужном месте формы. Мышью регулируются размер окошка и его положение на форме.

В заголовочный файл *Unit1.h* автоматически вставляется переменная *Edit\** (1, 2, ...) класса *TEdit*. В поле *Text* (*Edit1->Text*) такой переменной будет содержаться строка символов (тип *AnsiString*) и отображаться в соответствующем окне *Edit\**.

### Основные функции преобразования строк:

**StrToFloat**(*St*) – преобразует строку *St* в вещественное число;

**StrToInt**(*St*) – преобразует строку *St* в целое число;

**IntToStr** (*W*) – преобразует целое число *W* в строку символов;

**FloatToStr** (*W*) – преобразует вещественное число *W* в строку символов;

**FormatFloat** (*формат, W*) – преобразует вещественное число *W* в строку;

**FloatToStrF** (*W, формат, n1, n2*) – преобразует вещественное число *W* в строку символов под управлением *формата*.



Примеры используемых форматов:

***ffFixed*** – фиксированное положение разделителя целой и дробной частей, *n1* – общее количество цифр числа, *n2* – количество цифр в дробной части, причем число округляется с учетом первой отбрасываемой цифры;

***ffExponent*** – *n1* задает общее количество цифр мантиссы, *n2* – количество цифр порядка *XX* (число округляется);

***ffGeneral*** – универсальный формат, использующий наиболее удобную для чтения форму представления вещественного числа; соответствует формату ***ffFixed***, если количество цифр в целой части  $\leq n1$ , а само число больше 0.00001, в противном случае соответствует формату ***ffExponent***.

Если значения вводимых из *Edit1* и *Edit2* переменны *x* и *y* имеют целый и действительный типы соответственно, то следует записать:

*x* = StrToInt(*Edit1*→Text);


*y* = StrToFloat(*Edit2*→Text);

**Внимание!** При записи числовых значений в окошках *Edit\** не должно быть пробелов, а разделителем целой и дробной частей обычно является «**запятая**»!

В инспекторе объектов с помощью свойства **Font** устанавливается стиль (шрифт, вид начертания, размер и цвет) отображаемого в строке *Edit\** текста.

**Внимание!** Не все типы шрифтов поддерживают кириллицу.

### **Компонента Label**

Данная компонента используется для вывода надписей, для чего в меню компонент *Standard* выбираем пиктограмму  и в нужном месте формы устанавливаем надпись *Label1*, регулируем размер, место положения, изменяем свойство *Caption* инспектора объектов, в котором вводим нужный текст, например строка “Значение X = ”, а также выбираем стиль (свойство *Font*).

При установке таких компонент в текст *Unit1.h* вставляются переменные типа *TLabel*, в которых хранятся пояснительные строки. Эти строки можно изменять в процессе работы программы, например:


*Label1*→*Caption* = “”;

– «очистка» строки;

*Label1*→*Caption* = “Не выполняется!”;

– вывод строки.

### **Компонента Memo**

Для вывода результатов обычно используется компонента *Memo* – окно многострочного редактора текста. Выбираем пиктограмму  и помещаем ее на форму, регулируем размер, местоположение, а с помощью свойства *ScrollBars* (*SSBoth*) можем установить полосы прокрутки.

При установке этой компоненты на форму в файле *Unit1.h* появляется переменная *Memo1* типа *TMemo*. Информация, выводимая построчно в окне *Memo1*, находится в массиве строк *Memo1*→*Lines*, каждая из которых имеет тип *String*.

Для очистки окна используется метод *Memo1*→*Clear*( ).

Для добавления новой строки используется метод *Memo1*→*Lines*→*Add*( ).

Если нужно вывести числовое значение, то его надо преобразовать к типу *AnsiString* и добавить в массив *Memo1->Lines*, например вывести

```
int u = 100;
```

```
double w = -256.38666;
```

в результате записей

```
Memo1->Lines->Add (" Значение u = "+IntToStr(u));
```

```
Memo1->Lines->Add (" Значение w = "+FloatToStrF(w,ffFixed,8,2));
```

появятся строки

```
Значение u = 100
```

```
Значение w = -256.39
```

При этом под все число отводится восемь позиций, из которых две позиции занимает его дробная часть.

Если выводимая информация превышает размер окна *Memo1*, то для просмотра используются полосы прокрутки.

### **Обработка событий**

Напомним, что программа в среде *Builder* представляет собой набор функций, выполняющих обработку событий, связанных с формой, например, щелчок кнопкой мыши – событие *OnClick*, создание формы – событие *OnCreate*.

### **Функция-обработчик *FormCreate***


При запуске программы возникает событие «создание формы». Оформим функцию-обработчика этого события, которая обычно используется для инициализации начальных установок, таких как, например, занести начальные значения исходных данных в соответствующие окна *Edit\**, очистить окно *Memo*.

Для этого делаем двойной щелчок кнопкой мыши в любом **свободном** месте формы, после чего в листинг программы (*Unit1.cpp*) автоматически вносится заготовка для создания функции: ее заголовок ... ***FormCreate (...)*** и фигурные скобки.

Между символами { }, которые обозначают начало и конец функции соответственно, вставляем нужный текст программы (см. п. 1.4.1).


**Внимание! Не набирайте заголовки функций-обработчиков вручную.**

### **Функция-обработчик нажатия кнопки (*Button\*Click*)**

Выбираем в меню *Standard* пиктограмму  и помещаем ее на форму (например, *Button1*). С помощью инспектора объектов изменяем заголовок (*Caption*) на текст, например «Выполнить», регулируем положение и размер кнопки. Двойным щелчком кнопки мыши по этой компоненте в текст программы вставляем заготовку ее функции-обработчика ... ***Button1Click (...)*** { }. Между фигурными скобками набираем соответствующий код.

### ***Запуск и работа с программой***

Перед запуском программы на обработку следует сохранить программу, для чего нужно выбрать в меню *File* пункт *Save All*.

Запустить программу можно, нажав *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 (рис. 1.2).

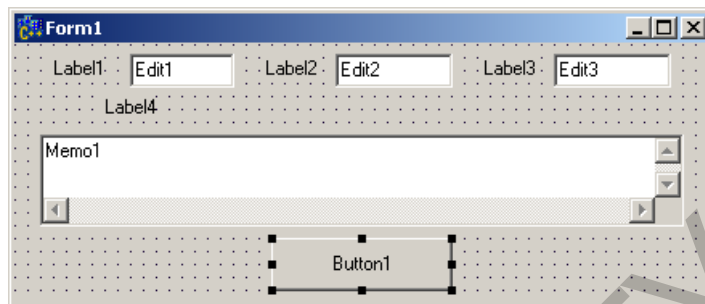



Рис. 1.2

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

### **1.3. Создание консольного приложения**

Программа, написанная на языке C/C++, состоит из одной или нескольких функций, одна из которых обязательно имеет идентификатор (имя) *main* – основная, главная. Ее назначение – управление всей работой программы (проекта).

#### ***Стандартные функции вывода информации***

Для вывода информации в консольном приложении чаще всего используются следующие функции:

***puts(S)*** – вывод строки символов *S* с переходом на начало новой строки и вывод данных с форматированием;

***printf*** (*Управляющая строка, Список вывода*).

*Управляющая строка* – заключенная в кавычки строка, содержащая спецификации преобразования объектов вывода, управляющие символы (признак «\») и любой набор символов, использующийся в качестве поясняющего текста (указывает компилятору вид выводимой информации).

*Список вывода* – печатаемые объекты (константы, переменные или выражения, вычисляемые перед выводом). Данные, указанные в списке, выводятся в соответствии со спецификациями управляющей строки.

Спецификации преобразования имеют вид

**% <Флаг> <Размер поля . Точность> Спецификация**

Выполняются следующие действия:

**флаг:** – (минус) выравнивание влево (по умолчанию выполняется выравнивание вправо); + (плюс) выводится знак положительного числа;

*размер поля* – задает ширину поля вывода (количество символов), при недостаточном значении выполняется автоматическое расширение;

*точность* – задает количество цифр в дробной части числа;

*спецификация* – формат преобразования выводимого объекта.

Приведем основные форматы печати:

%d	– десятичные целые ( <i>int</i> );	%c	– один символ ( <i>char</i> );
%s	– строка символов ( <i>string</i> );	%f	– данные типа <i>float</i> ;
%ld	– длинное целое;	%lf	– данные типа <i>double</i> ;
%x	– шестнадцатеричные данные;	%o	– восьмеричные данные.

При необходимости вывода управляющих символов (`% \` и т. п.) их нужно указать 2 раза, например:

```
printf("Только %d%% предприятий не работало. \n",5);
```

получим:

*Только 5% предприятий не работало.*

**Некоторые управляющие символы:** `\n` – переход на новую строку; `\t` – горизонтальная; `\r` – возврат в начало строки; `\a` – звуковой сигнал.

### **Стандартные функции ввода информации**

Функция ***gets (S)*** обеспечивает ввод строки символов *S* до нажатия клавиши *Enter*, т. е. позволяет ввести строку, содержащую пробелы.

Для форматированного ввода любой информации предназначена функция ***scanf (Управляющая строка, Список ввода)***;

В *Управляющей строке* указываются только спецификации преобразований, а в *Списке ввода* – **адреса** вводимых скалярных переменных, для чего перед именем переменной указывается операция `&`, обозначающая «взять адрес». Для ввода значений строковых (составных) переменных символ `&` не указывается. При использовании формата `%s` строка вводится до первого пробела.

*Пример* ввода данных *int a, double b* и *char str[20]*:

```
scanf ("%d %lf %s", &a, &b, str);
```

Вводить данные можно в одной строке через пробел или в разных строках.

### **Потоковый ввод-вывод информации**

Для ввода-вывода в языке C++ используются два класса, описанные в библиотеке *iostream.h*: *cout* (класс *вывода*) и *cin* (класс *ввода*).

Вывод информации на экран:

```
cout << "Hello world!" << endl;
```

*endl (end line)* – переход на новую строку.

Ввод информации (строки до появления первого пробела), например данных *int a, double b* и *char str[20]*:

```
cin >> a; cin >> b; cin >> str;           или           cin >> a >> b >> str;
```

## 1.4. Пример выполнения задания

Составить программу вычисления выражения для значений  $x, y$  и  $z$ :

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

### 1.4.1. Пример создания оконного приложения в *Builder*

В оконном режиме создать панель диалога программы в виде, представленном на рис. 1.2.

Для создания проекта необходимо выполнить следующие действия.

1. Запускаем C++ *Builder*. Создаем в разрешенной для пользователя папке (d:\work\ или c:\work\ ) папку со своим номером группы, открыв ее, сохраняем все файлы (стандартные имена файлов рекомендуем не изменять).

2. Оформляем окно формы, заменив заголовок *Form1* на нужный текст. Помещаем на форму необходимые компоненты *Label1, Label2, Label3, Label4* (вставляя в *Caption* соответствующие тексты), *Edit1, Edit2, Edit3, Memo1* с полосами прокрутки (см. п. 1.3), *Button1* (заменив в *Caption* текст). Используя свойство *Font*, выбираем стили выводимых текстов.

3. Оформляем листинг программы (*Unit1.cpp*). Двойным щелчком кнопки мыши по свободному месту формы создаем функцию *FormCreate* и заполняем ее (см. пример текста программы ниже). Переходим на форму (*F12*), нажимаем дважды кнопку *Button1* (ВЫПОЛНИТЬ) и заполняем созданную функцию *Button1Click* (см. пример).

4. Перед запуском программы на обработку **сохраняем все**.

5. Запускаем проект на выполнение, исправляем ошибки.

**Пример.** Текст программы может иметь следующий вид (курсивом выделен текст, **созданный компилятором, редактировать который не рекомендуется**):

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "math.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{
}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    Edit1->Text = "3,4";           Edit2->Text = "7,4e-2";
    Edit3->Text = "1,943e2";       Memo1->Clear();
    Memo1->Lines->Add("Лабораторная работа №1");
}
```

```
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    double x, y, z, a, b, c, rez;
    x = StrToFloat(Edit1->Text);
    y = StrToFloat(Edit2->Text);
    z = StrToFloat(Edit3->Text);
    a = pow(tan(x+y),2);
    b = exp(y-z);
    c = sqrt(cos(x*x)+sin(z*z));
    rez = a-b*c;
    Memo1->Lines->Add("При x = "+FloatToStrF(x,ffFixed,7,3) +
        " y = "+FloatToStrF(y,ffFixed,7,3) + " z = "+FloatToStrF(z,ffFixed,7,3));
    Memo1->Lines->Add("Результат = "+FloatToStr(rez));
}

```

**Внимание!** В строковых константах разделителем целой и дробной частей является *запятая* (Edit1->Text = "3,4") в отличие от числовых констант в тексте программы.

В результате должно получиться рабочее окно (рис. 1.3). Если нажать кнопку «ВЫПОЛНИТЬ», в окне *Memo1* появится соответствующий текст (результат). Далее в окошках *Edit\** можно изменять исходные значения и, нажимая кнопку «ВЫПОЛНИТЬ», получать новые результаты.

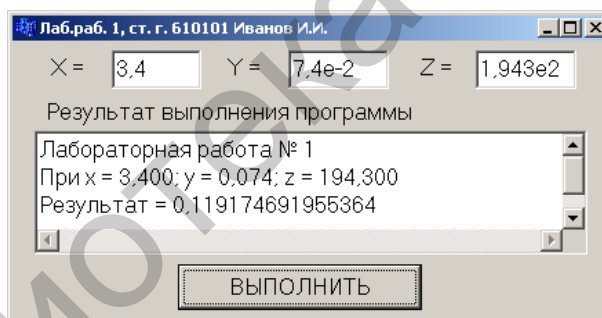


Рис. 1.3

#### 1.4.2. Создание консольного приложения в *Builder*

Чтобы создать проект в консольном приложении, выполняем следующую последовательность действий: *File* → *Close All* → *File* → *New* → *Other* → *Console Wizard* → *Ok*.

#### 1.4.3. Создание консольного приложения в *Visual*

Чтобы создать проект в консольном приложении, выполняем следующую последовательность действий: *File* → *New* → *Project* → *C++ Win32 Console Application* → *Empty Project* → *Ok*. Далее *File* → *New* → *File* → *C++ Source File* → *Ok*. Если работаем со средой *Microsoft Visual System (MVS)* 6.0, то создание консольного приложения на этом завершено.

В версиях *MVS* начиная с 2005 года мы должны присоединить к только что созданному проекту файл-источник: *File* → *Move Source to* → *Наш проект*.

В более поздних версиях *MVS* некоторые библиотеки могут использоваться без расширения (например, *iostream*, *ioomanip*). Для того чтобы проект был собран корректно, мы используем:

*using namespace std;*

Текст программы может иметь следующий вид:

```
#include <vcl.h> // Только для Builder
#include <stdio.h> // include <iostream.h>
#include <conio.h>
#include <math.h>
#pragma hdrstop // Только для Builder
#pragma argsused // Только для Builder
int main(int argc, char* argv[])
{
    double x, y, z, a, b, c, rez;
    puts("\n\tx,y,z = "); // cout << "x, y, z = ";
    scanf("%lf%lf%lf", &x, &y, &z); // cin >> x >> y >> z;
    a = pow(tan(x+y),2);
    b = exp(y - z);
    c = sqrt(cos(x*x)+sin(z*z));
    rez = a - b * c;
    printf("\n x = %7.3lf\n y = %7.3lf\n z = %7.3lf\nRezult = %lf\n", x, y, z, rez);
    //cout<<"x="<<x<<"\ny="<<y<<"\nz="<<z<<"\nResult="<<rez<<endl;
    getch(); // system("pause");
    return 0;
}
```

## 1.5. Индивидуальные задания

### 1.5.1. Первый уровень сложности

Составить программу для расчета двух значений  $z_1$  и  $z_2$ , результаты которых должны совпадать. Значения исходных данных вводить с клавиатуры,  $\pi = 3.1415926$ ,  $\alpha$  и  $\beta$  – в радианах.

$$1. \ z_1 = 2 \sin^2(3\pi - 2\alpha) \cos^2(5\pi + 2\alpha), \ z_2 = 1/4 - 1/4 \sin(5/2 \pi - 8\alpha).$$

$$2. \ z_1 = \cos \alpha + \sin \alpha + \cos 3\alpha + \sin 3\alpha, \ z_2 = 2\sqrt{2} \cos \alpha \cdot \sin(\pi/4 + 2\alpha).$$

$$3. \ z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos \alpha + 1 - 2 \sin^2 2\alpha}, \ z_2 = 2 \sin \alpha.$$

$$4. \ z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\beta}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\beta}{4}\right), \ z_2 = \frac{\sqrt{2}}{2} \sin \frac{\beta}{2}.$$

$$5. \ z_1 = 1 - \frac{1}{4} \sin^2 2\alpha + \cos 2\alpha, \ z_2 = \cos^2 \alpha + \cos^4 \alpha.$$

$$6. \ z_1 = \cos \alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha, \ z_2 = 4 \cos \frac{\alpha}{2} \cdot \cos \frac{5}{2}\alpha \cdot \cos 4\alpha.$$

7.  $z_1 = \cos^4 \alpha + \sin^2 \beta + \frac{1}{4} \sin^2 2\alpha - 1$ ,  $z_2 = \sin(\beta + \alpha) \cdot \sin(\beta - \alpha)$ .
8.  $z_1 = (\cos \alpha - \cos \beta)^2 - (\sin \alpha - \sin \beta)^2$ ,  $z_2 = -4 \sin^2 \frac{\alpha - \beta}{2} \cdot \cos(\alpha + \beta)$ .
9.  $z_1 = \frac{\sin(\pi/2 + 3\alpha)}{1 - \sin(3\alpha - \pi)}$ ,  $z_2 = \operatorname{ctg}\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right)$ .
10.  $z_1 = \frac{1 - 2\sin^2 \alpha}{1 + \sin 2\alpha}$ ,  $z_2 = \frac{1 - \operatorname{tg} \alpha}{1 + \operatorname{tg} \alpha}$ .
11.  $z_1 = \frac{\sin 4\alpha}{1 + \cos 4\alpha} \cdot \frac{\cos 2\alpha}{1 + \cos 2\alpha}$ ,  $z_2 = \operatorname{ctg}\left(\frac{3}{2}\pi - \alpha\right)$ .
12.  $z_1 = \frac{\sin \alpha + \cos(2\beta - \alpha)}{\cos \alpha - \sin(2\beta - \alpha)}$ ,  $z_2 = \frac{1 + \sin 2\beta}{\cos 2\beta}$ .
13.  $z_1 = \frac{\sqrt{(3\alpha + 2)^2 - 24\alpha}}{3\sqrt{\alpha} - 2/\sqrt{\alpha}}$ ,  $z_2 = -\sqrt{\alpha}$ .
14.  $z_1 = \frac{(\alpha - 1)\sqrt{\alpha} - (\beta - 1)\sqrt{\beta}}{\sqrt{\alpha^3\beta + \alpha\beta + \alpha^2 - \alpha}}$ ,  $z_2 = \frac{\sqrt{\alpha} - \sqrt{\beta}}{\alpha}$ .
15.  $z_1 = \frac{\sqrt{2\alpha + 2\sqrt{\alpha^2 - 4}}}{\sqrt{\alpha^2 - 4} + \alpha + 2}$ ,  $z_2 = \frac{1}{\sqrt{\alpha + 2}}$ .

### 1.5.2. Второй уровень сложности

Составить программу для расчета заданных выражений с проверкой исключительных ситуаций: деление на ноль, вычисление корня из отрицательного числа и т. п. При вводе данных использовать проверку на ввод нечисловых данных.

1.  $z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos \alpha + 1 - 2\sin^2 2\alpha}$ ,  $z_2 = 2\sin \alpha$ .
2.  $z_1 = \frac{\sin(\pi/2 + 3\alpha)}{1 - \sin(3\alpha - \pi)}$ ,  $z_2 = \operatorname{ctg}\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right)$ .
3.  $z_1 = \frac{1 - 2\sin^2 \alpha}{1 + \sin 2\alpha}$ ,  $z_2 = \frac{1 - \operatorname{tg} \alpha}{1 + \operatorname{tg} \alpha}$ .
4.  $z_1 = \frac{\sin 4\alpha}{1 + \cos 4\alpha} \cdot \frac{\cos 2\alpha}{1 + \cos 2\alpha}$ ,  $z_2 = \operatorname{ctg}\left(\frac{3}{2}\pi - \alpha\right)$ .
5.  $z_1 = \frac{\sin \alpha + \cos(2\beta - \alpha)}{\cos \alpha - \sin(2\beta - \alpha)}$ ,  $z_2 = \frac{1 + \sin 2\beta}{\cos 2\beta}$ .



6.  $z_1 = \frac{\sqrt{(3\alpha+2)^2 - 24\alpha}}{3\sqrt{\alpha} - 2/\sqrt{\alpha}}, \quad z_2 = -\sqrt{\alpha}.$
7.  $z_1 = \frac{(\alpha-1)\sqrt{\alpha} - (\beta-1)\sqrt{\beta}}{\sqrt{\alpha^3\beta + \alpha\beta + \alpha^2} - \alpha}, \quad z_2 = \frac{\sqrt{\alpha} - \sqrt{\beta}}{\alpha}.$
8.  $z_1 = \frac{\sqrt{2\alpha + 2\sqrt{\alpha^2 - 4}}}{\sqrt{\alpha^2 - 4} + \alpha + 2}, \quad z_2 = \frac{1}{\sqrt{\alpha + 2}}.$
9.  $z_1 = \frac{\alpha^2 + 2\alpha - 3 + (\alpha+1)\sqrt{\alpha^2 - 9}}{\alpha^2 - 2\alpha - 3 + (\alpha-1)\sqrt{\alpha^2 - 9}}, \quad z_2 = \sqrt{\frac{\alpha+3}{\alpha-3}}.$
10.  $z_1 = \left( \frac{\alpha+2}{\sqrt{2\alpha}} - \frac{\alpha}{\sqrt{2\alpha+2}} + \frac{2}{\alpha - \sqrt{2\alpha}} \right) \cdot \frac{\sqrt{\alpha} - \sqrt{2}}{\alpha+2}, \quad z_2 = \frac{1}{\sqrt{\alpha+2}}.$
11.  $z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\beta}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\beta}{4}\right), \quad z_2 = \frac{\sqrt{2}}{2} \sin \frac{\beta}{2}.$
12.  $z_1 = 2\sin^2(3\pi - 2\alpha)\cos^2(5\pi + 2\alpha), \quad z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right).$
13.  $z_1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha, \quad z_2 = 2\sqrt{2}\cos\alpha \cdot \sin\left(\frac{\pi}{4} + 2\alpha\right).$
14.  $z_1 = 1 - \frac{1}{4}\sin^2 2\alpha + \cos 2\alpha, \quad z_2 = \cos^2\alpha + \cos^4\alpha$
15.  $z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2, \quad z_2 = -4\sin^2 \frac{\alpha - \beta}{2} \cdot \cos(\alpha + \beta).$

### 1.5.3. Третий уровень сложности

Составить программу для расчета заданного значения с проверкой исключительных ситуаций: деление на нуль, выход значений аргументов используемых функций за допустимые пределы и т. п. При вводе данных использовать проверку на ввод нечисловых данных.

$$1. \quad t = \frac{2\cos(x - \pi/6)}{0,5 + \sin^2 y} \left( 1 + \frac{z^2}{3 - z^2/5} \right).$$

При  $x = 14.26, y = -1.22,$   
 $z = 3.5 \cdot 10^{-2} \rightarrow t = 0.564849.$

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

При  $x = -4.5, y = 0.75 \cdot 10^{-4},$   
 $z = 0.845 \cdot 10^2 \rightarrow u = -55.6848.$

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

При  $x = 3.74 \cdot 10^{-2}, y = -0.825,$   
 $z = 0.16 \cdot 10^2 \rightarrow v = 1.0553.$

$$4. w = |\cos x - \cos y|^{(1+2\sin^2 y)} \left( 1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4} \right). \quad \text{При } x = 0.4 \cdot 10^4, y = -0.875, \\ z = -0.475 \cdot 10^{-3} \rightarrow w = 1.9873.$$

$$5. \alpha = \ln(y^{-\sqrt{|x|}})(x - y/2) + \sin^2(\operatorname{arctg}(z)). \quad \text{При } x = -15.246, y = 4.642 \cdot 10^{-2}, \\ z = 20.001 \cdot 10^2 \rightarrow \alpha = -182.036.$$

$$6. \beta = \sqrt{10(\sqrt[3]{x} + x^{y+2})} \cdot (\arcsin^2 z - |x - y|). \quad \text{При } x = 16.55 \cdot 10^{-3}, y = -2.75, \\ z = 0.15 \rightarrow \beta = -38.902.$$

$$7. \gamma = 5 \operatorname{arctg}(x) - \frac{1}{4} \arccos(x) \frac{x + 3|x - y| + x^2}{|x - y|z + x^2}. \quad \text{При } x = 0.1722, y = 6.33, \\ z = 3.25 \cdot 10^{-4} \rightarrow \gamma = -172.025.$$

$$8. \varphi = \frac{e^{|x-y|}|x-y|^{x+y}}{\operatorname{arctg} x + \operatorname{arctg} z} + \sqrt[3]{x^6 + \ln^2 y}. \quad \text{При } x = -2.235 \cdot 10^{-2}, y = 2.23, \\ z = 15.221 \rightarrow \varphi = 39.374.$$

$$9. \psi = |x^{y/x} - \sqrt[3]{y/x}| + (y - x) \frac{\cos y - z / (y - x)}{1 + (y - x)^2}. \quad \text{При } x = 1.825 \cdot 10^2, y = 18.225, \\ z = -3.298 \cdot 10^{-2} \rightarrow \psi = 1.2131.$$

$$10. a = 2^{-x} \sqrt{x + \sqrt[4]{y}} \sqrt[3]{e^{x-1/\sin z}}. \quad \text{При } x = 3.981 \cdot 10^{-2}, y = -1.625 \cdot 10^3, \\ z = 0.512 \rightarrow a = 1.2619.$$

$$11. b = y^{\sqrt[3]{|x|}} + \cos^3 y \frac{|x - y| \cdot \left( 1 + \frac{\sin^2 z}{\sqrt{x + y}} \right)}{e^{|x-y|} + x/2}. \quad \text{При } x = 6.251, y = 0.827, \\ z = 25.001 \rightarrow b = 0.7121.$$

$$12. c = 2^{y^x} + (3^x)^y - \frac{y \cdot (\operatorname{arctg}(z) - \pi/6)}{|x| + \frac{1}{y^2 + 1}}. \quad \text{При } x = 3.251, y = 0.325, \\ z = 0.466 \cdot 10^{-4} \rightarrow c = 4.025.$$

$$13. f = \frac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x - y| (\sin^2 z + \operatorname{tg} z)}. \quad \text{При } x = 17.421, y = 10.365 \cdot 10^{-3}, \\ z = 0.828 \cdot 10^5 \rightarrow f = 0.33056.$$

$$14. g = \frac{y^{x+1}}{\sqrt[3]{|y-2|} + 3} + \frac{x + y/2}{2|x + y|} (x + 1)^{-1/\sin z}. \quad \text{При } x = 12.3 \cdot 10^{-1}, y = 15.4, \\ z = 0.252 \cdot 10^3 \rightarrow g = 82.8257.$$

$$15. h = \frac{x^{y+1} + e^{y-1}}{1 + x|y - \operatorname{tg} z|} (1 + |y - x|) + \frac{|y - x|^2}{2} - \frac{|y - x|^3}{3}. \quad \text{При } x = 2.444, y = 0.869 \cdot 10^{-2}, \\ z = -0.13 \cdot 10^3 \rightarrow h = -0.49871.$$

$$16. w = \sqrt[3]{x^6 + \ln^2 y} + \frac{e^{|x-y|}|x-y|^{x+y}}{\operatorname{arctg}(x) + \operatorname{arctg}(z)}. \quad \text{При } x = -2.235 \cdot 10^{-2}, y = 2.23, \\ z = 15.221 \rightarrow w = 39.374.$$

## Тема №2. Реализация разветвляющихся алгоритмов

**Цель работы:** изучить операции сравнения, логические операции, операторы передачи управления *if*, *switch*, *break*, научиться пользоваться простейшими компонентами организации переключений (*CheckBox*, *RadioGroup*). Написать и отладить программу с разветвлениями.

### 2.1. Общие теоретические сведения

#### **Оператор условной передачи управления *if***

Для выбора одной из ветвей вычислений применяется оператор условного перехода:

```
if (Выражение) Оператор 1;  
    else Оператор 2;
```

Вычисляется *Выражение*, и если его значение не равно 0 (истинно), то выполняется *Оператор 1*, иначе – *Оператор 2*, например:

```
if(x > y) max = x;  
    else max = y;
```

Если *Операторы 1, 2* содержат более одного оператора, то они заключаются в фигурные скобки { }, т. е. создается **блок**.

Конструкция ***else...*** может отсутствовать, и такую форму называют *сокращенной*, тогда в случае ложности условия управление передается на следующий за *if* оператор.

Если *Операторы 1, 2*, в свою очередь, являются операторами *if*, то такой оператор называют *вложенным*, при этом ключевое слово *else* принадлежит ближайшему предшествующему *if*.

Например, найти наибольшее значение из трех чисел *x, y, z*:

```
if (x > y) max=x;  
    else max=y;  
if(z > max) max=z;
```

**Операции сравнения:** < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), != (не равно), == (равно – **два знака равенства**).

Операции сравнения бинарные, их общий вид:

```
Операнд 1 Операция Операнд 2
```

Операндами операций сравнения могут быть данные любых базовых типов, значения которых перед сравнением преобразуются к одному типу.

**Логические операции** используются при составлении более сложных выражений. Приведем их перечень в порядке убывания приоритета:

! (отрицание, или логическое НЕ – унарная операция);

&& (конъюнкция, или логическое умножение И);

|| (дизъюнкция, или логическое сложение ИЛИ).

Например:  $(0 < x) \&\& (x \leq 100)$   
 $((!x) \&\& (y > 0)) \parallel ((z == 1) \&\& (k > 0))$

Выражения вычисляются слева направо, причем их вычисление прекращается, как только результат становится известен.

### **Тернарная (условная) операция ?:**

Ее общая форма:

*Операнд 1 ? Операнд 2 : Операнд 3*

Если значение *Операнда 1* истинно (не равно 0), то результатом операции является *Операнд 2*, иначе – *Операнд 3*.

Например, найти наибольшее из двух чисел:  $\max = a > b ? a : b;$

### **Оператор выбора switch**

Общая форма оператора выбора (переключателя):

```
switch (Выражение)  
{  
    case Const1 : Операторы; break;  
    ...  
    case ConstN : Операторы; break;  
    default : Операторы;  
}
```

Вычисляется *Выражение* и проверяется, совпадает ли его результат со значением одной из констант. При совпадении выполняются операторы этого *case*. Значениями *Const1*, ..., *ConstN* могут быть только целые или символьные константы. Необязательная конструкция *default* выполняется, если результат выражения не совпал ни с одной из констант.

Оператор **break** выполняет досрочный выход из *switch* (после выполнения соответствующей ветви *case* все остальные будут опущены). Если оператор **break** в *case* не записан, то будут выполняться операторы следующих ветвей *case* до появления оператора **break** либо до завершения оператора *switch*.

## **2.2. Создание оконного приложения в среде Builder**

При создании оконного приложения для организации разветвлений используются компоненты в виде кнопок-переключателей. Состояние такой кнопки (включено/выключено) визуальное отражается на форме. Кнопки-переключатели бывают двух типов: *TCheckBox* и *TRadioGroup*.

Компонента **CheckBox** создает кнопку независимого переключателя, с помощью которой можно указать свое решение типа да/нет. В программе состояние кнопки связано со значением булевой переменной (свойство **Checked**), которая проверяется с помощью оператора *if*.

Компонента **RadioGroup** создает группу кнопок – зависимых переключателей. При нажатии одной из кнопок группы все остальные кнопки отключаются. В программу передается номер включенной кнопки (0, 1, 2, ... – свойство **ItemIndex**), который анализируется с помощью оператора **switch**.

В языке C++ используются переменные типа **bool**, которые могут принимать только два значения – **true** и **false** (истина – 1, ложь – 0).

### 2.3. Пример выполнения задания

Ввести исходные данные  $x$ ,  $y$ ,  $z$ . Вычислить значение  $u$  в зависимости от выбора:  $\sin(x)$ ,  $\cos(x)$  или  $\operatorname{tg}(x)$ , после чего определить максимальное либо из  $u$ ,  $y$ ,  $z$ , либо из их модулей –  $|u|$ ,  $|y|$ ,  $|z|$ .

#### 2.3.1. Реализация примера в оконном приложении

Создать форму, которая может иметь вид, представленный на рис. 2.1, скорректировав указанные тексты, написать соответствующую программу.

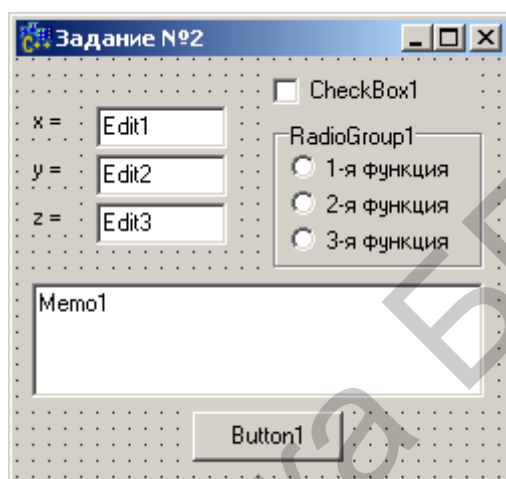




Рис. 2.1

#### Компонента *CheckBox*

В меню *Standard* выберите пиктограмму , поместив ее на форму, измените заголовок *Caption*. В тексте *Unit1.h* появится переменная *CheckBox1* типа *TCheckBox*. В зависимости от того, сделан выбор или нет, булева переменная *CheckBox1->Checked* будет принимать значение *true* или *false*.

#### Компонента *RadioGroup*

В меню *Standard* выберите пиктограмму , поместите ее в нужное место формы. На форме появится окаймленный линией чистый прямоугольник с заголовком *RadioGroup1*. Замените заголовок (*Caption*) на «Выбор функции».

В инспекторе объектов в свойстве *Items* вызовите строчный редактор списка заголовков кнопок и наберите три строки с именами выбираемых функций ( $\cos(x)$ ,  $\sin(x)$ ,  $\operatorname{tg}(x)$ ), нажмите *OK*. После этого внутри компоненты появятся три кнопки-переключателя с введенными надписями.

В *Unit1.h* появилась переменная *RadioGroup1* типа *TRadioGroup*. Теперь при нажатии одной из кнопок группы переменная *RadioGroup1->ItemIndex* будет равна номеру выбранной строчки (начинается с нуля).

В функции *FormCreate()* желательно установить начальное значение этой переменной, например *RadioGroup1->ItemIndex = 0*.

Подготовив форму, создайте функции-обработчики событий *FormCreate* и *Button1Click* аналогично первой работе.

Приведем части текста программы для реализации поставленной задачи.

1. Проверка номера нажатой кнопки для выбора соответствующей функции:

```
switch(RadioGroup1->ItemIndex)
{
    case 0: u = cos(x);
            Memo1->Lines->Add("Cos(x)= " + FloatToStrF(u,ffFixed,8,6));
            break;
    case 1: u = sin(x);
            Memo1->Lines->Add("Sin(x)= " + FloatToStrF(u,ffFixed,8,6));
            break;
    case 2: u = tan(x);
            Memo1->Lines->Add("Tg(x)= " + FloatToStrF(u,ffFixed,8,6));
            break;
}
```

2. Проверка состояния кнопки *CheckBox1*:

```
if (CheckBox1->Checked) {
    u = fabs(u); y = fabs(y); z = fabs(z);
}
```

3. Вывод информации в зависимости от состояния кнопки *CheckBox1*:

```
if (CheckBox1->Checked)
    Memo1->Lines->Add("Max модулей = " + FloatToStr(ma));
else Memo1->Lines->Add("Max = " + FloatToStr (ma));
```

### 2.3.2. Реализация примера в консольном приложении

Приведем части текста программы для реализации поставленной задачи.

1. Выбора соответствующей функции (ввели значение переменной *kod*):

```
cout << "0 – cos (And Default), 1 – sin, 2 – tg " << endl;
cin >> kod;
switch ( kod ) {
    case 0: default:
        u = cos(x); cout<<" Cos(x) (Default) = "<<u<<endl; break;
    case 1:      u = sin(x);  cout<<" Sin(x)= "<<u<<endl; break;
    case 2:      u = tan(x);  cout<<" Tg(x)= "<<u<<endl; break;
}
```

2. Выбор расчета максимума (с модулем или без):

```
cout<<"Max abs - 1, Else - Max";
cin>>kod;
if (kod == 1) {
    u = fabs(u); y = fabs(y); z = fabs(z);
}
```

3. Вывод информации в зависимости от выбора в п. 2:

```
if (kod == 1)    cout << "Max abs = " << ma << endl;
else            cout << "Max = " << ma << endl;
```

## 2.4. Индивидуальные задания

### 2.4.1. Первый уровень сложности

Составить программу для определения значения  $y$ . **Обязательно** вывести сообщения о том, по какой ветви происходило вычисление значения переменной  $x$ .

$$1. y = (\ln(1+x^2) + \cos(x+1)) e^{k \cdot x}, \quad \text{где } x = \begin{cases} k \cdot z^3, & k < 1, \\ z \cdot (z+1), & k \geq 1. \end{cases}$$

$$2. y = \frac{a \cdot x + b \cdot x \cdot \cos \sqrt{x}}{x + a \cdot b}, \quad \text{где } x = \begin{cases} \sqrt{a^2 + a^2 \cdot z}, & z < a \cdot b, \\ \sin^2 z + |a \cdot b \cdot z|, & z \geq a \cdot b. \end{cases}$$

$$3. y = -\pi + \cos^2 x^3 + \sin^3 x^2, \quad \text{где } x = \begin{cases} z/b, & z < 1, \\ \sqrt{(z \cdot b)^3}, & z \geq 1. \end{cases}$$

$$4. y = \cos^3 x^2 + \sin^2 x^3, \quad \text{где } x = \begin{cases} z^3 + 0.2, & z < 1, \\ z + \ln z, & z \geq 1. \end{cases}$$

$$5. y = \ln(x + 0,5) + (e^x - e^{-x}), \quad \text{где } x = \begin{cases} -z/3, & z < -1, \\ |z|, & z \geq -1. \end{cases}$$

$$6. y = \frac{2}{3} \sin^2 x - \frac{3}{4} \cos^2 x, \quad \text{где } x = \begin{cases} z, & z < 0, \\ \sin z, & z \geq 0. \end{cases}$$

$$7. y = \sin^3(c \cdot x + d^2 + k \cdot x^2), \quad \text{где } x = \begin{cases} z^2 - z, & z < 0, \\ z^3, & z \geq 0. \end{cases}$$

$$8. y = \sin^2 x + \cos^5 x^3 + \ln x^{2/5}, \quad \text{где } x = \begin{cases} 2z + 1, & z \geq 0, \\ \ln(z^2 - z), & z < 0. \end{cases}$$

$$9. y = \frac{1}{\cos x} + \ln \left| \operatorname{tg} \frac{x}{2} \right|, \quad \text{где } x = \begin{cases} z^b + |b/2|, & z \leq 0, \\ \sqrt{z}, & z > 0. \end{cases}$$

$$10. y = \frac{e^{\sin^3 x} + \ln(x+1)}{\sqrt{x}}, \quad \text{где } x = \begin{cases} z-1, & z \geq 1, \\ z^2 + 1, & z < 1. \end{cases}$$

$$11. y = \frac{2e^{-3x} - 4x^2}{\ln|x| + x}, \quad \text{где } x = \begin{cases} \frac{1}{z^2 + 2z}, & z > 0, \\ 1 - z^3, & z \leq 0. \end{cases}$$

$$\begin{aligned}
12. \quad y &= \sin^3(x^2 - 1) + \ln|x| + e^x, & \text{где } x &= \begin{cases} z^2 + 5, & z \leq 0, \\ \frac{1}{\sqrt{z-1}}, & z > 0. \end{cases} \\
13. \quad y &= \sin(n \cdot x) + \cos(k \cdot x) + \ln(m \cdot x), & \text{где } x &= \begin{cases} e^z + z, & z > 1, \\ z^2 + 1, & z \leq 1. \end{cases} \\
14. \quad y &= \cos 5x + \sin \frac{1}{5}x + e^x, & \text{где } x &= \begin{cases} \sqrt{z}, & z > 0, \\ (3z^3 - z) - 5, & z \leq 0. \end{cases} \\
15. \quad y &= x(\sin x + e^{-(x+3)}), & \text{где } x &= \begin{cases} -3z, & z > 0, \\ z^2, & z \leq 0. \end{cases}
\end{aligned}$$

#### 2.4.2. Второй уровень сложности

Вычислить значение  $y$  в зависимости от выбранной функции  $\phi(x)$ :  $2x$ ,  $x^2$ ,  $x/3$ ), аргумент которой  $x$  определяется в зависимости от переменной  $z$ . **Обязательно** выводить сообщения, показывающие, при каком условии и с какой функцией производились вычисления  $y$ .

При вводе данных использовать проверку на ввод нечисловых данных.

Организовать проверку исключительных ситуаций: деление на нуль, вычисление корня из отрицательного числа, выход значений аргументов используемых функций за допустимые пределы.

$$\begin{aligned}
1. \quad y &= a \ln(1 + x^{1/5}) + \cos^2[\phi(x) + 1], & \text{где } x &= \begin{cases} z^2, & z < 1, \\ z + 1, & z \geq 1. \end{cases} \\
2. \quad y &= \frac{2a\phi(x) + b \cos \sqrt{|x|}}{x^2 + 5}, & \text{где } x &= \begin{cases} 2 + z, & z < 1, \\ \sin^2 z, & z \geq 1. \end{cases} \\
3. \quad y &= -\pi\phi(x) + a \cos^2 x^3 + b \sin^3 x^2, & \text{где } x &= \begin{cases} z, & z < 1, \\ \sqrt{z^3}, & z \geq 1. \end{cases} \\
4. \quad y &= 2a \cos^3 x^2 + \sin^2 x^3 - b\phi(x), & \text{где } x &= \begin{cases} z^3 + 0.2, & z < 1, \\ z + \ln z, & z \geq 1. \end{cases} \\
5. \quad y &= a\phi(x) - \ln(x + 2,5) + b(e^x - e^{-x}), & \text{где } x &= \begin{cases} -z/3, & z < -1, \\ |z|, & z \geq -1. \end{cases} \\
6. \quad y &= \frac{2}{3}a \sin^2 x - \frac{3b}{4} \cos^2 \phi(x), & \text{где } x &= \begin{cases} z, & z < 0, \\ \sin z, & z \geq 0. \end{cases}
\end{aligned}$$



$$7. y = \sin^3[c\phi(x) + d^2 + x^2],$$

$$\text{где } x = \begin{cases} z^2 - z, & z < 0, \\ z^3, & z \geq 0. \end{cases}$$

$$8. y = \sin^2 \phi(x) + a \cos^5 x^3 + c \ln x^{2/5},$$

$$\text{где } x = \begin{cases} 2z + 1, & z \geq 0, \\ \ln(z^2 - z), & z < 0. \end{cases}$$

$$9. y = \frac{b\phi(x)}{\cos x} + a \ln \left| \operatorname{tg} \frac{x}{2} \right|,$$

$$\text{где } x = \begin{cases} z^2 / 2, & z \leq 0, \\ \sqrt{z}, & z > 0. \end{cases}$$

$$10. y = \frac{d\phi(x)e^{\sin^3 x} + c \ln(x+1)}{\sqrt{x}},$$

$$\text{где } x = \begin{cases} z^2 + 1, & z < 1, \\ z - 1, & z \geq 1. \end{cases}$$

$$11. y = \frac{2.5a \cdot e^{-3x} - 4bx^2}{\ln |x| + \phi(x)},$$

$$\text{где } x = \begin{cases} \frac{1}{z^2 + 2z}, & z > 0, \\ 1 - z^3, & z \leq 0. \end{cases}$$

$$12. y = a \sin^3[\phi(x)^2 - 1] + c \ln |x| + e^x,$$

$$\text{где } x = \begin{cases} z^2 + 1, & z \leq 1, \\ 1/\sqrt{z-1}, & z > 1. \end{cases}$$

$$13. y = \sin[n\phi(x)] + \cos kx + \ln mx,$$

$$\text{где } x = \begin{cases} z, & z > 1, \\ z^2 + 1, & z \leq 1. \end{cases}$$

$$14. y = b \cos[a\phi(x)] + \sin \frac{x}{5} + ae^x,$$

$$\text{где } x = \begin{cases} \sqrt{z}, & z > 0, \\ 3z + 1, & z \leq 0. \end{cases}$$

$$15. y = 2\phi(x)[a \sin x + d \cdot e^{-(x+3)}],$$

$$\text{где } x = \begin{cases} -3z, & z > 0, \\ z^2, & z \leq 0. \end{cases}$$

### 2.4.3. Третий уровень сложности

Составить программу нахождения требуемого значения с указанными исходными данными. Поиск минимального (min) и максимального (max) значений элементов организовать в виде отдельной функции.

При вводе данных использовать проверку на ввод нечисловых данных.

Организовать проверку исключительных ситуаций.

$$1. m = \frac{\max(x, y, z)}{\min(x, y)} + 5.$$

$$2. m = \frac{\min(x + y, y - z)}{\max(x, y, z)}.$$

$$3. m = \frac{\max(x + y + z, x \cdot y \cdot z)}{\min(x + y + z, x \cdot y \cdot z)}.$$

$$4. m = \frac{\min[\max(x, y), \max(y, z)]}{\max(y, z)}.$$

$$5. m = \frac{\min(z, x) + \min(x, y)}{\max^2(x, y, z)}.$$

$$6. m = \frac{\min(y, z)}{\max[\min(x, y), \min(y, z)]}.$$

$$7. m = \frac{\min(x + y + z, x \cdot y \cdot z)}{\min(x - y + z, x \cdot y / z)}.$$

$$8. m = \frac{\max(x + y + z, x \cdot y \cdot z)}{\max[x + y + z, x / (y \cdot z)]}.$$

$$9. m = \frac{\max(y, z)}{\min[\min(x, y), \min(y, z)]}.$$

$$10. m = \frac{\max[\max(x, y), \max(y, z)]}{\min(x, y, z)}.$$

$$11. F = \begin{cases} \min(x^2, y^2) + a, & a < 0, \\ \max(y, x, a), & a = 0, \\ |x - y| + y(x + \sqrt{a^3}), & a > 0. \end{cases}$$

$$12. F = \begin{cases} \max(\sqrt{x}, y) + a, & \text{при } a < 0, \\ \min(x, a), & \text{при } a = 0, \\ \sqrt{x^2 + y^2} + e^{y-x}, & \text{при } a > 0. \end{cases}$$

$$13. F = \begin{cases} \max(x, y) + \sqrt{x}, & \text{при } |x| + |y| \leq 1 \text{ или } x \geq 0, \\ \min(x, y) + \sin^2 x - \cos y^2, & \text{при } |x| + |y| > 0 \text{ или } x < 0, y < 0, \\ e^{x^2 + |y|}, & \text{иначе.} \end{cases}$$

$$14. F = \begin{cases} \max(x, y + \sqrt{x}), & \text{при } x > 0, y \geq 0, \\ \min(x, y) + \sin^2 x - \cos y^2, & \text{при } x < 0, \\ 0,5x + e^y, & \text{иначе.} \end{cases}$$

$$15. F = \begin{cases} \min(0.9y, e^{2x-3}), & \text{при } x \leq 0, \\ \frac{2\cos(x - \pi/6) + \sqrt[3]{y}}{5 - 2x}, & \text{при } x \geq 0, y > 0, \\ \max(\sin^2 y, \cos^2 y), & \text{иначе.} \end{cases}$$

### Тема №3. Реализация циклических алгоритмов

**Цель работы:** изучить циклические операторы *while*, *do-while*, *for*, научиться реализовывать циклические алгоритмы. Изучив простейшие средства отладки программ, составить и отладить программу.

#### 3.1. Общие теоретические сведения

Под циклом понимается многократное выполнение одних и тех же операторов при различных значениях промежуточных данных. Число повторений может быть задано в явной или неявной формах. Для организации повторений в языке C++ используются три различных оператора цикла.

##### 1. Оператор цикла с предусловием

***while* (Выражение)**

*Код цикла*

организует повторение операторов *Кода цикла* до тех пор, пока *Выражение* истинно (не равно 0); если *Выражение* ложно (равно 0) при первом входе, то код цикла не выполнится ни разу. Если *Код цикла* состоит более чем из одного оператора, то организуется блок.

##### 2. Оператор цикла с постусловием

***do***

*Код цикла*

***while* (Выражение);**

организует повторение *Кода цикла* до тех пор, пока *Выражение* истинно, после чего управление передается следующему за циклом оператору. Данный оператор гарантирует выполнение *Кода цикла* хотя бы один раз.

##### 3. Оператор с предусловием и коррекцией

***for* (Выражение1; Выражение2; Выражение3)**

*Код цикла*

где *Выражение1* – начальное значение параметра цикла; *Выражение2* – проверка условия на продолжение цикла; *Выражение3* – изменение (коррекция) параметра цикла.

Сначала вычисляется *Выражение1*, затем проверяется *Выражение2*: если оно истинно, то выполняется *Код цикла*, затем вычисляется *Выражение3*, и так до тех пор, пока *Выражение2* не примет значение «ложь».

Досрочный выход из операторов цикла выполняет оператор ***break***, а оператор ***continue*** выполняет передачу управления в головной оператор цикла.

#### ***Средства отладки программ в C++ Builder***

Практически в каждой вновь написанной программе после запуска обнаруживаются ошибки.

*Синтаксические* ошибки связаны с неправильной записью операторов. При обнаружении таких ошибок компилятор *Builder* останавливается напротив

оператора, в котором она обнаружена. В нижней части экрана появляется текстовое окно, содержащее сведения обо всех ошибках, найденных в проекте.

Для быстрого перехода к интересующей ошибке нужно дважды щелкнуть кнопкой мыши на строке с ее описанием, а для получения более полной информации – обратиться к *HELP* нажатием клавиши *F1*.

*Семантические* ошибки связаны с неверным выбором алгоритма решения или с неправильной программной реализацией задачи. Эти ошибки проявляются обычно в том, что результат расчета оказывается неверным (переполнение, деление на нуль и др.), поэтому перед использованием отлаженной программы ее надо протестировать, т. е. выполнить при значениях исходных данных, для которых заранее известен результат. Если тестовые расчеты указывают на ошибку, то для ее поиска следует использовать встроенные средства отладки.

В простейшем случае для поиска места ошибки рекомендуется выполнить следующие действия.

В окне редактирования текста установить курсор в строке перед подозрительным участком и нажать клавишу *F4* (выполнение до курсора) – выполнение программы будет остановлено на строке, содержащей курсор. Чтобы увидеть, чему равно значение интересующей переменной, нужно поместить на переменную курсор – на экране будет высвечено ее значение, либо нажать *Ctrl+F7* и в появившемся диалоговом окне указать эту переменную (с помощью данного окна можно также изменить значение переменной во время выполнения программы).

Нажимая клавишу *F7* (пошаговое выполнение), построчно выполнять программу, контролируя изменение тех или иных переменных и правильность вычислений. Если курсор находится внутри цикла, то после нажатия *F4* расчет останавливается после одного выполнения кода цикла.

Для продолжения расчетов следует нажать *<Run>* в меню *Run*.

### 3.2. Пример выполнения задания

Написать и отладить программу вывода всех значений функции  $S(x)$  для аргумента  $x$ , изменяющегося в интервале от  $a$  до  $b$  с шагом  $h$  и заданным  $n$ .

$$S(x) = \sum_{k=0}^n (-1)^k \frac{x^k}{k!}.$$

В начале составления алгоритма необходимо получить рекуррентную формулу. Для получения формулы рассмотрим значение слагаемого при различных значениях  $k$ :

при  $k=1$   $r_1 = -1 \frac{x}{1}$ ; при  $k=2$   $r_2 = 1 \frac{x \cdot x}{1 \cdot 2}$ ; при  $k=3$

$r_3 = -1 \frac{x \cdot x \cdot x}{1 \cdot 2 \cdot 3}$  и т. д. Видно, что на каждом шаге слагаемое дополнительно

умножается на  $-1 \frac{x}{k}$ . Исходя из этого формула рекуррентной последовательно-

сти будет иметь вид  $r_k = -r_{k-1} \frac{x}{k}$ . Начальные значения получаются при  $k = 0$  и равны  $r_0 = 1$  и  $S_0 = 1$ . Полученная формула позволяет избавиться от многократного вычисления факториала и возведения в степень.

Часть кода, реализующая алгоритм нахождения суммы, может выглядеть следующим образом:

```
r = s = 1;           // Начальные значения, полученные при k = 0
for(k = 1; k <= n; k++)
{
    r = -r*x/k;       // Расчет рекуррентного члена ряда
    s += r;           // Суммирование членов ряда
}
```

### 3.3. Индивидуальные задания

#### 3.3.1. Первый уровень сложности

Составить программу для определения всех значений функции  $y$  в произвольном диапазоне  $[a, b]$  изменения аргумента  $x$  с произвольным шагом  $h$ . Значения  $a, b, h$  вводятся с клавиатуры. Результат должен содержать следующие столбцы: порядковый номер, значение аргумента  $x$  и значение функции  $y$ .

Определить максимальное и минимальное значения функции  $y$ .

1.  $Y(x) = \frac{2 \sin x}{(1-x)^2},$   $a = -\pi; b = \pi; h = 0.4.$
2.  $Y(x) = -\ln \left| 2 \sin \frac{x}{2} \right|,$   $a = 0.7; b = 1.8; h = 0.1.$
3.  $Y(x) = \frac{x \sin(\pi/4)}{1 - 2x \cos(\pi/4) + x^2},$   $a = -0.5; b = 2.5; h = 0.2.$
4.  $Y(x) = (1 - x^2/4) \cos x - \frac{x}{2} \sin x,$   $a = -0.9; b = 2.7; h = 0.3.$
5.  $Y(x) = \frac{x \cos \frac{\pi}{4} - x^2}{1 - 2x \cos \frac{\pi}{4} + x^2},$   $a = -2; b = 0.8; h = 0.2.$
6.  $Y(x) = (x^2/4 + x/2 - 3) \cdot e^{x/2},$   $a = -1.9; b = 2.7; h = 0.3.$
7.  $Y(x) = x^2 \sqrt{15 + 10 \sin(x + \pi)},$   $a = -0.4\pi; b = 0.4\pi; h = 0.5.$
8.  $Y(x) = e^x \sin x,$   $a = -0.3\pi; b = 1.3\pi; h = \pi/10.$
9.  $Y(x) = x^2 \cos x \sin x,$   $a = -\pi/2; b = \pi/2; h = \pi/10.$
10.  $Y(x) = x \log(|x - 0,6|),$   $a = -3; b = 3; h = 0.5.$
11.  $Y(x) = \frac{x}{2} \cos x - \sin x,$   $a = -\pi; b = \pi; h = \pi/6.$

12.  $Y(x) = e^x + \sqrt{1 + e^{2x}} - 2,$   $a = -0.9; b = 1, h = 0.3.$
13.  $Y(x) = (1 - x^2 / 4) \cos x - \frac{x}{2} \sin x,$   $a = -0.9; b = 2.7; h = 0.3.$
14.  $Y(x) = \frac{1}{x^2 - x + 1},$   $a = -0.1; b = 2; h = 0.1.$
15.  $Y(x) = \frac{\sin x \cos x}{\sqrt{x}},$   $a = \pi; b = 2\pi; h = \pi/15.$

### 3.3.2. Второй уровень сложности

Для каждого  $x$ , изменяющегося от  $a$  до  $b$  с шагом  $h$ , найти значения функции  $Y(x)$ , суммы  $S(x)$  и  $|Y(x) - S(x)|$  и вывести в виде таблицы. Значения  $a, b, h$  и  $n$  вводятся с клавиатуры. Так как значение  $S(x)$  является рядом разложения функции  $Y(x)$ , при правильном решении значения  $S$  и  $Y$  для заданного аргумента  $x$  (для тестовых значений исходных данных) должны совпадать в целой части и в первых двух-четырех позициях после десятичной точки.

Работу программы проверить для  $a = 0,1; b = 1,0; h = 0,1$ ; значение параметра  $n$  выбрать так, чтобы  $|Y(x) - S(x)| < 0.001$  (0.0001, 0.00001).

1.  $S(x) = \sum_{k=0}^n (-1)^k \frac{x^{2k+1}}{(2k+1)!},$   $Y(x) = \sin(x).$
2.  $S(x) = \sum_{k=1}^n (-1)^{k+1} \frac{x^{2k}}{2k(2k-1)},$   $Y(x) = x \cdot \arctg(x) - \ln \sqrt{1+x^2}.$
3.  $S(x) = \sum_{k=0}^n \frac{\cos(k\pi/4)}{k!} x^k,$   $Y(x) = e^{x \cos \frac{\pi}{4}} \cos(x \sin(\pi/4)).$
4.  $S(x) = \sum_{k=0}^n (-1)^k \frac{x^{2k}}{(2k)!},$   $Y(x) = \cos(x).$
5.  $S(x) = \sum_{k=0}^n \frac{\cos(kx)}{k!},$   $Y(x) = e^{\cos x} \cos(\sin(x)).$
6.  $S(x) = \sum_{k=0}^n \frac{2k+1}{k!} x^{2k},$   $Y(x) = (1+2x^2)e^{x^2}.$
7.  $S(x) = \sum_{k=1}^n \frac{x^k \cos(k\pi/3)}{k},$   $Y(x) = -\frac{1}{2} \ln \left( 1 - 2x \cos \frac{\pi}{3} + x^2 \right).$
8.  $S(x) = \sum_{k=0}^n \frac{(2x)^k}{k!},$   $Y(x) = e^{2x}.$

$$9. S(x) = \sum_{k=1}^n (-1)^{k+1} \frac{x^{2k+1}}{4k^2 - 1},$$

$$Y(x) = \frac{1+x^2}{2} \operatorname{arctg}(x) - x/2.$$

$$10. S(x) = \sum_{k=0}^n \frac{x^{2k}}{(2k)!},$$

$$Y(x) = \frac{e^x + e^{-x}}{2}.$$

$$11. S(x) = \sum_{k=0}^n \frac{k^2 + 1}{k!} (x/2)^k,$$

$$Y(x) = (x^2/4 + x/2 + 1)e^{x/2}.$$

$$12. S(x) = \sum_{k=0}^n (-1)^k \frac{2k^2 + 1}{(2k)!} x^{2k},$$

$$Y(x) = (1 - x^2/2) \cos(x) - \frac{x}{2} \sin(x).$$

$$13. S(x) = \sum_{k=1}^n (-1)^k \frac{(2x)^{2k}}{(2k)!},$$

$$Y(x) = 2(\cos^2 x - 1).$$

$$14. S(x) = \sum_{k=0}^n \frac{x^{2k+1}}{(2k+1)!},$$

$$Y(x) = \frac{e^x - e^{-x}}{2}.$$

$$15. S(x) = \sum_{k=1}^n (-1)^{k+1} \frac{x^{2k}}{2k(2k-1)},$$

$$Y(x) = -\ln \sqrt{1+x^2} + x \operatorname{arctg}(x).$$

### 3.3.3. Третий уровень сложности

Составить программу по заданию из п. 3.3.2, в которой нахождение суммы  $S(x)$  и функции  $Y(x)$  организовать в виде отдельных функций, причем расчет функции выполнять не по заданному значению  $n$ , а до тех пор пока разница очередного значения суммы не будет отличаться от значения функции на некоторую величину (погрешность)  $\varepsilon$ , равную, например, 0.001 (0.0001), т. е. до тех пор пока  $|S(x) - Y(x)| \geq \varepsilon$ . Определить количество шагов вычисления суммы, при которых был достигнут указанный результат.

## Тема №4. Обработка одномерных массивов

**Цель работы:** изучить составной тип данных – массив, написать и отладить программу с использованием одномерных массивов.

### 4.1. Общие теоретические сведения

Массив – составной объект, представляющий конечную последовательность данных заданного типа. Каждый элемент массива определяется его именем и целочисленным значением индекса (номера), по которому производится доступ к этому элементу.

**Внимание! Индексы массивов в языке C/C++ начинаются с 0.**

В программе одномерный массив декларируется следующим образом:

*тип Имя\_массива [Размер];*

где **целочисленное** значение *Размер* задает максимальное количество элементов в массиве. *Размер* может задаваться константой или константным выражением. Для использования массивов переменного размера существует отдельный механизм – динамическое выделение памяти.

Примеры декларации (описания) одномерных массивов:

1) `const Nmax = 10;` – задание максимального значения;

`typedef double mas1[Nmax*2];` – описание пользовательского типа;

`mas1 a;` – декларация массива *a* типа *mas1*;

2) `int a[5];` – массив *a* из 5 целых чисел, в котором первый элемент *a*[0], второй – *a*[1], ..., пятый (последний) – *a*[4];

3) `double b[10] = {1.5, 2.5, 3.75};` – массив *b* из 10 действительных чисел с инициализацией: *b*[0] = 1.5, *b*[1] = 2.5, *b*[2] = 3.75, остальные *b*[3], ..., *b*[9] равны 0.

В языке C/C++ не проверяется выход индекса за пределы массива. Корректность использования индексов должен контролировать программист.

Элементы массивов могут использоваться в выражениях так же, как и обычные переменные, например:

`f = 2*a[3] + a[Ss[i] + 1]*3;`

`a[n] = 1 + sqrt(fabs(a[n - 1]));`

### 4.2. Создание оконного приложения в среде Builder

#### **Компонента StringGrid**

При работе с массивами ввод и вывод значений обычно организуется с использованием компоненты *StringGrid*, предназначенной для отображения информации в виде двумерной таблицы, каждая ячейка которой представляет собой окно однострочного редактора (аналогично окну *Edit*). Доступ к содержимому ячеек осуществляется с помощью элемента *Cells[ACol][ARow]* типа *AnsiString*, где целочисленные значения *ACol*, *ARow* указывают позицию элемента.



**Внимание! Первый индекс *ACol* определяет номер столбца, а второй *ARow* – номер строки** в отличие от индексов массива. Нумерация строк и столбцов начинается с 0, как и в массивах.

В инспекторе объектов значения *ColCount* и *RowCount* устанавливают начальные значения количества столбцов и строк в таблице, а *FixedCols* и *FixedRows* задают количество столбцов и строк фиксированной зоны. Фиксированная зона выделена другим цветом и обычно используется для надписей.

### 4.3. Пример выполнения задания

Удалить из массива *A* размером *n*, состоящего из целых чисел (положительных и отрицательных), все отрицательные числа. Новый массив не создавать. Для инициализации массива использовать функцию генератора случайных равномерно распределенных целых чисел.

Значение *n* вводить из *Edit*, значения массива *A* – из компоненты *StringGrid1*. Результат вывести в компоненту *StringGrid2*.

Панель диалога и результаты выполнения программы могут иметь вид, приведенный на рис. 4.1.

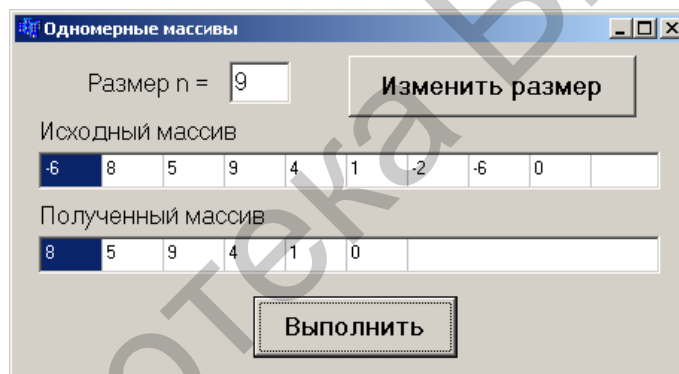



Рис. 4.1

### Настройка компоненты *StringGrid*

На закладке *Additional* выберите пиктограмму , установите компоненты *StringGrid1* и *StringGrid2* и отрегулируйте их размеры. В инспекторе объектов для обеих компонент установите значения *ColCount* равными, например, 5, *RowCount* равными 1, т. е. по пять столбцов и одной строке, а значения *FixedCols* и *FixedRows* равными 0.

По умолчанию в компоненту *StringGrid* ввод данных разрешен только программно. Для разрешения ввода данных с клавиатуры необходимо в свойстве **Options** строку **goEditing** для *StringGrid1* установить в положение **true**.

Приведем примеры участков кода, реализующих решение поставленной задачи для массива *int* размером 10 элементов в оконном приложении.

1. В тексте функции *FormCreate*:

```
int i, n = 5;  
randomize();           // Изменение начального адреса для random()
```

```

Edit1->Text = IntToStr(n);
StringGrid1->ColCount = n;
for(i=0; i<n; i++)      // Заполнение StringGrid1 случайными числами
    StringGrid1->Cells[i][0] = IntToStr(random(21) - 10);

```

2. В тексте функции-обработчика кнопки «*Изменить размер*»:

```

n = StrToInt(Edit1->Text);
if(n < 1 || n > 10) {      // Проверка корректности размера
    ShowMessage("Ошибка!");
    n = 10;
    Edit1->Text = "10";
}
StringGrid1->ColCount = n;
for(i=0; i<n; i++)
    StringGrid1->Cells[i][0] = IntToStr(random(21) - 10);

```

3. В тексте функции-обработчика кнопки «*Выполнить*»:

```

int i, kol = 0, a[10];      // Декларация некоторых переменных
for(i=0; i<n; i++)          // Заполнение массива
    a[i] = StrToInt(StringGrid1->Cells[i][0]);
for(i=0; i<n; i++)          // Удаление отрицательных элементов
    if(a[i]>=0) a[kol++] = a[i];
StringGrid2->ColCount = kol; // Вывод результата в StringGrid2
for(i=0; i<kol; i++)
    StringGrid2->Cells[i][0] = IntToStr(a[i]);

```

#### 4.4. Индивидуальные задания

##### 4.4.1. Первый уровень сложности

Для одномерного массива, состоящего из  $n$  вводимых с клавиатуры элементов ( $n$  – не больше 20), вычислить указанное в задании значение.

1. Среднее арифметическое положительных элементов массива.
2. Произведение элементов, расположенных после наибольшего элемента.
3. Сумму элементов массива, расположенных до последнего положительного элемента.
4. Сумму элементов, расположенных до наименьшего элемента.
5. Среднее арифметическое отрицательных элементов массива.
6. Сумму дробных частей элементов массива.
7. Сумму элементов массива, расположенных до минимального элемента.
8. Сумму целых частей элементов массива, расположенных после последнего отрицательного элемента.
9. Сумму элементов массива, расположенных после последнего элемента, равного нулю.

10. Сумму модулей элементов массива, расположенных после минимального по модулю элемента.
11. Сумму элементов массива, расположенных после минимального элемента.
12. Сумму элементов массива, расположенных после первого положительного элемента.
13. Сумму модулей элементов массива, расположенных после первого отрицательного элемента.
14. Сумму модулей элементов массива, расположенных после первого элемента, равного нулю.
15. Сумму положительных элементов массива, расположенных до максимального элемента.

#### **4.4.2. Второй уровень сложности**

Для одномерного массива, состоящего из  $n$  значений ( $n$  – не больше 20), решить поставленную задачу. Предусмотреть ввод значений массива как с клавиатуры, так и с помощью заполнения случайными числами. Выводить сообщения в случае невозможности выполнения поставленного задания.

1. Упорядочить по возрастанию элементы массива, расположенные между максимальным и минимальным элементами.
2. Вычислить сумму элементов массива, расположенных между первым и последним нулевыми элементами.
3. Вычислить сумму элементов массива, расположенных между первым и последним положительными элементами.
4. Удалить из введенного одномерного массива все отрицательные элементы, не вводя новый массив.
5. Вычислить произведение элементов массива, расположенных между первым и вторым нулевыми элементами.
6. Вычислить сумму дробных частей элементов массива, расположенных между первым и вторым отрицательными элементами.
7. Вычислить сумму элементов, расположенных между первым положительным и первым отрицательным элементами массива.
8. Вычислить произведение четных элементов массива, расположенных между первым и вторым нулевыми элементами.
9. Вычислить сумму модулей элементов массива, расположенных между минимальным и последним отрицательным элементами.
10. Удалить из введенного одномерного массива все положительные элементы, не вводя новый массив.
11. Вычислить сумму элементов введенного одномерного массива, расположенных между максимальным и последним нулевым элементами.
12. Вычислить сумму модулей элементов, стоящих на четных позициях в массиве и расположенных после первого отрицательного элемента.
13. Вычислить произведение элементов массива, расположенных после последнего нулевого элемента и не превышающих среднее арифметическое.

14. Вычислить сумму модулей элементов введенного одномерного массива, расположенных после наименьшего по модулю элемента (последнего из наименьших).

15. Удалить из введенного с клавиатуры одномерного массива все четные элементы, не вводя новый массив.

#### **4.4.3. Третий уровень сложности**

Для решения поставленной задачи использовать динамический массив. Решение задачи организовать в виде отдельной функции. В основной функции организовать ввод исходных данных, обращение к созданной функции и вывод результатов.

При вводе исходных данных использовать обработку ввода нечисловых значений.

1. Ввести целое число  $N$ . Выделить из этого числа цифры, кратные 3, и записать их в одномерный массив.

2. Для заданного целого числа  $N$  определить цифру  $a$ , наиболее часто встречающуюся в этом числе. Сформировать одномерный массив из 5 элементов:  $a, a^2, a^3, a^4, a^5$ .

3. Элементы заданного массива  $X$  циклически сдвинуть на  $K$  позиций влево.

4. 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.

5. Удалить из массива все повторяющиеся элементы.

6. 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.

7. Заданы два массива по  $N$  целых чисел. Найти наименьшее среди чисел первого массива, которое не входит во второй массив.

8. Дан массив символов английского алфавита. Вывести на экран в алфавитном порядке символы, встречающиеся в массиве один раз.

9. Элементы заданного массива  $X$  циклически сдвинуть на  $K$  позиций вправо.

10. Удалить из массива каждый третий элемент.

11. Дано целое число  $N$ . Составить массив, содержащий квадраты разрядов данного числа.

12. Посчитать количество различных элементов массива.

13. Расположить в обратном порядке элементы массива, находящиеся между первым и последним нулевым элементами.

14. Определить наименьшее число, входящее в массив наибольшее количество раз.

15. Удалить из массива все элементы, равные значению, введенному пользователем с клавиатуры.

## Тема №5. Обработка двумерных динамических массивов

**Цель работы:** изучить понятие «указатель», правила создания и приемы обработки динамических массивов на примере двумерного массива.

### 5.1. Краткие теоретические сведения

#### Особенности применения указателей

Обращение к объектам любого типа в языке С может проводиться по имени, как мы до сих пор делали, или по **указателю** (косвенная адресация).

Указатель – это переменная, которая содержит адрес некоторого объекта в памяти компьютера, например адрес другой переменной. Через указатель, установленный на переменную, можно обращаться к участку оперативной памяти (ОП), отведенной компилятором под ее значение.

Указатель объявляется следующим образом:

*тип \*Имя\_указателя;*

Перед использованием указатель должен быть инициализирован либо конкретным адресом, либо значением *NULL* (0) – отсутствие указателя.

С указателями связаны две унарные операции: *&* и *\**. Операция *&* означает «взять адрес», а операция разадресации *\** – «значение, расположенное по адресу», например:

```
int x, *y;           // x – переменная типа int, y – указатель на тип int
y = &x;              // y – адрес переменной x
*y = 1;              // по адресу y записать 1, в результате x = 1
```

При работе с указателями можно использовать операции сложения, вычитания (с целочисленными величинами) и сравнения, причем выполняются они в единицах того типа, на который установлен указатель.

Операции сложения (сдвиг вправо), вычитания (сдвиг влево) и сравнения (больше/меньше) имеют смысл только для последовательно расположенных данных – массивов. Операции сравнения «==» (равно) и «!=» (не равно) имеют смысл для любых указателей, т. е. если два указателя равны между собой, то они указывают на одну и ту же переменную.

#### Связь указателей с массивами

Указатели и массивы тесно связаны между собой. Идентификатор массива является указателем на его первый элемент, т. е. для массива *int a[10]*, выражения *a* и *&a[0]* имеют одинаковые значения, т. к. адрес первого (с индексом 0) элемента массива – это адрес начала размещения его элементов в ОП. Аналогично, выражения *\*a* и *a[0]* имеют одинаковые значения, т. к. значение, находящееся по адресу *a*, – это и есть значение первого элемента массива *a[0]*.

Пусть объявлены массив *a* из 10 элементов и указатель *p*:

```
double a[10], *p;
```

если  $p = a$ ; (установить указатель  $p$  на начало массива  $a$ ), то следующие обращения:  $a[i]$ ,  $*(a + i)$ ,  $p[i]$  и  $*(p + i)$  – эквивалентны, т. е. для любых указателей можно использовать две эквивалентные формы доступа к элементам массива:  $a[i]$  и  $*(a + i)$ .

### Декларация многомерных массивов:

*тип Имя\_массива [Размер1] [Размер2]... [РазмерN];*

Быстрее изменяется последний индекс, т. к. многомерные массивы размещаются в ОП в последовательности столбцов, например массив целого типа, состоящий из двух строк и трех столбцов (с инициализацией начальных значений):

```
int a[2][3] = { {0, 1, 2}, {3, 4, 5} };
```

в ОП будет размещен следующим образом:

```
a[0][0] = 0, a[0][1] = 1, a[0][2] = 2, a[1][0] = 3, a[1][1] = 4, a[1][2] = 5.
```

Если в списке инициализаторов данных не хватает, то соответствующим элементам присваивается значение 0.

### Указатели на указатели

Связь указателей и массивов с одним измерением справедлива и для массивов с большим числом измерений.

Если рассматривать предыдущий массив (`int a[2][3];`) как массив двух массивов размером по три элемента каждый, то обращению к элементу  $a[i][j]$  соответствует эквивалентное выражение  $*(*(a + i) + j)$ , а объявление этого массива с использованием указателей будет иметь вид

```
int **a;
```

Таким образом, имя двумерного массива – это указатель на указатель.

### Динамическое размещение данных

Для создания массивов с переменными размерами используется динамическое размещение данных, декларируемых указателями.

Для работы с динамической памятью могут использоваться стандартные функции библиотеки *alloc.h* (**malloc**, **calloc**, **free**), но удобнее использовать операции языка C++ *new* и *delete*.

Операция **new** возвращает адрес ОП, отведенной под динамически размещенный объект, при ошибке – значение *NULL*, а операция **delete** освобождает память.

Минимальный набор действий, необходимых для динамического размещения одномерного массива действительных чисел размером  $n$ :

```
double *a;
cin >> n;           // Ввод нужного размера
a = new double[n];   // Захват памяти для n элементов типа double
...
delete []a;          // Освобождение памяти
```

Минимальный набор действий, необходимых для динамического размещения двухмерного массива действительных чисел размером  $n \times m$ :

```
double **a;
cin >> n >> m;           // Ввод нужных размеров
a = new double *[n];       // Захват памяти под массив указателей
for(i=0; i<n; i++)
    a[i] = new double [m]; // и под массивы элементов
for(i=0; i<n; i++) ...     // Освобождение памяти
    delete []a[i];
delete []a;
```

Для современных компиляторов достаточно только `delete []a;`

## 5.2. Пример выполнения задания

Найти сумму положительных элементов *int* массива *a* размером  $n \times m$ .

Приведем примеры участков кода, реализующих решение поставленной задачи в оконном приложении.

1. В тексте функции *FormCreate*:

```
int i, j, n = 5, m = 5;           // п. 1
Edit1->Text = IntToStr(n);        Edit2->Text = IntToStr(m); // п. 2
StringGrid1->RowCount = n;        StringGrid1->ColCount = m; // п. 3
for(i=0; i<n; i++)                // Заполнение StringGrid1 случайными числами
    for(j=0; j<m; j++)
        StringGrid1->Cells[j][i] = IntToStr(random(21) - 10);
```

2. В тексте функции-обработчика кнопки «Изменить размеры» вместо строк п. 1 и 2 могут быть следующие:

```
int i, j, n, m;                   // п. 1
n = StrToInt(Edit1->Text);        m = StrToInt(Edit2->Text); // п. 2
```

Остальное аналогично функции *FormCreate*.

3. В тексте функции-обработчика кнопки «Выполнить»:

```
int i, j, sum, **a, n, m;         // Декларация переменных
...
for(i=0; i<n; i++)                //Заполнение массива a из StringGrid1
    for(j=0; j<m; j++)
        a[i][j] = StrToInt(StringGrid1->Cells[j][i]);
sum = 0;                          // Поиск суммы
for ( i=0; i<n; i++)
    for ( j=0; j<m; j++)
        if(a[i][j] > 0)
            sum += a[i][j];
```

### **5.3. Индивидуальные задания**

#### **5.3.1. Первый уровень сложности**

Написать программу по обработке двумерного динамического массива. Размеры массива  $n, m$  и значения его элементов вводятся с клавиатуры.

1. Определить количество строк, не содержащих ни одного нулевого элемента.
2. Определить количество столбцов, не содержащих ни одного нулевого элемента.
3. Определить количество столбцов, содержащих хотя бы один нулевой элемент.
4. Определить произведение элементов в тех строках, которые не содержат отрицательных элементов.
5. Определить сумму элементов в тех столбцах, которые не содержат отрицательных элементов.
6. Определить сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.
7. Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.
8. Найти сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.
9. Найти сумму модулей элементов, расположенных ниже главной диагонали.
10. Найти сумму модулей элементов, расположенных выше главной диагонали.
11. Найти количество строк, среднее арифметическое элементов которых меньше введенной с клавиатуры величины.
12. Найти номер первой из строк, содержащих хотя бы один положительный элемент.
13. Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент.
14. Найти номер первого из столбцов, не содержащих ни одного отрицательного элемента.
15. Найти номер первой из строк, не содержащих ни одного положительного элемента.

#### **5.3.2. Второй уровень сложности**

1. Из матрицы размером  $n \times m$  получить вектор  $B$ , присвоив его  $k$ -му элементу значение 0, если все элементы  $k$ -го столбца матрицы нулевые, иначе 1.
2. Из матрицы размером  $n \times m$  получить вектор  $B$ , присвоив его  $k$ -му элементу значение 1, если элементы  $k$ -й строки матрицы упорядочены по убыванию, иначе 0.



3. Из матрицы размером  $n \times m$  получить вектор  $B$ , присвоив его  $k$ -му элементу значение 1, если  $k$ -я строка матрицы симметрична, иначе значение 0.
4. Задана матрица размером  $n \times m$ . Определить количество «особых» элементов матрицы, считая элемент «особым», если он больше суммы остальных элементов своего столбца.
5. Задана матрица размером  $n \times m$ . Определить количество элементов матрицы, у которых слева находится элемент больше его, а справа – меньше.
6. Задана матрица размером  $n \times m$ . Определить количество различных значений матрицы, т. е. повторяющиеся элементы считать один раз.
7. В матрице размером  $n \times m$  упорядочить строки по возрастанию их первых элементов.
8. В матрице размером  $n \times m$  упорядочить строки по возрастанию суммы их элементов.
9. В матрице размером  $n \times m$  упорядочить строки по возрастанию их наибольших элементов.
10. Определить, является ли квадратная матрица симметричной относительно побочной диагонали.
11. Задана матрица размером  $n \times m$ . Определить количество элементов матрицы, у которых слева находится элемент меньше его, а справа – больше.
12. В матрице размером  $n$  найти сумму элементов, лежащих ниже главной диагонали, и произведение элементов, лежащих выше побочной диагонали.
13. В квадратной матрице найти максимальный среди элементов, лежащих ниже побочной 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 размером  $n$  получить матрицу размером  $n - 1$  путем удаления строки и столбца, на пересечении которых расположен элемент с наибольшим по модулю значением.

### 5.3.3. Третий уровень сложности

Для динамического двухмерного массива решить поставленную задачу, алгоритм которой реализовать в виде отдельной функции. При вводе исходных данных выполнить проверку ввода нечисловых значений.

1. Квадратную вещественную матрицу  $A$  размером  $n$  возвести в  $k$ -ю степень, т. е. вычислить  $A^1 = A$ ,  $A^2 = A \cdot A$ ,  $A^3 = A^2 \cdot A$  и т. д.
2. Дана вещественная матрица размером  $n \times m$ . Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент (один из них) оказался в верхнем левом углу.
3. Дана вещественная матрица размером  $n \times m$ . Упорядочить ее строки по возрастанию наибольших элементов в строках матрицы.

4. Задан массив размером  $n \times n$ , состоящий из нулей и единиц. Повернуть элементы массива на  $90^\circ$  по часовой стрелке.

5. Элемент матрицы назовем «особым», если он наименьший в своей строке и наибольший (одновременно) в своем столбце (или наоборот, наибольший в своей строке и наименьший в своем столбце). Для заданной целочисленной матрицы размером  $n \times m$  вывести индексы всех ее «особых» элементов.

6. Дана вещественная матрица размером  $n$ , все элементы которой различны. Найти скалярное произведение строки, в которой находится наибольший элемент матрицы, на столбец с наименьшим элементом.

7. Определить, является ли заданная целочисленная квадратная матрица размером  $n$  ортонормированной, т. е. такой, в которой скалярное произведение каждой пары различных строк равно 0, а скалярное произведение каждой строки на себя равно 1.

8. Определить, является ли заданная матрица  $n$ -го порядка магическим квадратом, т. е. такой, в которой сумма элементов во всех строках и столбцах одинакова.

9. Дана целочисленная матрица размером  $n$ . Найти сумму наименьших элементов ее нечетных строк и наибольших элементов ее четных строк.

10. Дана действительная квадратная матрица порядка  $n$ . Рассмотрим те элементы, которые расположены в строках, начинающихся с отрицательного элемента. Найти сумму тех из них, которые расположены соответственно ниже, выше и на главной диагонали матрицы.

11. Дана вещественная квадратная матрица порядка  $n$ . Получить целочисленную квадратную матрицу, в которой элемент равен 1, если соответствующий ему элемент исходной матрицы больше элемента, расположенного на главной диагонали, и равен 0 в противном случае.

12. В действительной квадратной матрице порядка  $n$  найти сумму и произведение элементов, расположенных в заштрихованной области (рис. 5.1, а).

13. В действительной квадратной матрице порядка  $n$  найти сумму и произведение элементов, расположенных в заштрихованной области (рис. 5.1, б).

14. В действительной квадратной матрице порядка  $n$  найти сумму и произведение элементов, расположенных в заштрихованной области (рис. 5.1, в).

15. В действительной квадратной матрице порядка  $n$  найти наименьшее и наибольшее из значений элементов, расположенных в заштрихованной области (см. рис. 5.1, а).

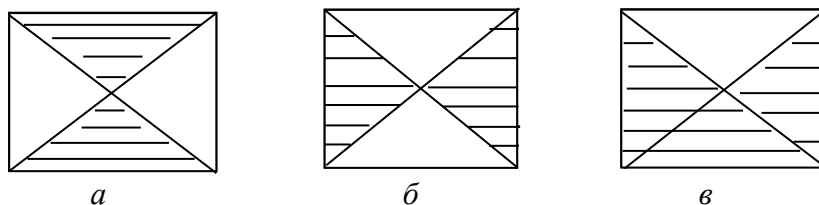


Рис. 5.1

## Тема №6. Использование строк

**Цель работы:** изучить особенности строковых данных, правила работы с компонентами *ListBox* и *ComboBox*, написать и отладить программу работы со строками.

### 6.1. Общие теоретические сведения

#### **Строки как одномерные массивы символов**

В языке C/C++ строка – это одномерный массив символов, заканчивающийся нулевым байтом, каждый бит которого равен нулю, при этом для нулевого байта определена константа `'\0'` (признак окончания строки или нуль-терминатор).

Для строки, состоящей из 80 символов, в описании массива необходимо указать размер 81, т. к. последний байт отводится под нуль-терминатор.

Напомним, что строковая константа – это набор символов, заключенных в кавычки, например “Лабораторная работа по строкам” (`'\0'` не указывается).

Строки можно инициализировать при декларации, например:

```
char S1[10] = "123456789";
```

Операции над строками рекомендуется выполнять с использованием стандартных библиотечных функций.

Рассмотрим наиболее часто используемые функции библиотеки *string.h*:

1) **strcpy** (*S1*, *S2*) – копирует содержимое строки *S2* в строку *S1*;

2) **strncpy** (*S1*, *S2*, *n*) – копирует *n* символов из строки *S2* в строку *S1*; при выполнении функции нуль-терминатор необходимо добавлять в конец формируемой строки вручную;

3) **strcat** (*S1*, *S2*) – присоединяет строку *S2* к строке *S1* и помещает ее в массив, где находилась строка *S1*, при этом строка *S2* не изменяется; нулевой байт, который завершал строку *S1*, заменяется первым символом строки *S2*;

4) **strncat** (*S1*, *S2*, *n*) – присоединяет справа к *S1* *n* символов из строки *S2*;

5) **strcmp** (*S1*, *S2*) – сравнивает строки *S1* и *S2* и возвращает значение 0, если строки равны, т. е. содержат одно и то же число одинаковых символов; значение < 0, если *S1* < *S2*; и значение > 0, если *S1* > *S2*;

6) **strlen** (*S*) – возвращает длину строки, т. е. количество символов, начиная с первого (*S*[0]) и до нуля-терминатора, который не учитывается;

7) **strstr** (*S1*, *S2*) – указывает первое появление подстроки *S2* в строке *S1*;

8) преобразование строки *S* в число (библиотека *stdlib.h*): целое – (*int*) **atoi** (*S*); длинное целое – (*long*) **atoll** (*S*); действительное – (*double*) **atof** (*S*); при возникновении ошибки данные функции возвращают значение 0;

9) преобразование числа *V* в строку *S* (библиотека *stdlib.h*): целое – **itoa** (*int V*, *char S*, *int k*); длинное целое – **ltoa** (*long V*, *char S*, *int k*); значение параметра *k* определяет выбор системы счисления для преобразования и находится в диапазоне  $2 \leq k \leq 36$ , для отрицательных чисел  $k = 10$ .

## 6.2. Создание оконного приложения в среде Builder

### *Tun AnsiString*

В компонентах и функциях C++ *Builder* основным строковым типом является тип *AnsiString* (просто *String*). Отдельные символы, входящие в строку типа *String*, имеют тип *char*. К ним можно обращаться так же, как к элементам массива *char*, но нумерация начинается с *единицы*.

Многие другие операции над строками осуществляются с помощью методов класса *String*. Метод – это тип функций, вызываемых особым образом: для обращения к методу надо после имени переменной типа *String* поставить точку, а затем имя метода и круглые скобки (в которых при необходимости указываются параметры метода).

Часто используемые методы этого класса:

***str.Length()*** – определяет длину строки *str*;

***str.c\_str()*** – преобразует строку *str* в массив символов;

***str.Insert(String s, int i)*** – вставляет строку *s* в строку *str*, начиная с *i*-й позиции;

***str.Delete(int i, int n)*** – удаляет *n* символов из строки *str*, начиная с *i*-й позиции. Если в исходной строке, начиная с *i*-й позиции, содержится меньше *n* символов, то удаляет имеющиеся символы с *i*-й позиции;

***str.Pos(s)*** – находит в строке *str* номер позиции, с которой начинается подстрока *s*, если подстрока не найдена, результат 0;

***str.SubString(int i, int n)*** – копирует из строки *str* подстроку длиной *n* символов, начиная с *i*-й позиции. Исходная строка не изменяется. Если в исходной строке, начиная с *i*-й позиции, содержится меньше *n* символов, то выделяет столько символов, сколько есть.

### *Компонента ListBox*

Данная компонента представляет собой список, элементы которого выбирают при помощи клавиатуры или мыши. Список элементов задается свойством *Items*, методы *Add*, *Delete* и *Insert* которого используются для добавления, удаления и вставки строк соответственно. Объект *Items* хранит строки списка. Для определения номера выделенного элемента используется свойство *ItemIndex*.

### *Компонента ComboBox*

Список *ComboBox* – комбинация списка *ListBox* и редактора текста *Edit*, поэтому практически все свойства аналогичны. Для работы с окном редактирования используется свойство *Text*, как в *Edit*, а для работы со списком выбора – свойство *Items*, как в *ListBox*. Существует пять модификаций компоненты, определяемых ее свойством *Style*, выбрав в котором модификацию *csSimple*, раскрываем список, потянув его за нижнюю кромку (захватив ее мышью).

### Компонента-кнопка *BitBtn*

Компонента *BitBtn* расположена на странице *Additonal* и представляет собой разновидность стандартной кнопки *Button*. Ее отличие – наличие свойства ***Kind***, которое задает одну из 11 стандартных разновидностей кнопок, нажатие любой из них, кроме *bkCustom* и *bkHelp*, закрывает окно и возвращает в программу результат *mr\*\*\** (например, *bkOk* – *mrOk*). Кнопка *bkClose* закрывает главное окно и завершает работу программы.

### Обработка событий

Обо всех происходящих в системе событиях, таких как создание формы, нажатие кнопки мыши или клавиши клавиатуры и т. д., ядро *Windows* информирует окна путем посылки соответствующих сообщений. Среда *Builder* позволяет принимать и обрабатывать большинство таких сообщений. Каждая компонента содержит обработчики сообщений на странице *Events* инспектора объектов.

Для создания обработчика события необходимо выделить нужную компоненту, далее на странице *Events* выбрать обработчик и двойным щелчком кнопкой мыши в белом (пустом) окошке в текст программы будет вставлена соответствующая функция. Например, выделив компоненту *Form1* и выбрав обработчик *OnActivate*, будет вставлена функция `...FormActivate(...) { }`.

Каждая компонента имеет свой набор обработчиков событий, однако некоторые из них присущи большинству компонент. Рассмотрим наиболее часто применяемые события:

*OnActivate* – форма получает это событие при активации;

*OnCreate* – возникает при создании формы (*Form*), в обработчике события задаются действия, которые должны происходить в момент создания формы;

*OnKeyPress* – возникает при нажатии клавиши клавиатуры, параметр *Key* типа *WORD* содержит *ASCII*-код нажатой клавиши (*Enter* имеет код 13, *Esc* – 27) и обычно используется, когда необходима реакция на нажатие одной из клавиш;

*OnKeyDown* – возникает при нажатии клавиши клавиатуры, обработчик этого события получает информацию о нажатой клавише и состоянии клавиш *Shift*, *Alt* и *Ctrl*, а также о нажатой кнопке мыши;

*OnKeyUp* – является парным событием для *OnKeyDown* и возникает при отпускании ранее нажатой клавиши;

*OnClick* – возникает при щелчке кнопкой мыши в области компоненты, а *OnDblClick* – при двойном щелчке кнопкой мыши в области компоненты.

### 6.3. Пример выполнения задания

Написать программу подсчета числа слов в строке, содержащей пробелы.

Для ввода строк и работы с ними использовать компоненту *ComboBox*. Ввод строки заканчивать нажатием клавиши *Enter*, для выхода использовать кнопку «*Close*». Панель диалога с результатами программы может иметь вид, представленный на рис. 6.1.

В тексте программы приведем только функции-обработчики:

```
void __fastcall TForm1::FormActivate(...)
{
    Form1->ComboBox1->SetFocus();           // Передача фокуса ComboBox1
}
//-----
void __fastcall TForm1::ComboBox1KeyPress(...)
{
    if (Key == 13) {
        ComboBox1->Items->Add(ComboBox1->Text);
// Строка из окна редактирования заносится в ComboBox1
        ComboBox1->Text = "";               // Очистка окна
    }
}
//----- Обработка нажатия кнопки мыши -----
void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
    int n, i;
    AnsiString st = ComboBox1->Text;         // Запись выбранной строки st
    if (st[1] != ' ') n = 1;                 // Здесь и ниже ' ' – пробел
        else n = 0;
    for(i = 1; i < st.Length(); i++)
        if(st[i] == ' ' && st[i+1] != ' ') n++;
    Edit1->Text = IntToStr(n);
}
```

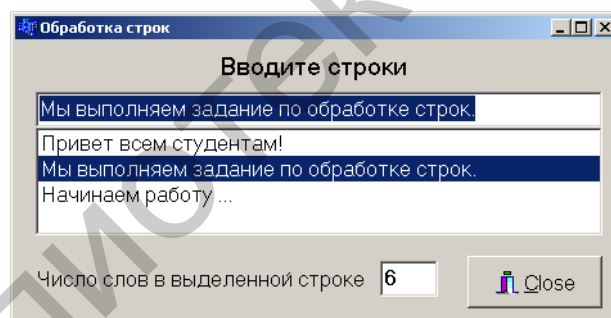


Рис. 6.1

## 6.4. Индивидуальные задания

В оконном приложении исходные данные вводить из компоненты *Edit* в *Listbox* (заканчивать нажатием *Enter*). Скалярный результат выводить в компоненту *Label*. Для выхода из программы использовать кнопку «Close».

### 6.4.1. Первый уровень сложности

1. Ввести целое число  $N$ . Выделить из этого числа цифры, кратные 3, и записать их в строку.

2. Для заданного целого числа  $N$  определить цифру  $a$ , наиболее часто встречающуюся в числе. Сформировать строку из 5 элементов:  $a, a^2, a^3, a^4, a^5$ .

3. Дана строка, содержащая запись числа целого типа. Определить, симметрично ли оно, т. е. одинаковы ли цифры слева и справа (12321). Записать 3 последние цифры в строку.
4. Создать новую строку символов, где каждый символ  $a$  исходной строки дублируется.
5. Вывести все цифры, содержащиеся в строке.
6. Ввести строку символов. Определить длину введенной строки  $L$ , и если длина  $L$  четная, то удаляются 2 первых и 2 последних символа.
7. Ввести строку символов. Определить длину введенной строки  $L$ , и если длина  $L$  нечетная, то удаляется символ, стоящий посередине строки.
8. Ввести строку символов. Заменить в ней каждый второй символ ! на \$.
9. Ввести строку символов. Заменить в ней пробелы на символ \$.
10. Ввести строку символов. Определить длину введенной строки  $L$ , и если длина  $L > 10$ , то удалить все цифры.
11. Ввести строку символов. Определить длину введенной строки  $L$ , и если длина  $L$  кратна 3, то удаляются все цифры, делящиеся на 3.
12. Ввести строку символов. Определить длину введенной строки  $L$ , и если длина  $L$  кратна 5, то подсчитывается количество скобок всех видов.
13. Ввести строку символов. Определить длину введенной строки  $L$ , и если длина  $L$  кратна 4, то первая часть строки меняется местами со второй.
14. Ввести строку символов. Определить длину введенной строки  $L$ , и если длина  $L = 10$ , то удаляются все буквы от A до Z.
15. Ввести строку символов. Определить длину введенной строки  $L$ , и если длина  $L > 15$ , то удаляются все буквы от  $a$  до  $z$ .

#### **6.4.2. Второй уровень сложности**

1. В строке, состоящей из групп нулей и единиц, найти количество групп с пятью цифрами.
2. В строке, состоящей из групп нулей и единиц, найти и вывести на экран самую короткую группу.
3. В строке, состоящей из групп нулей и единиц, подсчитать количество символов в самой длинной группе.
4. В строке, состоящей из групп нулей и единиц, найти и вывести на экран группы с четным количеством символов.
5. В строке, состоящей из групп нулей и единиц, подсчитать количество единиц в группах с нечетным количеством символов.
6. Из строки, состоящей из букв, цифр, запятых, точек, знаков + и – , выделить подстроку, которая соответствует записи целого числа.
7. Из строки, состоящей из букв, цифр, запятых, точек, знаков + и – , выделить подстроку, задающую вещественное число с фиксированной точкой.
8. Из строки, состоящей из букв, цифр, запятых, точек, знаков + и – , выделить подстроку, задающую вещественное число с плавающей точкой.

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

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

11. Дана строка, состоящая из слов на английском языке, разделенных пробелами. Вывести на экран эти слова в порядке алфавита.

12. Дана строка, состоящая из слов, разделенных пробелами. Вывести на экран порядковый номер слова, содержащего  $k$ -ю позицию, если в  $k$ -й позиции пробел, то номер предыдущего слова. Значение  $k$  ввести с клавиатуры.

13. Дана строка, состоящая из слов, разделенных 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. Значение  $k$  вводится с клавиатуры.

14. Дана строка, состоящая из слов, разделенных пробелами. Вывести на экран порядковый номер слова с максимальной длиной и номер позиции строки, с которой оно начинается.

15. Дана строка, состоящая из слов, разделенных пробелами. Вывести на экран порядковый номер слова с минимальной длиной и количество символов в этом слове.

16. В строке символов подсчитать количество скобок различного вида.

#### **6.4.3. Третий уровень сложности**

Для строки, введенной с клавиатуры, решить поставленную задачу, алгоритм которой реализовать в виде отдельной функции. В программе не должны использоваться функции стандартной библиотеки.

1. Ввести с клавиатуры строку символов, после чего подсчитать и вывести на экран ее длину (т. е. число символов, входящих в строку), а также подсчитать и вывести число вхождений буквы «а» в строку.

2. Ввести с клавиатуры строку символов, после чего подсчитать и вывести на экран число пробелов, а также число цифр, содержащихся в ней.

3. Ввести с клавиатуры две строки символов. Дописать вторую строку в конец первой, после чего повторить эту операцию еще раз. Далее вывести первую строку на экран.

4. Ввести с клавиатуры две строки символов, после чего выполнить посимвольное их сравнение и вывести на экран результат сравнения в виде числа (результат получается вычитанием первых несовпадающих кодов символов из первой и второй строк).

5. Ввести с клавиатуры строку символов, после чего подсчитать и вывести на экран число слов в ней, а затем и каждое отдельное слово.

6. Ввести с клавиатуры строку символов, после чего вывести на экран ее подстроку, включающую в себя символы с 5-го по 15-й, а также подстроку, начиная с символа после последнего пробела и до конца строки.



7. Ввести с клавиатуры строку символов, после чего каждый ее третий символ заменить символом процент (%), а каждый четвертый – символом амперсанд (&). Те символы, которые должны заменяться одновременно и на процент и на амперсанд, заменить в итоге символом решетка (#). Далее вывести результирующую строку на экран.

8. Ввести с клавиатуры строку символов, после чего вывести ее на экран вертикально (по одной букве в каждой строке), а затем по диагонали (также по одной букве в каждой строке, но еще и с линейно возрастающим смещением вправо).

9. Ввести с клавиатуры строку символов, после чего записать ее содержимое в другую строку сначала в обратном порядке, а затем в прямом порядке. Далее вывести вторую строку на экран.

10. Ввести с клавиатуры две строки символов, после чего вставить вторую строку в середину первой. Далее вывести первую строку на экран.

11. Ввести с клавиатуры строку символов, после чего заменить все буквы «a» (латиница) символом звездочка (\*), а все цифры – символом решетка (#). Вывести результирующую строку на экран.

12. Ввести с клавиатуры строку символов, после чего заменить все пробелы символом подчеркивания, а каждую первую и каждую последнюю буквы каждого слова – символами < и > соответственно. Вывести результирующую строку на экран.

13. Ввести с клавиатуры строку символов, представляющую собой некоторое предложение. Обрезать предложение так, чтобы оно содержало только второе, третье и четвертое слова. Результирующую строку вывести на экран.

14. Ввести с клавиатуры строку символов. Заменить первую букву в каждом втором слове на вопросительный знак и вывести результирующую строку на экран.

15. Ввести с клавиатуры строку символов. Переписать из нее первое, третье и пятое слова (через пробел) в другую строку, после чего последнюю вывести на экран.

## Тема №7. Обработка структур с использованием файлов

**Цель работы:** изучить правила создания и обработки данных структурного типа с использованием файлов, правила работы с компонентами *OpenDialog* и *SaveDialog*, написать и отладить программу по созданию файлов.

### 7.1. Теоретические сведения

Структура объединяет логически связанные данные одинакового или разного типов под одним именем. Структурный тип данных определяется описанием **шаблона**:

```
struct Person {  
    char Fio[30];           // ФИО студента  
    double sball;          // Средний балл успеваемости  
};
```

Объявление переменных созданного структурного типа:

```
Person Stud, *p_Stud;
```

Обращение к элементам структур производится посредством:

1) операции принадлежности ( **.** ) в виде

*Имя\_структуры . Имя\_поля* или *(\*указатель) . Имя\_поля*

2) операции косвенной адресации (**->**) в виде

*указатель -> Имя\_поля* или *&(Имя\_структуры) . Имя\_поля*

Для приведенного выше примера

1) `Stud.Fio = "Иванов А.И.;"`; //Инициализация данных

`Stud.sball = 5.75;`

2) `p_Stud -> Fio = "Иванов А.И.;"`;

`p_Stud -> sball = 5.75;`

В языке C/C++ файл рассматривается как поток (*stream*), представляющий собой последовательность считываемых или записываемых байтов. При этом последовательность записи определяется самой программой.

#### **Работа с файлами**

Файл – это набор данных, размещенный на внешнем носителе и рассматриваемый в процессе обработки как единое целое. Прототипы большинства функций по обработке файлов описаны в библиотеках *stdio.h* и *io.h*.

Прежде чем работать с файлом, его нужно открыть для доступа, т. е. создать и инициализировать область данных, которая содержит информацию о файле: имя, путь и т. д. В языке C/C++ это выполняет функция ***fopen()***, которая связывает физический файл на носителе с логическим именем в программе. Логическое имя – это указатель на файл, т. е. на область памяти, где хранится информация о файле. Указатели на файлы необходимо декларировать:

***FILE*** \*указатель на файл;

Формат функции:

**fopen**( “Строка1” , “Строка2” );

В *Строка1* указывается место, в которое мы собираемся поместить файл, например: “d:\\work\\sved.txt” – файл с именем *sved.txt*, который будет находиться на диске *d* в папке *work*; если путь к файлу не указывать, то он будет размещен в рабочей папке проекта.

В *Строка2* указывается режим открытия файла:

**w** – для записи: если файла с заданным именем нет, то он будет создан, если файл существует, то прежняя информация уничтожается;

**r** – для чтения: если файла нет, то возникает ошибка;

**a** – для добавления новой информации в конец файла.

По умолчанию файл создается (открывается) в текстовом режиме (**t**). Для работы с файлами данных указываем режим **b** (бинарный).

Если при открытии файла произошла ошибка, функция **fopen** возвращает значение *NULL*.

После работы доступ к файлу необходимо закрыть с помощью функции **fclose** (*Указатель файла*), например **fclose (f)**;

Для закрытия нескольких файлов введена функция **void fcloseall(void)**;

Приведем пример минимального набора операторов, необходимых для корректной работы с файлом:

```
#include <stdio.h>
```

```
...
```

```
FILE *f_my;
```

```
if( ! ( f_my = fopen(“rez.txt”, “rt” ) ) ) {  
    puts(“\n Ошибка открытия файла!”);
```

```
// В оконном режиме – ShowMessage(“Ошибка открытия файла”);
```

```
    return;
```

```
}
```

```
...
```

```
//      Работа с файлом
```

```
fclose(f_my);
```

```
...
```

Для работы с текстовыми файлами в консольном приложении удобнее всего пользоваться функциями **fprintf** () и **fscanf** (), параметры и выполняемые действия аналогичны функциям **printf** () и **scanf** (), (см. тему №1), только первым параметром добавлен указатель файла, к которому применяется данная функция.

Функции работы с текстовыми файлами удобны при создании результирующих файлов для отчетов по лабораторным и курсовым работам.

Для создания баз данных удобнее пользоваться функциями работы с бинарными файлами. Рассмотрим некоторые из них, обозначив указатель файла – **fp** (**FILE \*fp**);

- 1) *int fread (void \*ptv, int size, int n, fp)* – считывает *n* блоков по *size* байт каждый из файла *fp* в область памяти, на которую указывает *ptv* (необходимо заранее отвести память под считываемый блок);
- 2) *int fwrite (void \*ptv, int size, int n, fp)* – записывает *n* блоков по *size* байт каждый из области памяти, на которую указывает *ptv*, в файл *fp*;
- 3) *int fileno (fp)* – возвращает значение *дескриптора* файла *fp* (дескриптор – число, определяющее номер файла);
- 4) *long filelength (int Дескриптор)* – возвращает длину файла в байтах;
- 5) *int chsize (int Дескриптор, long pos)* – выполняет изменение размера файла *fp*, признак конца файла устанавливается после байта с номером *pos*;
- 6) *int fseek (fp, long size, int kod)* – выполняет смещение указателя на *size* байт в направлении признака *kod*: 0 – от начала файла; 1 – от текущей позиции; 2 – от конца файла;
- 7) *long ftell (fp)* – возвращает значение указателя на текущую позицию файла *fp* (–1 – ошибка);
- 8) *int feof (Указатель файла)* – возвращает ненулевое значение при правильной записи признака конца файла;
- 9) *int fgetpos (Указатель файла, long \*pos)* – определяет значение текущей позиции *pos* файла; при успешном завершении возвращает значение 0.

## 7.2. Создание оконного приложения в среде Builder

### Компоненты OpenFileDialog и SaveDialog



Компоненты *OpenDialog* и *SaveDialog* находятся на странице *Dialogs*. Все компоненты этой страницы невидимы, т. е. не видны при работе программы, поэтому их размещают в любом месте формы. Обе компоненты имеют идентичные свойства.

После вызова компоненты появляется стандартное диалоговое окно, с помощью которого выбирается имя программы и путь к ней. В случае успешного завершения диалога имя выбранного файла и его размещение содержатся в *FileName*. Для выбора файлов, отображаемых в окне просмотра, используется свойство *Filter*, а для изменения заголовка окна – свойство *Title*.

## 7.3. Пример выполнения задания

Написать программу обработки файла, содержащего информацию об успеваемости студентов. Каждая запись должна содержать ФИО и полученный за сессию средний балл. Вывести информацию, отсортированную в порядке увеличения баллов. Результаты выполнения программы сохранить в текстовом файле. При работе с файлом должны быть выполнены следующие действия: создание, просмотр, добавление новой записи, сортировка, сохранение результатов.

## Настройка компонент *OpenDialog* и *SaveDialog*

На странице *Dialogs* выбрать пиктограммы  ,  для установки компонент *OpenDialog* и *SaveDialog* соответственно.

Для выбора нужных файлов установить фильтры следующим образом: выбрав компоненту, дважды щелкнуть кнопкой мыши по правой части свойства *Filter* инспектора объектов и в появившемся окне *Filter Editor*, в левой части записать текст, характеризующий выбор, в правой части – маску. Для *OpenDialog1* установить значения маски, как показано на рис. 7.1. Формат *\*.dat* означает, что будут видны все файлы с расширением *dat*, а формат *\*.\** – будут видны все файлы (с любыми именами и расширениями).

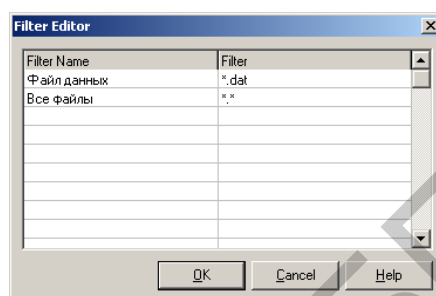


Рис. 7.1

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

Аналогичным образом настраивается *SaveDialog1* для текстового файла, который будет иметь расширение *.txt*.

## Работа с программой

Форма может иметь вид, представленный на рис. 7.2.

Кнопку «Создать» нажимаем только при первом запуске программы или если хотим заменить прежнюю информацию на новую. В окне *Мето1* отображается путь и имя созданного файла.

Заполнив оба поля информацией, нажимаем кнопку «Добавить», после чего введенная информация отображается в окне *Мето1*.

Для работы с уже созданным файлом нажимаем кнопку «Открыть» – в *Мето1* выводится содержимое всего файла, после чего можно добавлять новые данные в конец этого файла, не уничтожая предыдущие.

При нажатии кнопки «Сортировать» в *Мето1* выводятся записи, сортированные по возрастанию баллов.

При нажатии кнопки «Сохранить результаты» создается текстовый файл, в котором сохранится информация, выведенная в *Мето1*. Этот файл можно просмотреть в любом текстовом редакторе.

В текст программы включена пользовательская функция *void Out(TZap, TМето\*)*; – для вывода в *Мето1* одной записи.

Для создания результирующего текстового файла используется функция **SaveToFile** (*FileNameRez*), позволяющая записать все содержимое *Memo1* в файл с указанным именем.

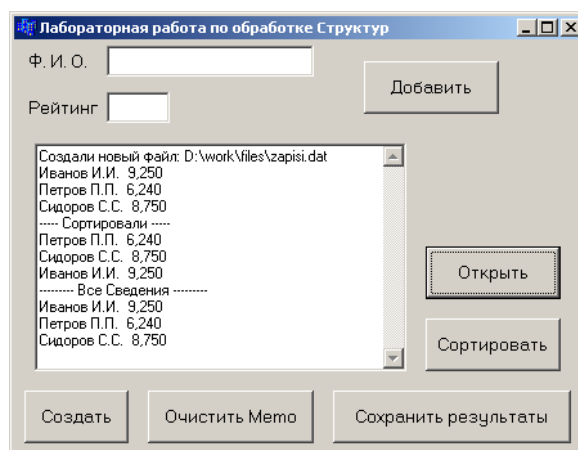


Рис. 7.2

Приведем примеры некоторых участков текста программы:

1. Функция-обработчик кнопки «Создать»:

```
OpenDialog1->Title="Создать новый файл";
if (OpenDialog1->Execute()){
    File_Zap = OpenDialog1->FileName;
    if ((Fz=fopen(File_Zap.c_str(),"wb"))==NULL) {
        ShowMessage("Ошибка создания ФАЙЛА!");
        return;
    }
}
Memo1->Lines->Add("Создали новый файл: "+AnsiString(File_Zap));
fclose(Fz);
```

2. Функция-обработчик кнопки «Добавить»:

```
Fz = fopen(File_Zap.c_str(),"ab");
strcpy(Zap.FIO, Edit1 -> Text.c_str());
Zap.s_b = StrToFloat(Edit2->Text);
Out(Zap, Memo1);
fwrite(&Zap, size, 1, Fz);
fclose(Fz);
```

3. Функция-обработчик кнопки «Сортировать»:

```
TZap st, *mas_Z;
int i, j, kol, len;
Fz = fopen(File_Zap.c_str(),"rb");
len = filelength(fileno(Fz));           // Находим размер файла
kol = len/size;                         // Находим количество записей в файле
mas_Z = new TZap[kol];
fread(mas_Z, size, kol, Fz);           // Считываем записи из файла в массив
```

```

fclose(Fz);
Memo1->Lines->Add("Сортированные сведения");
for (i=0; i < kol - 1; i++)
    for (j=i+1; j < kol; j++)
        if (mas_Z[i].s_b > mas_Z[j].s_b) {
            st = mas_Z[i];
            mas_Z[i] = mas_Z[j];
            mas_Z[j] = st;
        }
for (i=0; i<kol; i++)
    ut(mas_Z[i], Memo1);
delete []mas_Z;

```

4. Функция-обработчик кнопки «*Сохранить*»:

```

SaveDialog1->Title="Сохранить файл результатов";
if (SaveDialog1->Execute()) {
    AnsiString FileNameRez = SaveDialog1->FileName;
    Memo1->Lines->SaveToFile(FileNameRez);
}

```

5. Функция-обработчик кнопки «*Открыть*» (просмотреть все записи):

```

OpenDialog1->Title="Открыть файл";
if (OpenDialog1->Execute()) {
    File_Zap = OpenDialog1->FileName;
    if ((Fz=fopen(File_Zap.c_str(),"rb"))==NULL) {
        ShowMessage("Ошибка открытия ФАЙЛА!");
        return;
    }
}
Memo1->Lines->Add("----- Все сведения -----");
while(1){
    if(!fread(&Zap,size,1,Fz)) break;
    Out(Zap, Memo1);
}
fclose(Fz);

```

6. Функция вывода в *Memo1* одной записи:

```

void Out(TZap z, TMemo *Memo1)
{
    Memo1->Lines->Add (AnsiString ( z.FIO ) + "  "
        + FloatToStrF ( z.s_b, ffFixed,6,3));
}

```

## 7.4. Индивидуальные задания

### 7.4.1. Первый уровень сложности

Написать программу обработки файла записей, содержащую следующие пункты меню: «Создание», «Просмотр», «Добавление», «Решение индивидуального задания».

Каждая запись должна содержать следующую информацию о студентах:

- фамилия;
- номер группы;
- оценки за семестр: по физике, математике, информатике;
- средний балл.

Организовать ввод исходных данных, средний балл рассчитать по введенным оценкам.

Содержимое всего файла и результаты решения индивидуального задания записать в текстовый файл.

1. Найти информацию о студентах, сдавших сессию на 8, 9 и 10.
2. Найти информацию о студентах-отличниках, фамилии которых начинаются с интересующей вас буквы.
3. Найти информацию о студентах-отличниках из интересующей вас группы.
4. Найти информацию о студентах, сдавших математику на 8 или 9, фамилии которых начинаются с буквы *А*.
5. Найти информацию о студентах, имеющих отметку 4 или 5 по физике и больше 8 по остальным предметам.
6. Найти информацию о студентах интересующей вас группы. Фамилии студентов начинаются с букв *В*, *Г* и *Д*.
7. Найти информацию о студентах, не имеющих отметок меньше 4 по информатике и математике.
8. Вычислить общий средний балл всех студентов и распечатать список студентов со средним баллом выше общего среднего балла.
9. Вычислить общий средний балл всех студентов и распечатать список студентов интересующей вас группы, имеющих средний балл выше общего среднего балла.
10. Найти информацию о студентах интересующей вас группы, имеющих неудовлетворительную отметку (меньше 4).
11. Найти информацию о студентах интересующей вас группы, имеющих отметку 9 или 10 по информатике.
12. Найти информацию о студентах, имеющих отметку 7 или 8 по физике и 9 или 10 по высшей математике.
13. Вычислить общий средний балл студентов интересующей вас группы и распечатать список студентов этой группы, имеющих средний балл выше общего.
14. Найти информацию о студентах-отличниках интересующей вас группы.
15. Найти информацию о студентах интересующей вас группы, имеющих средний балл выше введенного с клавиатуры.
16. Найти информацию о студентах интересующей вас группы, имеющих отметку 8 по физике и 9 по высшей математике.



#### 7.4.2. Второй уровень сложности

Написать программу предыдущего варианта с добавлением пунктов меню: «Редактирование», «Удаление», «Сортировка» (для символьных данных – по алфавиту, для числовых данных – по возрастанию или убыванию).

#### 7.4.3. Третий уровень сложности

**Задачи шифровки.** Составить программу, которая вводит строку с клавиатуры (признак окончания ввода – *Enter*), шифрует введенный текст в файл на диске по заданному ниже алгоритму. Программа должна считывать эту строку из файла и далее дешифровать текст, выводя его на экран и записывая в выходной файл. Все алгоритмы реализовать в отдельных функциях.

В программе реализовать следующие действия:

- ввод с клавиатуры исходной строки текста и запись в файл;
- считывание строки из файла и вывод на экран;
- шифровка текста;
- расшифровка текста.

1. Каждая буква от «а» до «ю» заменяется на следующую по алфавиту, а «я» заменяется на «а».
2. Первая буква «а» заменяется на 11-ю, вторая «б» – на 12-ю, третья «в» – на 13-ю, ... , последняя «я» – на 10-ю.
3. После каждой согласной буквы вставляется буква «а».
4. После каждой согласной буквы вставляется слог «ла».
5. Каждая пара букв «ле» заменяется на «ю», «са» – на «щ», «ик» – на «ж».
6. Каждая пара букв «си», «ли» и «ти» заменяется соответственно на «иис», «иил» и «иит».
7. После каждой гласной буквы вставляется буква «с».
8. После каждой гласной буквы вставляется слог «ла».
9. Каждая из букв «а», «о», «и» заменяется соответственно на «ц», «ш», «щ».
10. Каждая буква заменяется на следующую в алфавите по часовой стрелке.
11. Каждая буква заменяется на следующую в алфавите против часовой стрелки.
12. Каждая буква «а» заменяется на слог «си», а «и» – на «са».
13. Четные и нечетные символы меняются местами.
14. Символы, кратные двум по порядку следования, заменяются на единицы.
15. Символы, кратные двум по порядку следования, заменяются на свой порядковый номер.

## Тема №8. Функции пользователя

**Цель работы:** познакомиться с механизмом составления и организации взаимодействия пользовательских функций, составить и отладить программу.

### 8.1. Краткие теоретические сведения

**Подпрограмма** – это именованная и определенным образом оформленная группа операторов, вызываемая по мере необходимости.

В языке C++ в качестве подпрограмм используют функции, которые должны быть декларированы до их первого использования. Предварительное описание функции называется **прототипом**, который обычно размещается в начале программы (\*.cpp) либо в заголовочном файле (\*.h) и сообщает компилятору о том, что далее в программе будет приведен ее полный текст, т. е. **реализация**.

Описание прототипа имеет следующий вид:

*Тип\_результата* *Имя\_функции* (*Список типов параметров*);

Определение функции имеет следующую структуру:

```
Тип_результата Имя_функции (Список параметров) // Заголовок
{
    Код функции
    return Результат;
}
```

*Результат* возвращается из функции в точку ее вызова при помощи оператора **return** и преобразуется к типу, указанному в заголовке функции. Если *Тип результата* не указан, то по умолчанию устанавливается тип **int**. Если функция не возвращает результат, то указывается пустой тип **void**. *Список параметров* состоит из перечня типов и имен параметров, разделенных запятыми.

Из функции оператором **return** можно передать только один результат, при необходимости вернуть несколько результатов – в списке параметров используют указатели, или ссылки.

Пример реализации функции, определяющей наименьшее из двух целых чисел:

```
int Min_x_y(int x, int y) {
    return (x < y) ? x : y;
}
```

**Вызов функции** имеет следующий формат:

*Имя\_функции* (*Список аргументов*)

Здесь в качестве аргументов можно использовать константы, переменные, выражения (их значения перед вызовом функции будут определены компилятором). Аргументы списка вызова должны совпадать со списком параметров

вызываемой функции по количеству и порядку следования, а типы аргументов при передаче будут преобразованы к указанным в функции типам параметров.

Вызов предыдущей функции может иметь вид `min = Min_x_y(a, b);`

### **Область действия переменных**

Область действия – это правила доступа к данным из текущего места программы, определяющие переменные двух типов: *глобальные* и *локальные*.

**Глобальные** переменные объявляются вне функций и могут быть использованы в любом месте файла (проекта). Область действия глобальных переменных – весь файл (проект) с момента их объявления. **При объявлении глобальные переменные обнуляются.** Классы памяти – статическая (*static*), внешняя (*extern*).

Область действия **локальных** переменных – блоки, где они объявлены. К локальным переменным относятся и параметры функций пользователя. При выходе из блока (функции) значения локальных переменных теряются. Классы памяти – автоматическая (*auto*), регистровая (*register*).

### **Способы передачи параметров**

**Передача параметров по значению** выглядит следующим образом:

```
double function (int x);
```

Вызов этой функции для *double f* и *int k* может иметь вид

```
f = function (k);
```

При вызове функции *function(k)* будет создана копия параметра *x* для переданного ей аргумента *k*, и если в процессе выполнения функции значение *x* изменится, то это никак не отразится на значении переменной *k*.

Если из функции необходимо передать более одного значения или если необходимо вернуть параметр, переданный в функцию и измененный в ней, то можно применить передачу параметров либо по ссылке (копия адреса), либо по адресу (с помощью указателя).

**Передача параметров по ссылке** выглядит следующим образом:

```
double function (int &x);
```

Вызов этой функции будет иметь такой же вид, как в предыдущем примере. Здесь при вызове функции создается копия адреса переменной *k* (*&x*), переданной в функцию, и если в процессе выполнения функции значение *x* изменится, то изменится и значение переменной *k*.

**Передача параметров по адресу** выглядит следующим образом:

```
double function (int *x);
```

Вызов этой функции для *double f* и *int k* может иметь вид

```
f = function (&k);
```

Здесь в качестве параметра используется указатель *x*, а в функцию передается адрес аргумента (*&k*), и если при выполнении функции значение, нахо-

дящееся по адресу  $x$  (в коде функции это значение  $*x$ ), изменится, то изменится и значение переменной  $k$ .

## 8.2. Пример выполнения задания

Написать программу вычисления выбранной функции  $e^x$  или  $\sin(x)$  для  $x$ , изменяющегося от  $a$  до  $b$  с шагом  $h$ , вид которой, в свою очередь, передается в качестве параметра в функцию вывода (*Out\_Rez*).

### Создание оконного приложения

Приведем некоторые участки текста программы, необходимые для решения поставленной задачи.

1. Описание функций пользователя (определение, прототип):

```
double fun1(double r){
    return exp(r);
}
double fun2(double r) {
    return sin(r);
}
void Out_Rez(double f(double), double, double, double, TMemo*);
```

2. Текст функции *Button1Click* («Вычислить»):

```
double a, b, h;
a = StrToFloat(Edit1->Text);
b = StrToFloat(Edit2->Text);
h = StrToFloat(Edit3->Text);
switch(RadioGroup1->ItemIndex) {
    case 0:      Out_Rez (fun1, a, b, h, Memo1);      break;
    case 1:      Out_Rez (fun2, a, b, h, Memo2);      break;
}
```

3. Текст функции *Out\_Rez*:

```
void Out_Rez(double f(double x), double xn, double xk, double h, TMemo *mem)
{
    for(double x=xn; x<=xk; x+=h)
        mem->Lines->Add(" x = " + FloatToStrF(x,ffFixed,8,2) +
                        " y = " + FloatToStrF(f(x),ffFixed,8,4));
}
```

## 8.3. Индивидуальные задания

### 8.3.1. Первый уровень сложности

На основании задания, размещенного в п. 3.3.1, написать программу расчета выбранной функции  $Y(x)$  или  $S(x)$  (желательно и  $|Y(x) - S(x)|$ ), вид которой, в свою очередь, передается в качестве параметра в функцию вывода (*Out\_Rez*).

### 8.3.2. Второй уровень сложности

На основании задания, размещенного в п. 3.3.2, написать программу расчета выбранной функции  $Y(x)$ ,  $S(x)$  или  $|Y(x) - S(x)|$ , вид которой, в свою очередь, передается в качестве параметра в функцию вывода (*Out\_Rez*).

### 8.3.3. Третий уровень сложности

Решить поставленную задачу с использованием рекурсивной и обычной функций. Сравнить полученные результаты.

1. Определить, является ли заданная строка палиндромом, т. е. читается одинаково слева направо и справа налево.

2. В упорядоченном массиве целых чисел  $a_i$  ( $i = 1, \dots, n$ ) найти номер находящегося в массиве элемента  $s$ , используя метод двоичного поиска.

3. Найти наибольший общий делитель (НОД) чисел  $M$  и  $N$ , используя теорему Эйлера: если  $M$  делится на  $N$ , то  $\text{НОД}(N, M) = N$ , иначе  $\text{НОД}(N, M) = (M \bmod N, N)$ .

4. Числа Фибоначчи определяются следующим образом:  $Fb(0) = 0$ ;  $Fb(1) = 1$ ;  $Fb(n) = Fb(n-1) + Fb(n-2)$ . Определить  $Fb(n)$ .

5. Вычислить длину строки.

6. Найти методом деления отрезка пополам минимум функции  $f(x) = 7\sin^2(x)$  на отрезке  $[2, 6]$  с заданной точностью  $\varepsilon$  (например, 0.01).

7. Вычислить значение  $x = \sqrt{a}$ , используя рекуррентную формулу  $x_n = \frac{1}{2} \left( x_{n-1} + \frac{a}{x_{n-1}} \right)$ , в качестве начального значения использовать  $x_0 = 0.5(1 + a)$ .

8. Найти максимальный элемент в массиве  $a_i$  ( $i = 1, \dots, n$ ), используя очевидное соотношение  $\max(a_1, \dots, a_n) = \max[\max(a_1, \dots, a_{n-1}), a_n]$ .

9. Вычислить значение  $y(n) = \sqrt{1 + \sqrt{2 + \dots + \sqrt{n}}}$ .

10. Найти максимальный элемент в массиве  $a_i$  ( $i = 1, \dots, n$ ), используя соотношение (деления пополам)  $\max(a_1, \dots, a_n) = \max[\max(a_1, \dots, a_{n/2}), \max(a_{n/2+1}, \dots, a_n)]$ .

11. Вычислить значение  $y(n) = \frac{1}{n + \frac{1}{(n-1) + \frac{1}{(n-2) + \frac{1}{\dots + \frac{1}{\dots + \frac{1}{\left(1 + \frac{1}{2}\right)}}}}}}$ .

12. Вычислить произведение четного количества  $n$  ( $n \geq 2$ ) сомножителей следующего вида  $y = \left(\frac{2}{1} \cdot \frac{2}{3}\right) \cdot \left(\frac{4}{3} \cdot \frac{4}{5}\right) \cdot \left(\frac{6}{5} \cdot \frac{6}{7}\right) \dots$ .

13. Вычислить  $y = x^n$  по следующему правилу:  $y = (x^{n/2})^2$ , если  $n$  четное и  $y = x \cdot y^{n-1}$ , если  $n$  нечетное.

14. Вычислить значение  $C_n^k = \frac{n!}{k!(n-k)!}$  (значение  $0! = 1$ ).


15. Вычислить  $y(n) = \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{\dots}}}}$ ,  $n$  задает число ступеней.

## Тема №9. Отображение графической информации

**Цель работы:** изучить некоторые возможности построения графиков функций с помощью компоненты *Chart* и научиться работать с графическими объектами с помощью класса *Canvas* и компоненты *Image*. Написать и отладить программу с использованием функций отображения графической информации.

### 9.1. Краткие теоретические сведения

#### *Построение графиков с помощью компоненты Chart*

Обычно результаты расчетов представляются в виде графиков и диаграмм. Система *Builder* имеет мощный пакет стандартных программ вывода на экран и редактирования графической информации, который реализуется с помощью компоненты *Chart*, находящейся на панели компонент *Additional* – .

Построение графика (диаграммы) производится по вычисленным значениям координат точек  $x$  и  $y = f(x)$ , которые с помощью метода *AddXY* передаются в специальный массив *Series[k]* компоненты *Chart* ( $k = 0, 1, 2, \dots$  – номер используемого графика).

Компонента *Chart* строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданные точки в виде графиков или диаграмм.

Установив компоненту *Chart1* на форму, для изменения ее параметров двойным щелчком кнопки мыши вызываем окно редактирования *EditingChart1* (рис. 9.1). Для создания *Series1* (*Series[1]*) нажимаем кнопку *Add* на странице *Series*.

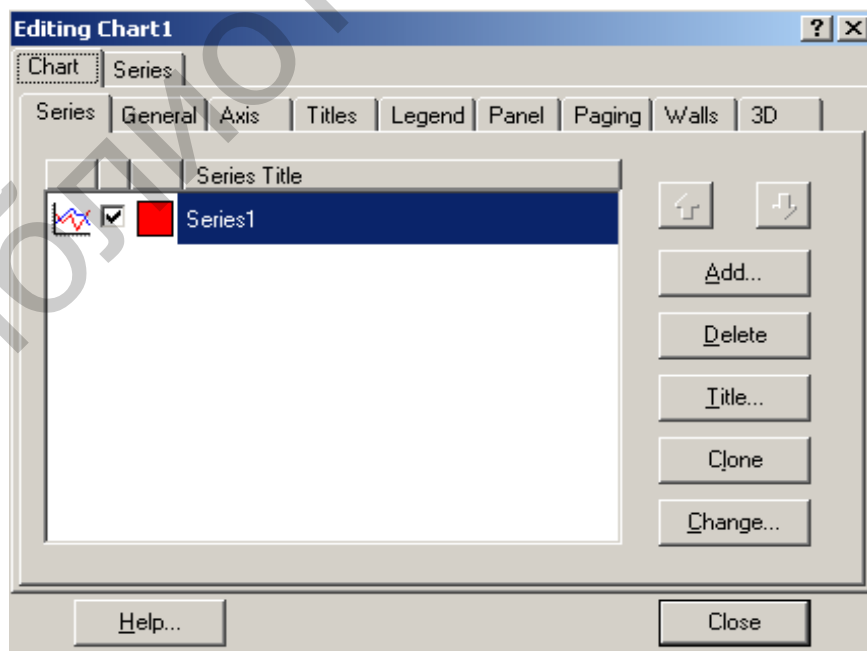


Рис. 9.1

В появившемся после этого окне *TeeChart Gallery* выбираем пиктограмму с надписью *Line* (график выводится в виде линий). Если нет необходимости представления графика в трехмерном виде, отключается независимый переключатель *3D*. Для изменения названия нажимаем кнопку *Title*. Название графика вводится на странице *Titles*.

Данные по оси *X* автоматически сортируются, поэтому, если необходимо нарисовать, например, окружность, сортировку отключают функцией *Order: Chart1->Series[0]->XValues->Order = loNone*.

Нажимая различные кнопки меню, познакомьтесь с другими возможностями редактора *EditingChart*.

Построение графика функции  $\cos(x)$  на отрезке  $[a; b]$  с шагом  $h$  в *Chart* может иметь вид

```
Chart1->Series[0]->Clear();           // Очистка графика
for(double x = a; x <= b; x += h)
    Chart1->Series[0]->AddXY(x, cos(x));
```

### **Использование класса *Canvas***

Для рисования используется класс типа *TCanvas*, который является не самостоятельной компонентой, а свойством многих компонент, таких как *Image*, *PaintBox*, и представляет собой холст (контекст *GDI* в *Windows*) с набором инструментов для рисования. Каждая точка холста имеет свои координаты. Начало осей координат располагается в верхнем левом углу холста. Данные по оси *X* увеличиваются слева направо, а по оси *Y* – сверху вниз.

Компонента *Image* находится на странице *Additional*, а *PaintBox* – *System*.

Основные свойства класса *Canvas*:

***Pen*** – перо (определяет параметры линий);

***Brush*** – кисть (определяет фон и заполнение замкнутых фигур);

***Font*** – шрифт (определяет параметры шрифта).

Некоторые методы класса *Canvas*:

***Ellipse*** ( $x1, y1, x2, y2$ ) – чертит эллипс в охватывающем прямоугольнике ( $x1, y1, x2, y2$ ) и заполняет внутреннее пространство эллипса текущей кистью;

***MoveTo*** ( $x, y$ ) – перемещает карандаш в положение ( $x, y$ );

***LineTo*** ( $x, y$ ) – чертит линию от текущего положения пера до точки ( $x, y$ );

***Rectangle*** ( $x1, y1, x2, y2$ ) – вычерчивает и заполняет прямоугольник ( $x1, y1, x2, y2$ ). Для вычерчивания без заполнения используйте *FrameRect* или *Polyline*;

***Polygon*** (*const TPoint\* Points, const int Points\_Size*) – вычерчивает многоугольник по точкам, заданным в массиве *Points* размером *Points\_Size*. Конечная точка соединяется с начальной и многоугольник заполняется текущей кистью. Для вычерчивания без заполнения используется метод *Polyline*.



**TextOut** (*x, y, const AnsiString Text*) – выводит строку *Text* так, чтобы левый верхний угол прямоугольника, охватывающего текст, располагался в точке (*x, y*).

## 9.2. Пример выполнения задания

Составить программу, отображающую движение автомобиля.

1. Функция создания и вывода изображения автомобиля:

```
void Avtomobil (int xx, int yy, TColor cc)
{
    Form1->Canvas->Pen->Color = cc;           // Цвет карандаша Pen
    Form1->Canvas->Brush->Color = cc;          // Цвета кисти Brush
    TPoint pnt[7];                             // Создание массива точек pnt для кузова
    pnt[0] = Point(xx+0,yy+20);
    pnt[1] = Point(xx+0,yy+40);
    pnt[2] = Point(xx+110,yy+40);
    pnt[3] = Point(xx+110,yy+20);
    pnt[4] = Point(xx+90,yy+20);
    pnt[5] = Point(xx+70,yy+0);
    pnt[6] = Point(xx+20,yy+0);
    // Вывод изображения составных элементов автомобиля:
    Form1->Canvas->Polygon(pnt,6);               // Кузов
    Form1->Canvas->Ellipse(xx+10,yy+30,xx+30,yy+50); // Колесо 1
    Form1->Canvas->Ellipse(xx+80,yy+30,xx+100,yy+50); // Колесо 2
}
```

2. Часть функции-обработчика для имитации движения автомобиля:

```
Canvas->Pen->Color = clBlack;           // Установка цвета карандаша
Canvas->Brush->Color = clGreen;          // Установка цвета кисти
Canvas->Rectangle(10,10,520,90);         // Изображение прямоугольника
double hx = 1, h = 0, x = 10;
int n = 0;
while (x < 410)
{
    Avtomobil (x, 40, clYellow);         // Рисование автомобиля
    Sleep(10);                            // Задержка
    Avtomobil (x, 40, clGreen);           // Стирание автомобиля
    x += hx;
    n++;
}
```

### **9.3. Индивидуальные задания**

#### **9.3.1. Первый уровень сложности**

На основании заданий п. 3.3.1 написать программу вывода графиков функции  $Y(x)$ , ее разложения в ряд  $S(x)$  и модуля их разности  $|Y - S|$  для аргумента  $x$ , изменяющегося от  $a$  до  $b$  с шагом  $h$  (вводятся с клавиатуры) с использованием компоненты *Chart* и графика функции  $Y(x)$  с использованием компоненты *Image*.

#### **9.3.2. Второй уровень сложности**

1. Написать программу, которая выводит на экран флаг олимпийских игр (круги разных цветов).
2. Написать программу, которая, используя метод базовой точки, выводит на экран изображение кораблика.
3. Написать программу, которая вычерчивает на экране узор из 100 окружностей случайного диаметра и цвета.
4. Написать программу, которая вычерчивает на экране ломаную линию, состоящую из 200 звеньев, окрашенных в разные цвета, выбираемые случайным образом, причем координаты звеньев тоже выбираются случайно.
5. Написать программу, которая выводит на экран контур пятиконечной звезды.
6. Написать программу, которая рисует флаг Республики Беларусь.
7. Написать программу, которая выводит на экран изображение шахматной доски.
8. Написать программу, которая рисует на экране раскрытую книгу.

#### **9.3.3. Третий уровень сложности**

В задании второго уровня сложности имитировать движение построенного изображения с помощью клавиш (вверх, вниз и т. п.).

## Основные математические функции

Для использования математических функций необходимо подключить файл ***math.h***. Параметры и возвращаемые результаты большинства математических функций имеют тип ***double***.

Аргументы тригонометрических функций задаются в радианах. Напомним, что  $2\pi$  (радиан) равно  $360^\circ$  (градусов).

Математическая функция	Имя функции
$\sqrt{x}$	<i>sqrt</i> (x)
$ x $	<i>fabs</i> (x)
$e^x$	<i>exp</i> (x)
$x^y$	<i>pow</i> (x, y)
$\ln x$	<i>log</i> (x)
$\lg_{10} x$	<i>log10</i> (x)
$\sin x$	<i>sin</i> (x)
$\cos x$	<i>cos</i> (x)
$\operatorname{tg} x$	<i>tan</i> (x)
$\arcsin x$	<i>asin</i> (x)
$\operatorname{arctg} x$	<i>atan</i> (x)
$\operatorname{arctg} (x / y)$	<i>atan2</i> (x, y)
$\operatorname{sh}^*(x) = (e^x - e^{-x}) / 2$	<i>sinh</i> (x)
$\operatorname{ch}(x) = (e^x + e^{-x}) / 2$	<i>cosh</i> (x)
$\operatorname{tgh} x$	<i>tanh</i> (x)
Остаток от деления x на y	<i>fmod</i> (x, y)
Округление к большему	<i>ceil</i> (x)
Округление к меньшему	<i>floor</i> (x)

\* Синус гиперболический, а в следующих строках – косинус и тангенс.

## Описание общих структур файлов проекта

До тех пор пока вы не научились основным действиям, необходимым для изменения настроек и свойств элементов, входящих в проект, советуем не изменять имен файлов и других элементов, присвоенных им автоматически.

**Внимание! Не изменять и не удалять уже имеющиеся в указанных файлах тексты.**

### Общая структура файла текста программы *Unit\*.cpp*

```
// Директивы препроцессора
#include <vcl.h>           // Подключение файла библиотеки VCL
#pragma hdrstop           // Установки компилятора
#include "Unit1.h"         // Подключение заголовочного файла
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;            // Объявление объекта формы
//----- Вызов конструктора формы -----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
// Описания глобальных объектов пользователя
// Набор функций-обработчиков, использующихся в проекте
```

### Структура заголовочного файла *Unit\*.h*

```
// Директивы препроцессора
#ifndef Unit1H
#define Unit1H
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
// Объявление класса формы
class TForm1 : public TForm
{
```

```

// Компоненты, размещенные на форме
    __published:                                // IDE-managed Components
        TLabel *Label1;
        TEdit *Edit1;
        TMemo *Memo1;
        TButton *Button1;

    private:                                    // User declarations
// Объявления функций, типов переменных, доступных только в данном модуле
    public:                                    // User declarations
// Объявления функций, типов и переменных, доступных в проекте

    __fastcall TForm1(TComponent* Owner);
};
    extern PACKAGE TForm1 *Form1;
// Объявления элементов, которые не включаются в данный класс
#endif

```

### **Общая структура файла проекта Project\*.cpp**

```

// Директивы препроцессора
#include <vcl.h>
#pragma hdrstop
/* – директива #pragma служит для установки параметров компилятора. Эти
установки могут быть определены и другим способом с использованием диало-
га Project Options. */
// Подключение файлов форм и файлов ресурсов
USEFORM("Unit1.cpp", Form1);
USEFORM("Unit2.cpp", Form2);
// Главная программа
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
// Инициализация
    Application->Initialize();
// Создание объектов форм
    Application->CreateForm(__classid(TForm1), &Form1);
    Application->CreateForm(__classid(TForm2), &Form2);
// Выполнение программы
    Application->Run();
}

```

## ЛИТЕРАТУРА

1. Березин, Б. И. Начальный курс С и С++ / Б. И. Березин, С. Б. Березин. – М. : ДИАЛОГ-МИФИ, 2005.
2. Демидович, Е. М. Основы алгоритмизации и программирования. Язык СИ / Е. М. Демидович. – СПб. : БХВ – Петербург, 2006.
3. Керниган, Б. Язык программирования СИ / Б. Керниган, Д. Ритчи. – М. : Вильямс, 2006.
4. Страуструп, Б. Программирование: принципы и практика использования С++ / Б. Страуструп. – М. : Вильямс, 2011.
5. Страуструп, Б. Язык программирования С++ / Б. Страуструп. – М. : Бином, 2011.
6. Архангельский, А. Я. Программирование в С++ Builder / А. Я. Архангельский. – 7-е изд. – М. : Бином, 2010.
7. Шилд, Г. С++: базовый курс / Г. С. Шилд. – 3-е изд. – М. : Вильямс, 2012.

*Учебное издание*

**ОСНОВЫ АЛГОРИТМИЗАЦИИ  
И ПРОГРАММИРОВАНИЯ (ЯЗЫК С/С++).  
ЛАБОРАТОРНЫЙ ПРАКТИКУМ**

В двух частях

Часть 1

**Беспалов Сергей Алексеевич  
Зайцева Ирина Евгеньевна  
Кривоносова Татьяна Михайловна и др.**

**УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ**

Редактор *М. А. Зайцева*  
Корректор *Е. И. Герман*  
Компьютерная правка, оригинал-макет *Е. Г. Бабичева*

Подписано в печать 20.10.2016. Формат 60×84 1/16. Бумага офсетная. Гарнитура Таймс.  
Отпечатано на ризографе. Усл. печ. л. 4,3. Уч.-изд. л. 4,8. Тираж 300 экз. Заказ 28.

Издатель и полиграфическое исполнение: учреждение образования  
«Белорусский государственный университет информатики и радиоэлектроники».  
Свидетельство о государственной регистрации издателя, изготовителя,  
распространителя печатных изданий №1/238 от 24.03.2014,  
№2/113 от 07.04.2014, №3/615 от 07.04.2014.  
ЛП №02330/264 от 14.04.2014.  
220013, Минск, П. Бровки, 6