

Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики
и радиоэлектроники»

Кафедра «Вычислительные методы и программирование»

Основы алгоритмизации и программирования (язык C/C++)

Лабораторный практикум по курсу

«Основы алгоритмизации и программирования»

для студентов 1-го курса БГУИР

В 2-х частях

Часть 1

Минск 2015

УДК 004.42 (076.5)
ББК 32.973.26-018.1я73
О – 75

Авторы:

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

Основы алгоритмизации и программирования (язык C/C++):
лаб.практикум по курсу «Основы алгоритмизации и программирования» для
студ. 1-го курса БГУИР. В 2 ч. Ч. 1 / Беспалов С. А. [и др.] . – Минск: БГУИР,
2015. – 80 с.: ил.

ISBN (ч. 1)

В практикуме изложены краткие теоретические сведения по основам алгоритмического языка C/C++ и программированию в современных средах программирования. Содержатся примеры реализации алгоритмов в консольном и оконном приложениях среды C++ *Builder*, приведены 8 лабораторных работ и индивидуальные задания к ним.

УДК 004.42 (076.5)
ББК 32.973.26-018.1я73

ISBN (ч. 1)
ISBN 985-444-583-6

© УО «Белорусский
государственный
университет информатики
и радиоэлектроники», 2015

СОДЕРЖАНИЕ

ТЕМА №1. ЛИНЕЙНЫЙ ВЫЧИСЛИТЕЛЬНЫЙ ПРОЦЕСС.....	5
1.1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	5
1.2. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ В СРЕДЕ BUILDER.....	9
1.3. СОЗДАНИЕ КОНСОЛЬНОГО ПРИЛОЖЕНИЯ	12
1.4. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ.....	14
1.5. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ.....	18
ТЕМА №2. РЕАЛИЗАЦИЯ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ	22
2.1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	22
2.2. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ В СРЕДЕ BUILDER.....	23
2.3. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ.....	24
2.4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ.....	26
ТЕМА №3. РЕАЛИЗАЦИЯ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ	31
3.1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	31
3.2. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ.....	32
3.3. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ.....	33
ТЕМА №4. ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ	36
4.1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	36
4.2. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ	36
4.3. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ.....	37
4.4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ.....	39
ТЕМА №5. ОБРАБОТКА ДВУХМЕРНЫХ ДИНАМИЧЕСКИХ МАССИВОВ	42
5.1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	42
5.2. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ.....	44
5.3. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ.....	45
ТЕМА №6. ИСПОЛЬЗОВАНИЕ СТРОК	49
6.1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	49
6.2. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ	50
6.3. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ.....	52
6.4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ.....	53
ТЕМА №7. ОБРАБОТКА СТРУКТУР С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВ	56
7.1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	56
7.2. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ	58
7.3. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ.....	58
7.4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ.....	62
ТЕМА №8. ФУНКЦИИ ПОЛЬЗОВАТЕЛЯ	65
8.1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	65
8.2. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ.....	67
8.3. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ.....	68
ТЕМА №9. ПОСТРОЕНИЕ ГРАФИКОВ ФУНКЦИЙ.....	70
9.1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	70
9.2. ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ.....	71

ПРИЛОЖЕНИЕ 1. ОПЕРАЦИИ И ОСНОВНЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ	73
ПРИЛОЖЕНИЕ 2. ОПИСАНИЕ ОБЩИХ СТРУКТУР ФАЙЛОВ ПРОЕКТА.....	75
ЛИТЕРАТУРА	77

Тема №1. Линейный вычислительный процесс

Цель работы: изучить правила составления текстов программ, научиться реализовывать линейные алгоритмы. Написать и отладить программу линейного алгоритма.

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 основным (базовым) типам данных относятся целый (*int*), вещественный (*float*, *double*) и символьный (*char*) типы. В свою очередь, данные целого типа могут быть короткими (*short*) и длинными (*long*), со знаком (*signed*) и 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. п.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
Вещественные с плавающей точкой	$\pm n.mE\pm p$ смысл записи $\pm n,m \cdot 10^{\pm p}$	1.01E-10 0.12537e+4
Символьные	' * '	'A' 'x' '0' '<'
Строковые	" ***** "	"Minsk" "Press any key"

В таблице * – любой символ, набранный на клавиатуре.

Декларация объектов

Все объекты (переменные, массивы и т.д.), с которыми работает программа, необходимо декларировать. В декларации объектам присваиваются идентификаторы (*ID*), которые могут включать латинские буквы, символ нижнего подчеркивания «_» и цифры, причем первым символом *ID* 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 *ID* переменных строчные буквы, а в именованных константах – прописные, например:

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

Разделителями *ID* являются пробелы, символы табуляции, перевода строки и страницы, а также комментарии.

Комментарий – любая последовательность символов, начинающаяся парой символов */** и заканчивающаяся парой символов **/* или начинающаяся *//* и 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 от деления 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:

```
int i, n;  
float x;  
i = 7;  
x = i / 4; // x=1, делится целое на целое
```

$x = i / 4;$ // $x=1.75$, делится целое на дробное
 $x = (\text{float}) i / 4;$ // $x=1.75$, делится дробное на целое
 $n = 7. / 4.;$ // $n=1$, результат записывается в целую переменную

Операция присваивания имеет полную и сокращенную формы записи.

Полная форма: $ID = \text{выражение};$

– выполняется справа налево, т.е. сначала вычисляется *выражение*, а затем его результат присваивается указанному *ID*, например: $y = (x+2)/(3*x) - 5;$

В одном операторе можно присвоить значение нескольким переменным, например: $x = y = z = 0;$

или $z = (x = y) * 5;$ – сначала переменной *x* присваивается значение переменной *y*, далее вычисляется выражение $x*5$, и результат присваивается переменной *z*.

Сокращенная форма: $ID \text{ операция} = \text{выражение};$

где **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 операции **инкремента** ($--$) и **декремента** ($++$), т.е. уменьшения или увеличения значения переменной на 1. Операции могут быть **префиксные** ($++i$ и $--i$) и **постфиксные** ($i++$ и $i--$). При использовании данной операции в выражении 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 рисунке 1.1.

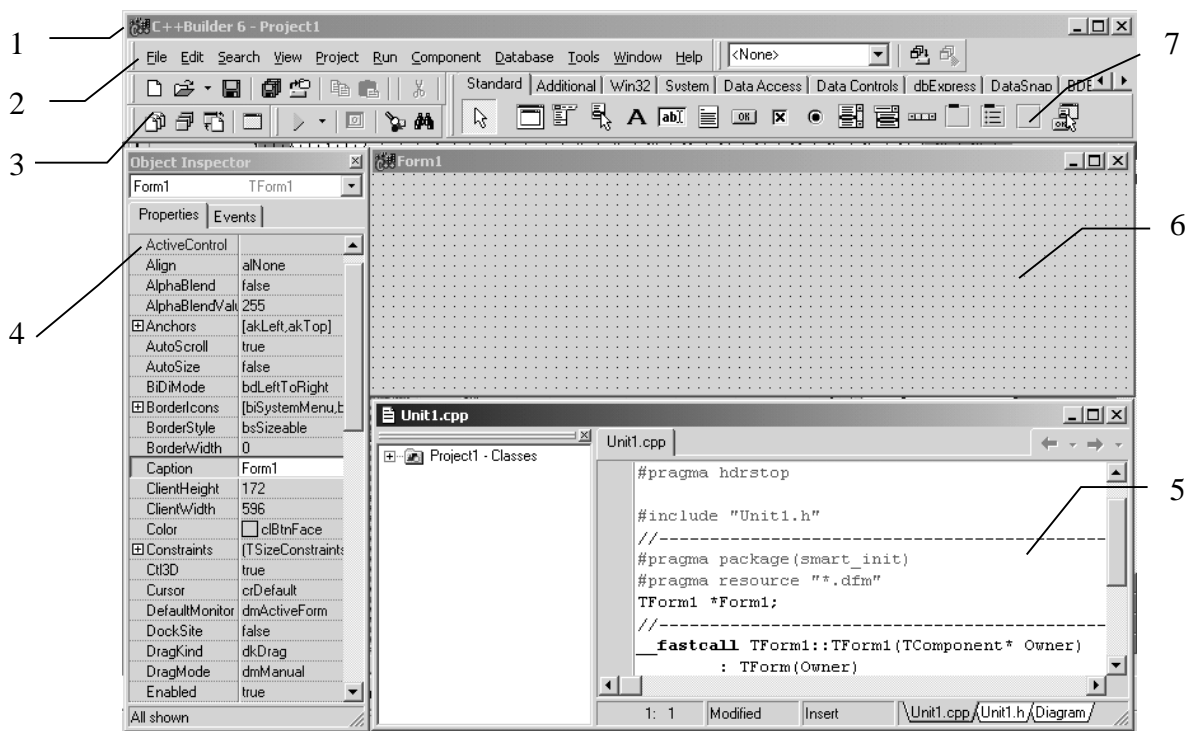


Рисунок 1.1

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

Главное окно предназначено для управления процессом создания программы. Основное меню содержит все необходимые средства для управления проектом. Пиктограммы облегчают доступ к наиболее часто применяемым командам основного меню. Через меню компонент осуществляется доступ к набору стандартных сервисных программ среды, которые описывают некоторый визуальный элемент (компоненту), помещенный программистом в окно формы. Каждая компонента имеет определенный набор свойств, которые задаются Пользователем, например, заголовок окна, надпись на кнопке, стиль выводимого текста и т.п.

Окно инспектора объектов (вызывается нажатием клавиши **F11**) предназначено для изменения свойств выбранных компонент и состоит из двух страниц. Страница *Properties* (свойства) предназначена для изменения необходимых свойств компоненты, страница *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.

Окно текста (имя *Unit1.cpp*), предназначено для написания, редактирования и просмотра текста программы. При первоначальной загрузке в окне тек-

ста программы находится текст, содержащий минимальный набор операторов для нормального функционирования пустой формы в качестве *Windows*-окна.

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

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

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

Структура программ 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 как консольного, так и 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 , для разворачивания формы на весь экран , для возвращения к исходному размеру  и для 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, «Лаб.раб.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.

Компоненты, предназначенные для ввода-вывода

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** выбирается пиктограмма  и щелчком кнопкой мыши устанавливается 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.

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

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

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**.

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

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

Например, 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 «**заятая**»!

В инспекторе объектов с помощью свойства **Font** устанавливается стиль, отражаемого в строке *Edit** текста.


Компонента Label

Данная компонента используется для вывода надписей, для чего в меню компонент *Standard* выбирается пиктограмма , и в нужном месте формы устанавливается надпись *Label1*, регулируется размер, место положения, изменяется свойство *Caption* инспектора объектов, в котором вводится нужный текст, например, строка “Значение X = ”, а также выбирается 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 работы программы обычно используется окно многострочного редактора текста, представленное компонентой *Memo*, для чего выбирается пиктограмма , помещается на форму, регулируется ее размер и местоположение. После установки с помощью инспектора свойства *ScrollBars – SSBoth* в окне появятся вертикальная и горизонтальная полосы прокрутки.

При установке данной компоненты в *Unit1.h* прописывается 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» (*OnCreate*). Оформим функцию-обработчик 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 на 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. пример, п. 1.4.1).


Внимание! Не набирайте заголовки функций-обработчиков вручную.


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

Выбрав в меню *Standard* пиктограмму , помещаем на форму компоненту *Button1* (2,3,...). С помощью инспектора объектов изменяем заголовок (*Caption*) на текст, например, «*Выполнить*», регулируем положение и размер кнопки. Двойным щелчком кнопкой мыши по компоненте *Button1* 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 (см. 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 объектов 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 объектов 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

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 (*int*); *%c* – один символ (*char*);

%s – строка символов (*string*); *%f* – данные типа *float*;

%ld – длинное целое; *%lf* – данные типа *double*;

%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.

Управляющие символы: *\n* – переход на новую строку; *\t* – горизонтальная; *\v* – вертикальная табуляция; *\b* – возврат назад на один символ; *\r* – возврат в начало строки; *\a* – звуковой сигнал; *\f* – прогон бумаги до начала новой страницы; *\?* – знак вопроса.

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 информации любого вида предназначена функция

scanf (*управляющая строка, список адресов объектов ввода*);

в *управляющей строке* которой указываются только спецификации преобразований, а в *списке объектов ввода* – **адреса** вводимых скалярных переменных, для чего перед *ID* 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 как в одной строке через пробел, так и в разных строках.

Потоковый ввод-вывод информации

Оператор *cout* из библиотеки *iostream.h* обеспечивает вывод информации на экран.

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

Оператор *cin* из библиотеки *iostream.h* обеспечивает ввод строки до появления первого пробела.

Пример ввода 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 арифметического выражения для заданных значений *x*, *y*, *z* :

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

1.4.1. Пример создания оконного приложения в среде Builder

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

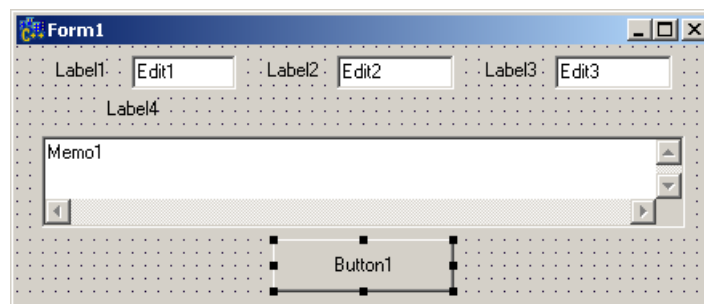


Рисунок 1.2

Для создания проекта необходимо выполнить следующие действия.

1. Запускаем C++ *Builder*. Создаем в разрешенной для Пользователя папке (d:\work\ или c:\work\)) папку с номером группы (\610101\), открыв ее, сохраняем предлагаемые файлы *Unit1.cpp* и *Project1.cpp* (рекомендуем без изменения).

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 кнопкой мыши по свободному месту формы создаем функцию *FormCreate* и заполняем ее (см. пример). Переходим на форму (*F12*), щелкаем дважды по кнопке «ВЫПОЛНИТЬ» и заполняем созданную функцию *Button1Click* (см. пример).

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

5. Запускаем проект на выполнение, исправляем ошибки.

Текст программы может иметь следующий вид (наклонным мелким шрифтом выделен текст, 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 разделитель целой и дробной частей – запятая: Edit1->Text = "3,4"; в отличие от числовых констант в тексте программы.

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

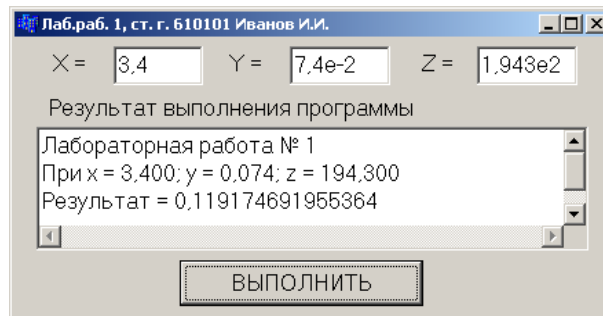


Рисунок 1.3

1.4.2. Создание консольного приложения в среде 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 среде 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 MVS 6.0, то создание консольного приложения на этом завершено.

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

В более поздних версиях среды Visual могут отсутствовать расширения в некоторых библиотеках (iostream, iomanip). Для того, чтобы проект был собран корректно мы используем директиву *using*:

```
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
#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 вводятся с клавиатуры, $\pi = 3.1415926$, α и β – в радианах.

1. $z_1 = 2\sin^2(3\pi - 2\alpha)\cos^2(5\pi + 2\alpha)$, $z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$.
2. $z_1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha$, $z_2 = 2\sqrt{2}\cos\alpha \cdot \sin\left(\frac{\pi}{4} + 2\alpha\right)$.
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 отрицательно-го числа, выход значений аргументов используемых функций за допустимые пределы. При вводе данных использовать проверку на ввод не числовых данных.

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

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

$$3. z_1 = \frac{1 - 2\sin^2 \alpha}{1 + \sin 2\alpha}, \quad z_2 = \frac{1 - \operatorname{tg} \alpha}{1 + \operatorname{tg} \alpha}.$$

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

$$5. z_1 = \frac{\sin \alpha + \cos(2\beta - \alpha)}{\cos \alpha - \sin(2\beta - \alpha)}, \quad z_2 = \frac{1 + \sin 2\beta}{\cos 2\beta}.$$

$$6. z_1 = \frac{\sqrt{(3\alpha + 2)^2 - 24\alpha}}{3\sqrt{\alpha} - 2/\sqrt{\alpha}}, \quad z_2 = -\sqrt{\alpha}.$$

$$7. z_1 = \frac{(\alpha - 1)\sqrt{\alpha} - (\beta - 1)\sqrt{\beta}}{\sqrt{\alpha^3\beta + \alpha\beta + \alpha^2 - \alpha}}, \quad z_2 = \frac{\sqrt{\alpha} - \sqrt{\beta}}{\alpha}.$$

$$8. z_1 = \frac{\sqrt{2\alpha + 2\sqrt{\alpha^2 - 4}}}{\sqrt{\alpha^2 - 4 + \alpha + 2}}, \quad z_2 = \frac{1}{\sqrt{\alpha + 2}}.$$

$$9. z_1 = \frac{\alpha^2 + 2\alpha - 3 + (\alpha + 1)\sqrt{\alpha^2 - 9}}{\alpha^2 - 2\alpha - 3 + (\alpha - 1)\sqrt{\alpha^2 - 9}}, \quad z_2 = \sqrt{\frac{\alpha + 3}{\alpha - 3}}.$$

$$10. z_1 = \left(\frac{\alpha + 2}{\sqrt{2\alpha}} - \frac{\alpha}{\sqrt{2\alpha + 2}} + \frac{2}{\alpha - \sqrt{2\alpha}}\right) \cdot \frac{\sqrt{\alpha} - \sqrt{2}}{\alpha + 2}, \quad z_2 = \frac{1}{\sqrt{\alpha + \sqrt{2}}}.$$

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

$$12. z_1 = 2\sin^2(3\pi - 2\alpha)\cos^2(5\pi + 2\alpha), \quad z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right).$$

$$13. z_1 = \cos \alpha + \sin \alpha + \cos 3\alpha + \sin 3\alpha, \quad z_2 = 2\sqrt{2} \cos \alpha \cdot \sin\left(\frac{\pi}{4} + 2\alpha\right).$$

$$14. z_1 = 1 - \frac{1}{4}\sin^2 2\alpha + \cos 2\alpha, \quad z_2 = \cos^2 \alpha + \cos^4 \alpha$$

$$15. z_1 = (\cos \alpha - \cos \beta)^2 - (\sin \alpha - \sin \beta)^2, \quad z_2 = -4\sin^2 \frac{\alpha - \beta}{2} \cdot \cos(\alpha + \beta).$$

1.5.3. Третий уровень сложности

Составить программу для расчета каких-нибудь сложных геометрических или физических задач с проверкой исключительных ситуаций: деление на ноль, вычисление корня из отрицательного числа, выход значений аргументов используемых функций за допустимые пределы. При вводе данных использовать проверку на ввод не числовых данных.

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

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

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

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

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

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

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

При $x = 0.4 \times 10^4$, $y = -0.875$,
 $z = -0.475 \times 10^{-3} \rightarrow w = 1.9873$.

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

При $x = -15.246$, $y = 4.642 \times 10^{-2}$,
 $z = 20.001 \times 10^2 \rightarrow \alpha = -182.036$.

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

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

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

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

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

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

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

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

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

При $x = 3.981 \times 10^{-2}$,
 $y = -1.625 \times 10^3$, $z = 0.512 \rightarrow a = 1.2619$.

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

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

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

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

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

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

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

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

$$15. h = \frac{x^{y+1} + e^{y-1}}{1+x|y-\operatorname{tg} z|} (1+|y-x|) + \frac{|y-x|^2}{2} - \frac{|y-x|^3}{3}.$$

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

$$16. w = \sqrt[3]{x^6 + \ln^2 y} + \frac{e^{|x-y|} |x-y|^{x+y}}{\operatorname{arctg}(x) + \operatorname{arctg}(z)}.$$

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

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

Операции сравнения: < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), != (не равно), == (равно). Пары символов разделять нельзя.

Операции сравнения бинарные, их общий вид:

Операнд 1 **знак операции** *Операнд 2*

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

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

! (отрицание или логическое НЕ – унарная операция), && (конъюнкция или логическое И), || (дизъюнкция или логическое ИЛИ).

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

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 const_1: операторы; break;  
    ...  
    case const_N: операторы; break;  
    default: операторы;  
}
```

Вычисляется *выражение* и проверяется, совпадает ли его результат со значением одной из констант. При совпадении выполняются операторы этого *case*. Значениями *const_1, ..., const_N* могут быть только целые или символьные константы. Конструкция *default* выполняется, если результат выражения не совпал ни с одной из констант, и может отсутствовать. Оператор *break* выполняет досрочный выход из *switch* (после выполнения 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) визуально отражается на форме. Кнопки-переключатели бывают двух типов: *TCheckBox* и *TRadioGroup*.

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

Компонента *Radiogroup* создает группу кнопок – зависимых переключателей. При нажатии одной из кнопок группы все остальные кнопки отключают-

ся. В программу передается номер включенной кнопки (0,1,2,...), 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 $tg(x)$, после чего определить максимальное либо из u , y , z , либо из их модулей – $|u|$, $|y|$, $|z|$.


2.3.1. Реализация примера оконного приложения

Создать форму, представленную на рис. 2.1, скорректировать текст надписей, положение окон *Edit* и написать соответствующую программу.

Компонента *CheckBox*

В меню *Standard* выберите пиктограмму , поместите ее в нужное место формы, измените заголовок *Caption* на «max модулей», после чего в тексте программы 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 ($\cos(x)$, $\sin(x)$, $tg(x)$), нажмите *OK*. После этого внутри компоненты появятся три кнопки-переключателя с введенными надписями.

В тексте *Unit1.h* появилась переменная *RadioGroup1* типа *TRadioGroup*. Теперь при нажатии одной из кнопок группы в переменной целого типа *RadioGroup1->ItemIndex* будет находиться номер выбранной строчки (отсчитывается от 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.

Текст программы может быть следующим:

```
// Проверка номера нажатой кнопки и выбор 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;
    }
// Проверка состояния кнопки CheckBox1
    if (CheckBox1->Checked){
        u = fabs(u);
        y = fabs(y);
        z = fabs(z);
    }
// Нахождение максимального из трех чисел
    if (u>y) ma=u;
        else ma=y;
    if (z>ma) ma=z;
    if (CheckBox1->Checked)
        Memo1->Lines->Add("Max модулей=
                                "+FloatToStrF(ma,ffFixed,8,6));
    else Memo1->Lines->Add("Max= "+FloatToStrF(ma,ffFixed,8,6));
}

```

2.3.2. Пример написания программы консольного приложения

Текст программы может иметь вид

```

//-----
switch(kod){
    case 0:    u=cos(x);    cout<<" Cos(x)= "<<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;
}
cout<<"Max ABS - 1, Else - Max";
cin>>kod;
if (kod == 1){
    u = fabs(u);
    y = fabs(y);
    z = fabs(z);
}
if (u>y) ma=u;
    else ma=y;
if (z>ma) ma=z;
if (kod==1) cout<<"Max ABS = "<<ma;
    else cout<<"Max = "<<ma;

```

2.4. Индивидуальные задания

2.4.1. Первый уровень сложности

Составить программу для определения значения функции. Предусмотреть вывод сообщения о том, по какой ветви происходило вычисление значения аргумента функции x .

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

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

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

$$4. y = \cos^3 x^2 + \sin^2 x^3, \quad \text{где } x = \begin{cases} z^3 + 0,2; & \text{при } z < 1 \\ z + \ln z, & \text{при } z \geq 1 \end{cases}$$

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

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

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

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

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

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

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

$$12. y = \sin^3(x^2 - 1) + \ln|x| + e^x, \quad \text{где } x = \begin{cases} z^2 + 5, & \text{при } z \leq 0 \\ \frac{1}{\sqrt{z-1}}, & \text{при } z > 0 \end{cases}$$

$$13. y = \sin(n \cdot x) + \cos(k \cdot x) + \ln(m \cdot x), \quad \text{где } x = \begin{cases} e^z + z, & \text{при } z > 1 \\ z^2 + 1, & \text{при } z \leq 1 \end{cases}$$

$$14. y = \cos 5x + \sin \frac{1}{5}x + e^x, \quad \text{где } x = \begin{cases} \sqrt{z}, & \text{при } z > 0 \\ (3z^3 - z) - 5, & \text{при } z \leq 0 \end{cases}$$

$$15. y = x(\sin x + e^{-(x+3)}), \quad \text{где } x = \begin{cases} -3z, & \text{при } z > 0 \\ z^2, & \text{при } z \leq 0 \end{cases}$$

2.4.2. Второй уровень сложности

Вычислить значение y в зависимости от выбранной функции $\phi(x)$, аргумент которой определяется из поставленного условия. Возможные значения функции $\phi(x)$: $2x$, x^2 , $x/3$. Предусмотреть вывод сообщений, показывающих, при каком условии и с какой функцией производились вычисления y .

В программе организовать проверку исключительных ситуаций: деление на ноль, вычисление корня из отрицательного числа, выход значений аргументов используемых функций за допустимые пределы. При вводе данных использовать проверку на ввод не числовых данных.

$$1. y = a \ln(1 + x^{1/5}) + \cos^2[\phi(x) + 1], \quad \text{где } x = \begin{cases} z^2, & z < 1 \\ z + 1, & z \geq 1 \end{cases}$$

$$2. y = \frac{2a\phi(x) + b \cos \sqrt{|x|}}{x^2 + 5}, \quad \text{где } x = \begin{cases} 2 + z, & z < 1 \\ \sin^2 z, & z \geq 1 \end{cases}$$

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

$$4. y = 2a \cos^3 x^2 + \sin^2 x^3 - b\phi(x), \quad \text{где } x = \begin{cases} z^3 + 0,2, & z < 1 \\ z + \ln z, & z \geq 1 \end{cases}$$

$$5. y = a\phi(x) - \ln(x + 2,5) + b(e^x - e^{-x}), \quad \text{где } x = \begin{cases} -z/3, & z < -1 \\ |z|, & z \geq -1 \end{cases}$$

$$6. y = \frac{2}{3} a \sin^2 x - \frac{3b}{4} \cos^2 \phi(x),$$

$$\text{где } x = \begin{cases} z, & z < 0 \\ \sin z, & z \geq 0 \end{cases}$$

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

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

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

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

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

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

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

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

$$11. y = \frac{2,5a \cdot e^{-3x} - 4bx^2}{\ln |x| + \phi(x)},$$

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

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

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

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

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

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

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

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

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

2.4.3. Третий уровень сложности

Составить программу нахождения требуемого значения с исходными данными x, y, z . Нахождение минимального и максимального из перечисленных в скобках значений элементов (\min, \max) организовать в виде отдельной функции.

В программе организовать проверку исключительных ситуаций: деление на ноль, вычисление корня из отрицательного числа, выход значений аргументов используемых функций за допустимые пределы. При вводе данных использовать проверку на ввод не числовых данных.

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

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

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

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

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

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

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

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

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

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

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

$$12. F = \begin{cases} \min(x^2, y^2) + a, & \text{при } a < 0 \\ \max(y, x, a), & \text{при } a = 0 \\ |x - y| + y(x + \sqrt{a^3}), & \text{при } a > 0 \end{cases}$$

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

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

$$15. F = \begin{cases} \min(0, 9y, e^{2x-3}), & \text{при } x \leq 0 \\ \frac{2\cos\left(x - \frac{\pi}{6}\right) + \sqrt[3]{y}}{5 - 2x}, & \text{при } x \geq 0, y > 0 \\ \max(\sin^2 y, \cos^2 y), & \text{иначе} \end{cases}$$

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* = 0 (ложно) 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 выполнится *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 (*выражение 1*; *выражение 2*; *выражение 3*)

код цикла

где *выражение 1* – начальное значение параметра цикла; *выражение 2* – проверка условия на продолжение цикла; *выражение 3* – изменение (коррекция) параметра цикла.

Вначале вычисляется *выражение 1*, затем проверяется *выражение 2*, если оно – истинно, то выполняется код цикла, затем производится коррекция в *выражении 3*, и так до тех пор, пока *выражение 2* 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, в каждой строке которого указаны имя файла, номер строки и характер ошибки.

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

Ошибки *семантические* 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 ноль 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 среды.

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 заданном n .

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

Вначале 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 последовательности 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 решения, может выглядеть следующим образом:

```
for(x = a; x<=b; x+=h) {
    r = s = 1; //начальные значение
    for(k = 1; k<=n; k++) {
        r=-r*x/k; //расчет члена ряда через рекурренту
        s+=r; //суммирование всех слагаемых
    }
}
```

3.3. Индивидуальные задания

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. Таблица должна содержать следующие столбцы: порядковый номер, значение аргумента x и значение функции.

Определить максимальное и минимальное значения функции.

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. $Y(x) = \frac{x \sin(\frac{\pi}{4})}{1 - 2x \cos \frac{\pi}{4} + x^2},$ $a=-0,5; b=2,5; h=0,2.$
4. $Y(x) = (1 - \frac{x^2}{4}) \cos x - \frac{x}{2} \sin x,$ $a=-0,9; b=2,7; h=0,3.$
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. $Y(x) = (\frac{x^2}{4} + \frac{x}{2} - 3) \cdot e^{\frac{x}{2}},$ $a=-1,9; b=2,7; h=0,3.$

- | | | |
|-----|---|--|
| 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. | $Y(x) = (1 - \frac{x^2}{4}) \cos x - \frac{x}{2} \sin x,$ | $a = -0,9; b = 2,7; h = 0,3.$ |
| 14. | $Y(x) = \frac{1}{x^2 - x + 1},$ | $a = -0,1; b = 2; h = 0,1.$ |
| 15. | $Y(x) = \frac{\sin x \cos x}{\sqrt{x}},$ | $a = \pi; b = 2\pi; h = \pi/15.$ |

3.2. 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-четырёх позициях после десятичной точки.

Работу программы проверить для $a = 0,1; b = 1,0; h = 0,1$; значение параметра n выбрать в зависимости от задания.

- | | | |
|----|---|--|
| 1. | $S(x) = \sum_{k=0}^n (-1)^k \frac{x^{2k+1}}{(2k+1)!},$ | $Y(x) = \sin(x).$ |
| 2. | $S(x) = \sum_{k=1}^n (-1)^{k+1} \frac{x^{2k}}{2k(2k-1)},$ | $Y(x) = x \cdot \operatorname{arctg}(x) - \ln \sqrt{1+x^2}.$ |
| 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. | $S(x) = \sum_{k=0}^n \frac{2k+1}{k!} x^{2k},$ | $Y(x) = (1 + 2x^2)e^{x^2}.$ |

$$7. S(x) = \sum_{k=1}^n \frac{x^k \cos(k\pi/3)}{k}, \quad Y(x) = -\frac{1}{2} \ln(1 - 2x \cos \frac{\pi}{3} + x^2).$$

$$8. S(x) = \sum_{k=0}^n \frac{(2x)^k}{k!}, \quad Y(x) = e^{2x}.$$

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

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

$$11. S(x) = \sum_{k=0}^n \frac{k^2 + 1}{k!} (x/2)^k, \quad Y(x) = (x^2/4 + x/2 + 1)e^{x/2}.$$

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

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

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

$$15. S(x) = \sum_{k=1}^n (-1)^{k+1} \frac{x^{2k}}{2k(2k-1)}, \quad Y(x) = -\ln \sqrt{1+x^2} + x \operatorname{arctg}(x).$$

3.3. Третий уровень сложности

Составить программу по заданию сложности 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) ε , 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 – конечная последовательность данных одного типа. Массив – объект сложного типа, каждый элемент которого определяется именем (*ID*) и целочисленным значением индекса (номера), по которому к элементу массива производится доступ. Рассмотрим одномерные массивы.

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

В программе одномерный массив декларируется следующим образом:

```
тип ID массива [размер];
```

где *размер* – указывает количество элементов в массиве. Размер массива может задаваться константой или константным выражением. Для использования массивов переменного размера существует отдельный механизм – динамическое выделение памяти.

Примеры декларации массивов:

```
int a[5];  
double b[4] = {1.5, 2.5, 3.75};
```

в целочисленном массиве *a* первый элемент *a*[0], второй – *a*[1], ..., пятый – *a*[4]. Для массива *b*, состоящего из действительных чисел, выполнена инициализация, причем элементы массива получают следующие значения: *b*[0]=1.5, *b*[1]=2.5, *b*[2]=3.75, *b*[3]=0.

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

Примеры описания массивов:

```
const Nmax=10;           – задание максимального значения;  
typedef double mas1[Nmax*2]; – описание типа одномерного массива;  
  
mas1 a;                 – декларация массива a типа mas1;  
int ss[10];             – массив из десяти целых чисел.
```

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

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 информации 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.

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 заполнения массива использовать функцию *random(kod)* – генератор случайных равномерно распределенных целых чисел от 0 до $(int)kod$.

Значение *N* вводить из *Edit*, значения массива *A* – из компоненты *StringGrid*. Результат вывести в компоненту *StringGrid*.

Панель диалога и результаты выполнения программы приведена на рисунке 4.1.

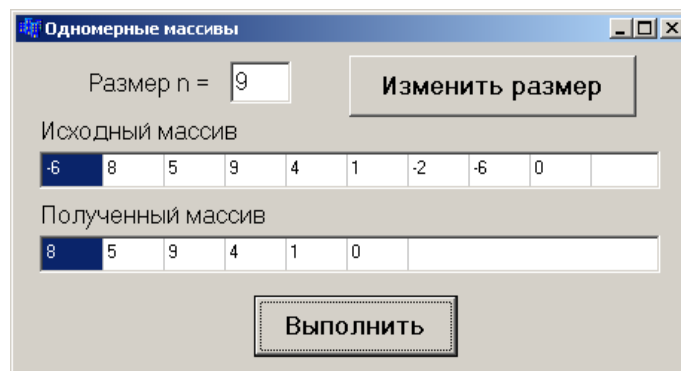



Рисунок 4.1

Настройка компоненты *StringGrid*

На закладке *Additional* выберите пиктограмму , установите компоненты *StringGrid1* и *StringGrid2* и отрегулируйте их размеры. В инспекторе объектов для обоих компонент установите значения *ColCount* равными 2, *RowCount* равными 1, т.е. по два столбца и одной строке, а значения *FixedCols* и *FixedRows* равными 0. Значение ширины клетки столбца *DefaultColWidth* равным 40.

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

Текст функций-обработчиков может иметь следующий вид:

...

```

        int n = 4;
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    randomize();           // Изменение начального адреса для
random()
    Edit1->Text=IntToStr(n);
    StringGrid1->ColCount=n;
    for(int i=0; i<n;i++)   // Заполнение массива А случайными чис-
лами
        StringGrid1->Cells[i][0] = IntToStr(random(21)-10);
    Label3->Hide();       // Скрыть компоненту
    StringGrid2->Hide();
}
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    n=StrToInt(Edit1->Text);
    if(n>10){
        ShowMessage("Максимальное количество 10!");
        n=10;
        Edit1->Text = "10";
    }
    StringGrid1->ColCount=n;
    for(int i=0; i<n;i++)
        StringGrid1->Cells[i][0]=IntToStr(random(21)-10);
    Label3->Hide();
    StringGrid2->Hide();
}
//-----
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    int i, kol = 0, a[10];   // Декларация одномерного массива
//Заполнение массива А элементами из таблицы StringGrid1
    for(i=0; i<n;i++)
        a[i]=StrToInt(StringGrid1->Cells[i][0]);
//Удаление отрицательных элементов из массива А
    for(i=0; i<n;i++)
        if(a[i]>=0) a[kol++] = a[i];
    StringGrid2->ColCount = kol;
    StringGrid2->Show();    // Показать компоненту
    Label3->Show();
//Вывод результата в таблицу 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 значений (n – не больше 20), вычислить:

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), предусмотреть ввод значений массива с клавиатуры и используя рандомное заполнение; предусмотреть вывод сообщений 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 нового массива.

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.

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 нового массива.

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 нового массива.

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 не числовых 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.

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 по неубыванию.

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, которые расположены 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 C может проводиться по имени, как мы до сих пор делали, и по **указателю** (косвенная адресация).

Указатель – это переменная, которая может содержать 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 * ID 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 типа 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 и сравнения, причем выполняются они в единицах того типа, на который установлен указатель.

Операции сложения, вычитания и сравнения (больше/меньше) имеют смысл только для последовательно расположенных данных – массивов. Операции сравнения «**==**» и «**!=**» 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.

Пусть объявлены – массив из 10 элементов и указатель типа *double*:

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

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

$\&a[0] \quad \leftrightarrow \quad \&>(*p) \quad \leftrightarrow \quad p$

Декларация многомерного массива:

тип ID[размер 1][размер 2]...[размер N];

причем 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 соответствующему элементу присваивается значение 0.

Указатели на указатели

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

Если рассматривать предыдущий массив (*int a[2][3]*;) как массив двух массивов размерностью по три элемента каждый, то обращение 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 – *ID* указателя на указатель.

Динамическое размещение данных

Для создания массивов с переменной размерностью используется динамическое размещение данных, декларируемых указателями.

Для работы с динамической памятью используются стандартные функции библиотеки *alloc.h*:

*void *malloc(size)* и *void *calloc(n, size)* – выделяют блок памяти размером *size* и *n×size* байт соответственно; возвращают указатель на выделенную область, при ошибке – значение *NULL*;

void free(bf); – освобождает ранее выделенную память с адресом *bf*.

Другим, более предпочтительным подходом к динамическому распределению памяти является использование операций языка C++ *new* и *delete*.

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

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

```
double *a;
```

```
a = new double[n]; // Захват памяти для n элементов
```

```
delete []a; // Освобождение памяти
```

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

```
int i, n, m; // n, m – размеры массива
double **a;
a = new double *[n]; // Захват памяти под указатели
for(i=0; i<n; i++) a[i] = new double [m]; // и под элементы

for(i=0; i<n; i++) delete []a[i]; // Освобождение памяти
delete []a;
```

Для современных компиляторов (версий старше «6») для освобождения памяти достаточно записать только `delete []a;`

5.2. Пример выполнения задания

Найти сумму положительных элементов квадратной матрицы $N \times N$.

Пример создания оконного приложения

Значение N вводить из *Edit*, A – из компоненты *StringGrid*. Результат вывести в компоненту *Edit*.

Настройка компонент StringGrid

Для компоненты *StringGrid1* значения *ColCount* и *RowCount* установите равными, например, 3 – три столбца и три строки, а *FixedCols* и *FixedRows* – 1.

В свойстве *Options* строку *goEditing* для компонент установите в положение *true*.

Для изменения размера n используется функция-обработчик *EditChange*, полученная двойным щелчком по компоненте *Edit*.

Текст программы может иметь следующий вид:

```
//----- Глобальные переменные -----
int n = 3;
double **a; // Декларации указателей
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    Edit1->Text=IntToStr(n);
    StringGrid1->ColCount = n+1;    StringGrid1->RowCount = n+1;
    // Ввод в левую верхнюю ячейку таблицы названия массивов
    StringGrid1->Cells[0][0] = "Матрица A";
    for(int i=1; i<=n;i++){
        StringGrid1->Cells[0][i]="i="+IntToStr(i);
        StringGrid1->Cells[i][0]="j="+IntToStr(i);
    }
}
//-----
void __fastcall TForm1::Edit1Change(TObject *Sender)
```

```

{
    int i;
    n=StrToInt(Edit1->Text);
    StringGrid1->ColCount = n+1;    StringGrid1->RowCount = n+1;
    for(i=1; i<=n;i++){
        StringGrid1->Cells[0][i]="i="+IntToStr(i);
        StringGrid1->Cells[i][0]="j="+IntToStr(i);
    }
}
//-----
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    delete []a;
    delete []b;
    ShowMessage("Память освобождена!");
    Close();
}
//-----Часть кода, для реализации поставленной задачи-----
int sum=0;
for ( i=0; i<n; i++)
for ( i=0; i<n; i++)
    for ( j=0; j<n; j++)
        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 V , 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 V , 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 V , 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 двумерного массива решить поставленную задачу, алгоритм которой реализовать в виде отдельной функции. При вводе исходных данных выполнить проверку ввода не числовых 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 0 и 1. Повернуть элементы массива на 90° по часовой стрелке.

5. Элемент матрицы назовем седловой точкой, 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 седловых точек.

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 соответству-

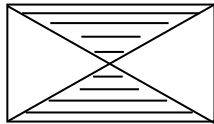
ющий ему элемент исходной матрицы больше элемента, расположенного на главной диагонали, и равен 0 в противном случае.

12. Дана действительная квадратная матрица порядка N . Найти сумму и произведение элементов, расположенных в заштрихованной части матрицы, см. рисунок «а».

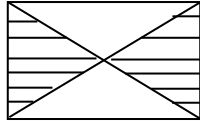
13. То же, см. рисунок «б».

14. То же, см. рисунок «в».

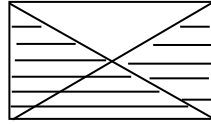
15. Дана действительная квадратная матрица порядка N . Найти наименьшее и наибольшее из значений элементов, расположенных в заштрихованной части матрицы, см. рисунок «а».



а



б



в

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 отведется 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", S2[] = "12345";
```

в последнем случае размер строки устанавливается по фактическому количеству.

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*), которая 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 нуль-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*) ***atol***(*S*); действительное – (*double*) ***atof***(*S*); при возникновении ошибки данные функции возвращают значение 0;
- 9) преобразование числа *V* в строку *S*: целое – ***itoa***(*int V*, *char S*, *int kod*); длинное целое – ***ltoa***(*long V*, *char S*, *int kod*); значение параметра *kod* определяет

выбор системы счисления для преобразования и находится в диапазоне $2 \leq kod \leq 36$, для отрицательных чисел $kod = 10$.

6.2. Создание оконного приложения

Tun AnsiString

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

Многие другие операции над строками осуществляются с помощью методов типа *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:

int Length() – определение длины строки

char c_str() – преобразование строки *AnsiString* в массив символов;

Insert(String S2, int i) – вставляет строку *S2* в середину строки, начиная с *i*-ой позиции;

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

Пример: Заменить в строке *S* символы с 3-го по 7-ой на многоточие.

```
S.Delete(3,5);
```

```
S.Insert("...",3);
```

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

Пример: Вывести все символы строки, кроме первого:

```
Edit1->Text=s.SubString(2, s.Length()-1);
```

int Pos(String S2) – ищет первое вхождение подстроки *S2* в данной строке. Результат - номер позиции, начиная с которой в строке содержится *S2*, либо 0, если *S2* в ней не содержится.

Пример:

```
S1="Ababab", S2="ab";
```

```
i=S1.Pos(S2); // i=3
```

Компонента ListBox

Данная компонента представляет собой список, элементы которого выбираются при помощи клавиатуры или мыши. Список элементов задается свой-

ством *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 изображения на поверхности, определяемое свойством *Glyph*. Кроме того, свойство ***Kind*** 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 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 отпуске ранее нажатой клавиши;

OnClick – возникает при нажатии кнопки мыши в области компоненты, а *OnDoubleClick* – 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 рисунке 6.1.

В тексте программы приведем только функции-обработчики:

```
//-----
void __fastcall TForm1::FormActivate(TObject *Sender)
{
    Form1->ComboBox1->SetFocus();           // Передача фокуса ComboBox1
}
//-----
void __fastcall TForm1::ComboBox1KeyDown(TObject *Sender, WORD &Key, TShiftState
Shift)
{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, nst;
nst = ComboBox1->ItemIndex;                // Определение номера строки
String st = ComboBox1->Items->Strings[nst]; // Запись выбранной строки st
    if (st[1] != ' ') n = 1;                // Здесь и ниже ' ' – пробел
        else n = 0;
    for(i=1; i<st.Length(); i++)           // Просмотр символов строки st
        if(st[i] == ' ' && st[i+1] != ' ') n++;
    Edit1->Text = IntToStr(n);              // Вывод числа слов в Edit1
```

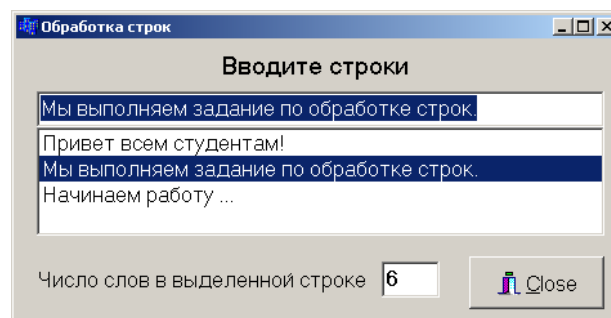


Рисунок 6.1

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 одномерный массив.

2. Для заданного целого числа N определить цифру a , наиболее часто встречающуюся в числе. Сформировать одномерный массив 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 одномерный массив.

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 - A... Z.

15. Ввести строку символов. Определить длину введенной строки L , и если длина $L > 15$, то удаляются все буквы - a...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 максимальной длины и номер позиции строки с которой оно начинается.

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

16. В строке символов посчитать 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. Введите с клавиатуры строку символов, после чего подсчитайте и выведите на экран ее длину (т.е. число символов, входящих в строку), а также подсчитайте и выведите число вхождений буквы «а» в строку.

2. Введите с клавиатуры строку символов, после чего подсчитайте и выведите на экран число пробелов, а также число цифр, содержащихся в ней.

3. Введите с клавиатуры две строки символов. Допишите вторую строку в конец первой, после чего повторите эту операцию еще раз. Далее выведите первую строку на экран.

4. Введите с клавиатуры две строки символов, после чего выполните по-символьное их сравнение и выведите на экран результат сравнения в виде числа (результат получается вычитанием первых несовпадающих кодов символов из первой и второй строк).

5. Введите с клавиатуры строку символов, после чего подсчитайте и выведите на экран число слов в ней, а затем и каждое отдельное слово.

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

8. Введите с клавиатуры строку символов, после чего выведите 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. Введите с клавиатуры строку символов, после чего запишите ее содержимое в другую строку сначала в обратном порядке, а затем в прямом порядке. Далее выведите вторую строку на экран.

10. Введите с клавиатуры две строки символов, после чего вставьте вторую строку в середину первой. Далее выведите первую строку на экран.

11. Введите с клавиатуры строку символов, после чего замените все буквы "а" (латиница) символом звездочка (*), а все цифры – символом решетка (#). Выведите результирующую строку на экран.

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

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

14. Введите с клавиатуры строку символов. Замените первую букву в каждом втором слове на вопросительный знак и выведите результирующую строку на экран.

15. Введите с клавиатуры строку символов. Перепишите из нее первое, третье и пятое слова (через пробел) в другую строку, после чего последнюю выведите 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 разных типов. Структурный тип данных определяется описанием *шаблона*:

```
struct Person {  
    char Fio[30];  
    double sball;  
};
```

Объявление переменных созданного структурного типа:

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

Обращение к элементам структур производится посредством:

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

ID_структуры . ID_поля или (**указатель*) . *ID_поля*

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

указатель -> ID_поля или *&(ID_структуры) . ID_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 байт. При этом последовательность записи определяется самой программой.

Работа с файлами

Файл – это набор данных, размещенный на внешнем носителе и рассматриваемый в процессе обработки и пересылке 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(“строка 1” , “строка 2”);

в строке 1 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 строке 2 указывается код открытия файла:

w – для записи, если файла с заданным именем нет, то он будет создан, если же такой файл существует, то перед открытием прежняя информация уничтожается;

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

a – для добавления новой информации в конец;

r+, *w+* – возможны чтение и запись информации;

a+ – то же, что и для *a*, только запись можно выполнять в любое место файла, доступно и чтение файла.

По умолчанию файл открывается в текстовом режиме (*t*), указав *b* – файл открывается в двоичном режиме.

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”, “r+t” ) ) ) {
    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. лаб.раб.№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

Компоненты OpenFileDialog и SaveDialog

Компоненты *OpenDialog* и *SaveDialog* находятся на странице *Dialogs*. Все компоненты этой страницы невизуальны, 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 – используется свойство *Title*.

7.3. Пример выполнения задания

Написать программу обработки файла, содержащего информацию о рейтинге студентов. Каждая запись должна содержать Ф.И.О. и полученный балл рейтинга. Вывести информацию, отсортированную в порядке увеличения рейтинга. 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* оконного приложения

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 рисунке 7.1. Формат **.dat* означает, что будут видны все файлы с расширением *dat*, а формат **.** – будут видны все файлы (с любыми именами и расширениями).

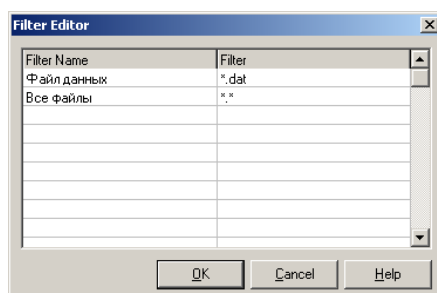


Рисунок 7.1

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 захотим заменить прежнюю информацию на новую, в окне *Memo1* отображается путь и имя созданного файла.

Заполнив оба поля информацией, нажимаем кнопку «**Добавить**», после чего введенная информация отображается в окне *Memo1*.

Для работы с уже созданным файлом нажимаем кнопку «**Открыть**» – в *Memo1* выводится содержимое всего файла, после чего можно добавлять новые данные в конец этого файла, не уничтожая предыдущие.

При нажатии кнопки «**Сортировать**» в *Memo1* выводятся записи, сортированные по возрастанию рейтинга.

При нажатии кнопки «**Сохранить результаты**» создается текстовый файл, в котором сохранится информация, выведенная в *Memo1*. Этот файл можно просмотреть в любом текстовым редакторе (блокноте, *Word*).

В текст программы включена пользовательская функция `void Out(TZap, TMemo*)`; – для вывода в `Memo1` одной записи.

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

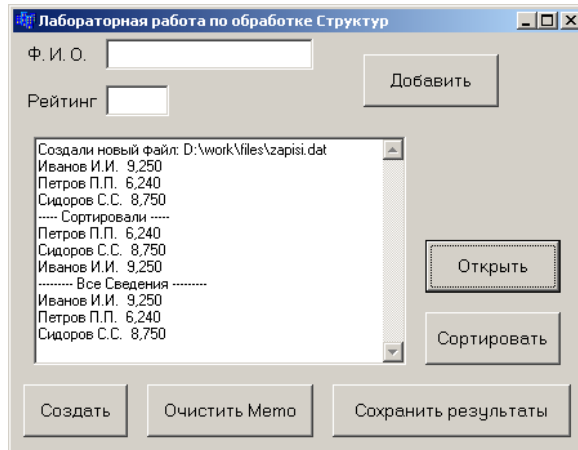


Рисунок 7.2

Текст программы может иметь следующий вид:

```

...
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    OpenFileDialog1->Title="Создать новый файл";
    if (OpenDialog1->Execute()){
        File_Zap = OpenFileDialog1->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);
}
//----- Добавить-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    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);
    Edit1->Text=""; Edit2->Text="";
    fclose(Fz);
}

```

```

//----- Сортировать -----
void __fastcall TForm1::Button4Click(TObject *Sender)
{
    TZap st, *mas_Z;
    Fz = fopen(File_Zap.c_str(),"rb");
    int D_f = fileno(Fz);           // Находим дескриптор файла
    int len = filelength(D_f);     // Находим размер файла
    int i, j, kol;
    kol = len/size;                //Количество записей в файле
    mas_Z = new TZap[kol];

// Считываем записи из файла в динамический массив
    for (i=0; i < kol; i++)
        fread((mas_Z+i), size, 1, Fz);
    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++)
        Out(mas_Z[i], Memo1);
    delete []mas_Z;
}

//----- Сохранить -----
void __fastcall TForm1::Button5Click(TObject *Sender)
{
    SaveDialog1->Title="Сохранить файл результатов";
    if (SaveDialog1->Execute()) {
        AnsiString FileNameRez = SaveDialog1->FileName;
        Memo1->Lines->SaveToFile(FileNameRez);
    }
}

//----- Открыть -----
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    OpenFileDialog1->Title="Открыть файл";
    if (OpenDialog1->Execute()) {
        File_Zap = OpenFileDialog1->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);
}
//----- Функция вывода 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 типа запись, содержащую следующие пункты меню: «Создание», «Просмотр», «Коррекция» (добавление новых данных или редактирование старых), «Решение индивидуального задания».

Каждая запись должна содержать следующую информацию о студентах:

- фамилия и инициалы;
- год рождения;
- номер группы;
- оценки за семестр: по физике, математике, информатике, химии;
- 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. Распечатать анкетные данные студентов, сдавших сессию на 8, 9 и 10.
2. Распечатать анкетные данные студентов-отличников, фамилии которых начинаются с интересующей вас буквы.
3. Распечатать анкетные данные студентов-отличников из интересующей вас группы.
4. Распечатать анкетные данные студентов, фамилии которых начинаются с буквы *A*, и сдавших математику на 8 или 9.
5. Распечатать анкетные данные студентов, имеющих оценки 4 или 5 по физике и оценку больше 8 по остальным предметам.
6. Распечатать анкетные данные студентов интересующей вас группы. Фамилии студентов начинаются с букв *B*, *Г* и *Д*.
7. Распечатать анкетные данные студентов, не имеющих оценок 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. Распечатать анкетные данные студентов интересующей вас группы, имеющих неудовлетворительную оценку (меньше 4).

11. Распечатать анкетные данные студентов интересующей вас группы, имеющих оценку 9 или 10 по информатике.

12. Распечатать анкетные данные студентов, имеющих оценки 7 или 8 по физике и оценки 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. Распечатать анкетные данные студентов-отличников интересующей вас группы.

15. Распечатать анкетные данные студентов интересующей вас группы, имеющих средний балл выше введенного с клавиатуры.

16. Распечатать анкетные данные студентов интересующей вас группы, имеющих оценку 8 по физике и оценку 9 по высшей математике.

7.4.2. Второй уровень сложности

Написать программу предыдущего варианта, создав из предложенных анкетных данных динамический массив введенной с клавиатуры размерности. Полученные данные упорядочить: для символьных данных – по алфавиту (выбрав нужное поле), для числовых данных – по возрастанию (убыванию).

7.4.3. Третий уровень сложности

Задачи шифровки. Составить программу, которая вводит строку с клавиатуры; признак окончания ввода – нажатие клавиши Enter, шифрует введенный текст в файл на диске по определенному 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 *a.txt*;
- считывание строки из файла и вывод на экран;
- шифровка текста;
- расшифровка.

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 – 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 из пары букв «си», «ли» и «ти» заменяются 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 ID_функции (список типов параметров);

а определение функции имеет следующую структуру:

```
тип_результата ID_функции (список параметров)
{
    код функции
    return результат;
}
```

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

Из функции можно передать только одно значение, при необходимости вернуть несколько – в списке параметров используют указатели.

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:

ID_функции(список аргументов);

где в качестве аргументов можно использовать константы, переменные, выражения (их значения перед вызовом функции будут компилятором определены). Аргументы списка вызова должны совпадать со списком параметров вызываемой функции по количеству и порядку следования параметров, а типы аргумен-

тов 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 переменной – это правила, которые устанавливают, какие данные доступны из текущего места программы, и определяют переменные двух типов: *глобальные* и *локальные*.

Глобальные переменные объявляются вне какой-либо функции и могут быть использованы в любом месте программы, но перед их первым использованием они должны быть объявлены и инициализированы. Область действия глобальных переменных – вся программа с момента их объявления. **При объявлении глобальные переменные обнуляются.**

Область действия **локальных** переменных – это блоки, где они объявлены. К локальным относятся и формальные параметры в заголовке функций пользователя с областью действия – кодом функции. При выходе из блока значения локальных переменных теряются.

В языке C каждая переменная принадлежит одному из четырех классов памяти – автоматической (*auto*), внешней (*extern*), статической (*static*) и регистровой (*register*). Тип памяти указывается перед спецификацией типа, например, `register int a;` или `static double b;` По умолчанию устанавливается класс *auto* и переменные размещаются в стеке.

Способы передачи параметров

Существует 3 способа передачи параметров в функцию: по значению, по ссылке и по указателю.

Передача параметров по значению выглядит следующим образом:

```
int factorial(int x);
```

В этом случае, в момент вызова функции *factorial* в *x* будет передана копия значения параметра *x*. Если в процессе выполнения операторов функции значение, поданное на вход, претерпит изменения, в точке вызова это не будет доступно.

В случае, если из функции необходимо получить более одного значения, или, если необходимо вернуть параметр, переданный в функцию, с изменениями, которые произошли в ходе выполнения функции, можно применить передачу параметров либо по ссылке, либо по указателю.

Передача параметров по ссылке выглядит следующим образом:

```
int factorial(int &x);
```

Здесь в качестве параметра в функцию передается адрес переменной, полученный с помощью операции «взять адрес».

Передача параметров по адресу выглядит следующим образом:

```
int factorial(int *x);
```

Указатель на функцию

Функции могут быть использованы в качестве формальных параметров, для этого используется указатель на функцию. Например, указатель *p* на функцию, возвращающую результат типа *double* и имеющую два параметра *double* и *int*, объявляется следующим образом:

```
double (*p)(double, int);
```

Используя операцию *typedef*, тип такого указателя может быть объявлен так

```
typedef double (*TFun)(double, int);
```

8.2. Пример выполнения задания

Написать программу вычисления выбранной функции, вид которой в свою очередь передается в качестве параметра в функцию вывода (*Out_Rez*).

Создание оконного приложения

Текст программы может иметь следующий вид:

```
...
#include "math.h"
//----- Декларация типа указателя на функции -----
typedef double (*TFun)(double);
//----- Декларации прототипов функций Пользователя -----
double fun1(double);
double fun2(double);
void Out_Rez(TFun, double, double, double, TMemo*);
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    Edit1->Text="0,1";    Edit2->Text="3";    Edit3->Text="0,3";
    Memo1->Clear();    Memo2->Clear();
    RadioGroup1->ItemIndex=0;
}
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    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; }
}
//----- Реализации функций Пользователя -----
double fun1( double r){
    return 2*exp(r*r); }
double fun2(double r) {
```

```

return pow(sin(r), 2); }
void Out_Rez (TFun f,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

По заданию лабораторной работы №3(первый уровень сложности) 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

По заданию лабораторной работы №3(второй уровень сложности) 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*).

8.3.3. 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 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. Напишите рекурсивную функцию, которая вычисляет длину строки.

6. Найти методом деления отрезка пополам минимум функции $f(x) = 7\sin^2(x)$ на отрезке $[2, 6]$ с заданной точностью ε (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 $y(n) = \frac{1}{n + \frac{1}{(n-1) + \frac{1}{(n-2) + \frac{1}{\dots + \frac{1}{1 + \frac{1}{2}}}}}}$.

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. Построение графиков функций

Цель работы: изучить некоторые возможности построения графиков функций с помощью компонент *Chart* и *Image*; научиться работать с графическими объектами; 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 двумерный 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 кнопкой мыши вызываем окно редактирования *EditingChart1* (рис. 9.1). Для создания *Series1* нажимаем кнопку *Add* на странице *Series*.

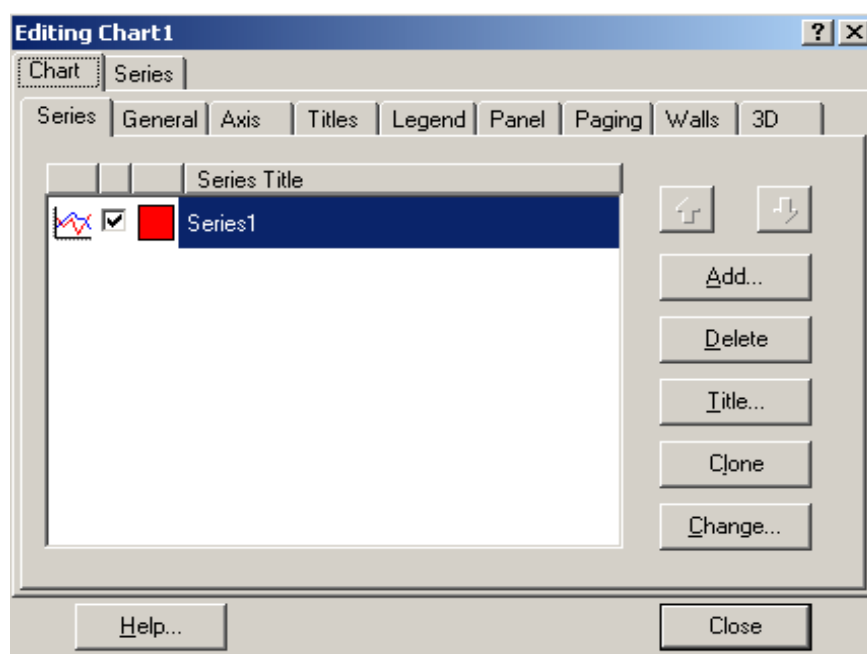


Рис. 9.1

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 пере-

ключатель *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(x_1, y_1, x_2, y_2) – чертит эллипс в охватывающем прямоугольнике (x_1, y_1), (x_2, y_2) 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(x_1, y_1, x_2, y_2) – вычерчивает и заполняет прямоугольник (x_1, y_1), (x_2, y_2). Для вычерчивания без заполнения используйте *FrameRect* или *Polyline*;

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

TextOut($x, y, \text{const AnsiString Text}$) – 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.

```
//-----
```

```

void Shar(int xx,int yy, TColor cc) // Вывод изображения машины
{
Form1->Canvas->Pen->Color=cc; // Установка цвета карандаша
Form1->Canvas->Brush->Color=cc; // Установка цвета кисти
TPoint pnt[7];
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);
Form1->Canvas->Polygon(pnt,6); // Кузов машины
Form1->Canvas->Ellipse(xx+10,yy+30,xx+30,yy+50); // Колесо
Form1->Canvas->Ellipse(xx+80,yy+30,xx+100,yy+50); // Колесо
} // Конец функции вывода изображения машины
//-----

```

```

void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
Canvas->Pen->Color = clBlack; // Установка цвета карандаша
Canvas->Brush->Color = clGreen; // Установка цвета кисти
Canvas->Rectangle(10,10,520,90); // Рисует прямоугольник
double hx=1; double h=0;
double x=10; int n=0;
while (x<410)
{
Shar(x,40,clYellow); // Рисование машины
Sleep(10); // Задержка
Shar(x,40,clGreen); // Стирание нарисованной машины
x+=hx;
n++; }}

```

```

void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
Chart1->Series[0]->Clear(); // Очистка графика
}
//-----

```

```

void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
Chart1->CopyToClipboardMetafile(True); // Передача графика в буфер
// обмена для последующей вставки в отчет
}
//-----

```


Приложение 1. Операции и основные математические функции

1. Операции приведены в порядке убывания приоритета, операции с разными приоритетами разделены чертой.

<i>Опера-ция</i>	<i>Краткое описание</i>	<i>Использование</i>	<i>Выполне-ние</i>
Первичные (унарные) операции			
.	Доступ к члену	<i>объект . член</i>	Слева направо
->	Доступ по указателю	<i>указатель -> член</i>	
[]	Индексирование	<i>переменная [выражение]</i>	
()	Вызов функции	<i>ID(список)</i>	
Унарные операции			
++	Постфиксный инкремент	<i>lvalue++</i>	Справа налево
--	Постфиксный декремент	<i>lvalue--</i>	
sizeof	Размер объекта (типа)	<i>sizeof(ID или тип)</i>	
++	Префиксный инкремент	<i>++lvalue</i>	
--	Префиксный декремент	<i>--lvalue</i>	
~	Побитовое НЕ	<i>~выражение</i>	
!	Логическое НЕ	<i>!выражение</i>	
- (+)	Унарный минус (плюс)	<i>- (+)выражение</i>	
*	Разадресация указателя	<i>*выражение</i>	
&	Адрес	<i>&выражение</i>	
()	Приведение типа	<i>(тип)выражение</i>	
Бинарные и тернарная операции			
*	Умножение	<i>выражение * выражение</i>	Слева направо
/	Деление	<i>выражение / выражение</i>	
%	Получение остатка	<i>выражение % выражение</i>	
+ (-)	Сложение (вычитание)	<i>выражение + (-) выражение</i>	
<<	Сдвиг влево	<i>выражение << выражение</i>	
>>	Сдвиг вправо	<i>выражение >> выражение</i>	
<	Меньше	<i>выражение < выражение</i>	
<=	Меньше или равно	<i>выражение <= выражение</i>	
>	Больше	<i>выражение > выражение</i>	
>=	Больше или равно	<i>выражение >= выражение</i>	
==	Равно	<i>выражение == выражение</i>	
!=	Не равно	<i>выражение != выражение</i>	
&	Побитовое И	<i>выражение & выражение</i>	
^	Побитовое исключ. ИЛИ	<i>выражение ^ выражение</i>	
	Побитовое ИЛИ	<i>выражение выражение</i>	
&&	Логическое И	<i>выражение && выражение</i>	
	Логическое ИЛИ	<i>выражение выражение</i>	

Опера- ция	Краткое описание	Использование	Выполне- ние
?:	Условная операция (<i>тернарная</i>)	<i>выражение ? выражение : выражение</i>	
= *= /= %= += - = <<= >>= &= = ^=	Присваивание Умножение с присваиванием Деление с присваиванием Остаток от деления с присв-м Сложение с присваиванием Вычитание с присваиванием Сдвиг влево с присваиванием Сдвиг вправо с присваиванием Поразрядное И с присваив-м Поразрядное ИЛИ с присв-м Поразрядное ИСКЛЮЧАЮ- ЩЕЕ ИЛИ с присваиванием	<i>lvalue = выражение lvalue *= выражение lvalue /= выражение lvalue %= выражение lvalue += выражение lvalue -= выражение lvalue <<= выражение lvalue >>= выражение lvalue &= выражение lvalue = выражение lvalue ^= выражение</i>	Справа налево
,	Последовательное вычисление	<i>выражение, выражение</i>	Слева направо

Стандартные 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 2π (рад) равно 360° (град).

Математиче- ская функция	ID функции	Математическая функция	ID функции
\sqrt{x}	sqrt(x)	arcsin(x)	asin(x)
$ x $	fabs(x)	arctg(x)	atan(x)
e^x	exp(x)	arctg(x/y)	atan2(x,y)
x^y	pow(x, y)	sh*(x)=1/2 (e ^x -e ^{-x})	sinh(x)
ln(x)	log(x)	ch(x)=1/2 (e ^x +e ^{-x})	cosh(x)
lg ₁₀ (x)	log10(x)	tgh(x)	tanh(x)
sin(x)	sin(x)	Остаток от деления x на y	fmod(x,y)
cos(x)	cos(x)	Округление к большему	ceil(x)
tg(x)	tan(x)	Округление к меньшему	floor(x)

* Синус гиперболический, а в следующей строке – косинус гиперболический.

Приложение 2. 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)
{
}
// Описание 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
{
```

Окончание прил. 2

```

// 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, – используя диалог
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. – Москва .: ДИАЛОГ-МИФИ, 2005.
2. Демидович, Е. М. Основы алгоритмизации и программирования. Язык СИ / Е. М. Демидович. – Санкт-Петербург : БХВ - Петербург, 2006.
3. Керниган, Б. Язык программирования СИ / Б. Керниган, Д. Ритчи. – Москва: Вильямс, 2006
4. Страуструп, Б. Программирование: принципы и практика использования С++, исправленное издание = Programming: Principles and Practice Using С++. — Москва.: «Вильямс», 2011.
5. Страуструп, Б. Язык программирования С++ / Б. Страуструп: спец.изд.: В 2 т. – Москва : Бином, 2011.
6. Архангельский, А. Я. Программирование в С++ *Builder* / А. Я. Архангельский: 7-е изд.. – Москва : Бином, 2010.
7. Шилд, Г. С++: базовый курс, 3-е издание = С++ from the Ground Up Third Edition. — Москва.: «Вильямс», 2012.

Учебное издание

**Беспалов Сергей Алексеевич,
Зайцева Ирина Евгеньевна
Кривоносова Татьяна Михайловна,
Рак Татьяна Александровна,
Смирнов Вячеслав Леонидович
Шестакович Вячеслав Павлович,
Шатилова Ольга Олеговна**

Основы алгоритмизации и программирования (язык C/ C++)

Лабораторный практикум по курсу
«Основы алгоритмизации и программирования»
для студентов 1-го курса БГУИР
В 2-х частях
Часть 1

Редактор **Е. Н. Батурчик**

Подписано в печать.
Печать ризографическая.
Уч. изд. л.

Формат 60x84 1/16
Гарнитура «Times»
Тираж 500 экз.

Бумага офсетная.
Усл. печ. л.
Заказ

Издатель и полиграфическое исполнение:
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектрони-
ки»
Лицензия ЛП №156 от 05.02.2001.
Лицензия ЛП №509 от 03.08.2001.
220013, Минск, П. Бровки, 6