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

С. В. Снисаренко, Н. А. Стасевич, Н. А. Капанов

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ УПРАВЛЕНИЯ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия для студентов, получающих образование по специальности 1-53 01 07 «Информационные технологии и управление в технических системах»

Минск БГУИР 2012

УДК 004.413:681.51(076.5)
ББК 32.973.26-018.2я73+32.965я73
С53

Р е ц е н з е н т ы :
доцент кафедры автоматизации технологических процессов и электротехники учреждения образования «Белорусский государственный технологический университет» И. Ф. Кузьмицкий;

заведующий кафедрой «Информационные системы и технологии» Белорусского национального технического университета, доктор технических наук А. А. Лобатый

Снисаренко, С. В.

С53 Технология разработки программного обеспечения систем управления. Лабораторный практикум : пособие / С. В. Снисаренко, Н. А. Стасевич, Н. А. Капанов. – Минск : БГУИР, 2012. – 76 с. : ил.

ISBN 978-985-488-869-9.

В пособии рассматриваются вопросы, относящиеся к использованию технологии объектно-ориентированного программирования и программирования под Windows на языке C++ при проектировании систем управления. Описание методологии построения и использования основных принципов объектно-ориентированного программирования и программирования под Windows сопровождается примерами.

Предназначено для студентов специальности 1-53 01 07 «Информационные технологии и управление в технических системах» всех форм обучения, выполняющих лабораторные работы по курсу «Технология разработки программного обеспечения систем управления».

УДК 004.413:681.51(076.5)
ББК 32.973.26-018.2я73+32.965я73

ISBN 978-985-488-869-9

СОДЕРЖАНИЕ

Введение

При использовании вычислительных средств в качестве устройств управления технологическими процессами или подвижными объектами системы управления необходимо рассматривать не только с точки зрения аппаратного, но и программного обеспечения. Именно с помощью программных средств устанавливаются правила решения задач управления, порядок формирования управляющей информации.

Данное пособие предназначено для начинающих изучение технологии объектно-ориентированного программирования и на ее основе освоения программирования под Windows для создания управляющих программ для аппаратно-технического комплекса автоматических систем и создания программного интерфейса автоматизированных систем управления на основе C++.

Лабораторный практикум содержит четырнадцать работ. Первые семь лабораторных работ раскрывают основные принципы и приемы программирования при создании управляющих программ для систем управления и посвящены освоению технологии объектно-ориентированного программирования, которое лежит в основе создания приложений под операционную систему Windows. Остальные семь лабораторных работ посвящены созданию приложений Windows, которые будут представлять собой программный интерфейс автоматических и автоматизированных систем управления. В каждой работе приведены краткие теоретические сведения, примеры реализации поставленных задач, варианты заданий и контрольные вопросы. Особое внимание уделено созданию программного интерфейса для автоматизированных информационных систем, ядром которых является реляционная база данных.

Технология объектно-ориентированного программирования выбрана в качестве базовой, так как она лежит в основе большинства пакетов проектирования и создания систем управления. Язык C++, в котором реализуются поставленные задачи, также является основой для многих языков программирования при создании программного интерфейса систем управления.

Лабораторная работа №1
Программирование алгоритмов с использованием динамических массивов

Цель работы – научиться использовать операции динамического выделения и освобождения памяти на примере работы с одномерными и двухмерными массивами, а также косвенное обращение к элементам массива.

Теоретические сведения

Объявление динамического массива

Массивы, создаваемые в динамической памяти, будем называть *динамическими* (размерность становится известна в процессе выполнения программы). При описании массива после имени в квадратных скобках задается количество его элементов (размерность), например int a[10]. Размерность массива может быть задана только константой или константным выражением.

При описании массив можно инициализировать, т. е. присвоить его элементам начальные значения, например:

```
int a[10] = {1, 1, 2, 2, 5, 100};
```

Если инициализирующих значений меньше, чем элементов в массиве, остаток массива обнуляется, если больше – лишние значения не используются. Элементы массивов нумеруются с нуля, поэтому максимальный номер элемента всегда на единицу меньше размерности. Номер элемента указывается после его имени в квадратных скобках: например a[0], a[3].

Если до начала работы программы неизвестно, сколько в массиве элементов, то в программе следует использовать динамические массивы. Память под них выделяется с помощью операции new или функции malloc в динамической области памяти во время выполнения программы. Адрес начала массива хранится в переменной, называемой указателем. Например:

```
int n = 10;
int *mass1 = new int[n];
```

Во второй строке объявлен указатель на целую величину, которому присваивается адрес начала непрерывной области динамической памяти, выделенной с помощью операции new. Выделяется столько памяти, сколько необходимо для хранения n величин типа int. Величина n может быть переменной. Инициализировать динамический массив нельзя.

Обращение к элементу динамического массива осуществляется так же, как и к элементу обычного. Если динамический массив в какой-то момент работы программы перестает быть нужным и мы собираемся впоследствии использовать эту память повторно, то необходимо освободить ее с помощью операции delete[], например: delete [] a; (размерность массива при этом не указывается):

```
delete[] mass1;
```

При необходимости создания многомерных динамических массивов сначала следует с помощью операции new выделить память под n указателей (вектор, элемент которого – указатель), при этом все указатели располагаются в памяти последовательно друг за другом. После этого необходимо в цикле каждому указателю присвоить адрес выделенной области памяти размером, равным второй границе массива:

```
mass2=new int*[row];  // mass2 – указатель на  массив указателей на одномерные массивы
        for(i=0;i<row;i++)
        mass2[i]=new int[col];// каждый элемент массива указывает на  одномерный
        for (i=0; i<row;i++)
        for (j=0;j<col;j++);
```

Освобождение памяти от двухмерного динамического массива:

```
for(i=0;i<row;i++)//удаление всех одномерных  массивов
        delete[] mass2[i];
    delete[] mass2;// удаление массива указателей/ на одномерные массивы.
```

Задание к лабораторной работе

Общая постановка. Составить программы – одномерные массивы : задания 1 – 25, двухмерные массивы: задания 26 – 44. Массивы создаются в динамической области памяти с использованием операций new и delete. Ввод исходных данных: реальный размер массивов и их значения. Обращение к элементам массива – через косвенную адресацию.

Варианты заданий

1 Заданы два массива – A(5) и B(4). Первым на печать вывести массив, сумма значений которого окажется наименьшей.

2 Заданы два массива – A(5) и B(4). Первым на печать вывести массив, произведение значений которого окажется наименьшим.

3 Заданы два массива – A(5) и B(5). В каждом из массивов найти наименьшее значение и прибавить его ко всем элементам массивов. На печать вывести исходные и преобразованные массивы.

4 Заданы два массива – A(5) и B(5). В каждом из массивов найти наибольшее значение и вычесть его из всех элементов массивов. На печать вывести исходные и преобразованные массивы.

5 Заданы два массива – A(5) и B(5). В каждом из массивов найти среднее арифметическое всех элементов массивов. На печать вывести исходные массивы и найденные значения.

6 Заданы два массива – A(5) и B(4). Первым на печать вывести массив, содержащий наибольшее значение. Напечатать также это значение и его порядковый номер.

7 Заданы два массива – A(5) и B(5). Подсчитать в них количество отрицательных элементов и первым на печать вывести массив, имеющий наименьшее их количество.

8 Заданы два массива – A(5) и B(5). Подсчитать в них количество положительных элементов и первым на печать вывести массив, имеющий наименьшее их количество.

9 Заданы два массива – A(5) и B(5). Подсчитать в них количество отрицательных элементов и первым на печать вывести массив, имеющий наибольшее их количество.

10 Заданы два массива – A(5) и B(5). Подсчитать в них количество положительных элементов и первым на печать вывести массив, имеющий наибольшее их количество.

11 Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, больших значения t, и первым на печать вывести массив, имеющий наименьшее их количество.

12 Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, меньших значения t, и первым на печать вывести массив, имеющий наименьшее их количество.

13 Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, больших значения t, и первым на печать вывести массив, имеющий наибольшее их количество.

14 Заданы два массива – A(5) и B(5). В каждом из массивов найти наименьшее значение и умножить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.

15 Заданы два массива – A(5) и B(5). В каждом из массивов найти наибольшее значение и умножить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.

16 Заданы два массива – A(5) и B(5). В каждом из массивов найти наименьшее значение и разделить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.

17 Заданы два массива – A(5) и B(5). В каждом из массивов найти наибольшее значение и разделить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.

18 Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, кратных двум, и первым на печать вывести массив, имеющий наибольшее их количество.

19 Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, кратных трем, и первым на печать вывести массив, имеющий наибольшее их количество.

20 Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, меньших значения t, и первым на печать вывести массив, имеющий наибольшее их количество.

21 Задан массив – A(10). Получить из него массив B, состоящий из элементов массива A, которые больше 0.

22 Задан массив – A(10). Получить из него массив B, состоящий из элементов массива A, которые меньше 0.

23 Задан массив – A(10). Получить из него массив B, состоящий из элементов массива A, которые кратны двум.

24 Задан массив – A(10). Получить из него массив B, состоящий из элементов массива A, которые больше значения T.

25 Задан массив – A(10). Получить из него массив B, состоящий из элементов массива A, которые кратны трем.

26 Дан массив – A(n, n). Найти число элементов массива a(i, j) > t и просуммировать все эти элементы.

27 Дан массив – A(n, n). Вычислить сумму всех неотрицательных элементов, а также их количество.

28 Дан массив – A(n, n). Вычислить сумму всех отрицательных его элементов и их количество.

29 Дан массив – A(n, n). Сформировать вектор B(k) из a(i, j) < 0. На печать вывести исходный массив, полученный вектор и его размерность.

30 Дан массив – A(n, n). Написать программу поворота массива на 90° относительно его центра. На печать вывести исходный и повернутый массивы.

31 Дан массив – A(n, n). Написать программу его поворота на 180° относительно его центра. На печать вывести исходный и повернутый массивы.

32 Дан массив – A(n, n). Написать программу его поворота на 270° относительно его центра. На печать вывести исходный и повернутый массивы.

33 Дан массив – A(n, n). Найти сумму всех его элементов, расположенных выше главной диагонали.

34 Дан массив – A(n, n). Найти сумму всех его элементов, расположенных ниже главной диагонали.

35 Дан массив – A(n, n). Найти сумму всех его элементов, расположенных выше диагонали, противоположной главной.

36 Дан массив – A(n, n). Найти сумму всех его элементов, расположенных ниже диагонали, противоположной главной.

37 Задана матрица – A(n, n). Найти суммы и произведения элементов, стоящих на главной и противоположной (побочной) диагоналях.

38 Задана матрица – A(n, n), состоящая из нулей и единиц. Подсчитать количество нулей и единиц в этой матрице.

39 Задана матрица – A(n, n). Переставить местами k-ю и i-ю строки, а затем i-й и j-й столбцы.

40 Задан двухмерный массив целых чисел A размером N на M. Найти сумму элементов, расположенных на главной диагонали.

41 Задан двухмерный массив целых чисел A размером N на M. Найти произведение элементов, расположенных на главной диагонали.

42 Задан двухмерный массив целых чисел A размером N на M. Найти максимальный элемент и поменять его с элементом A[1,1].

43 Задан двухмерный массив целых чисел A размером N на M. Найти минимальный элемент и поменять его с элементом A[1,1].

44 Задан двухмерный массив целых чисел A размером N на M. Найти максимальный элемент и поменять его с последним.

Контрольные вопросы

1 В чем заключается особенность динамических массивов?

2 Какие вы знаете операции динамического выделения и освобождения памяти в C++?

3 Как объявить динамический многомерный массив, используя указатели?

4 Что содержит указатель на массив?

Лабораторная работа №2
Классы. Программирование линейных алгоритмов с использованием функций инициализации set() и вывода результатов print()

Цель работы – изучить основные способы работы с пользовательским типом данных «класс», его объектами, методами и способы доступа к ним.

Теоретические сведения

Основное отличие C++ от C состоит в том, что в C++ имеются классы. С точки зрения языка C классы в C++ – это структуры, в которых вместе с данными определяются функции. Это и есть инкапсуляция в терминах ООП.

Класс (class) – это тип, определяемый пользователем, включающий в себя данные и функции, называемые методами или функциями-членами класса.

Данные класса – это то, что класс «знает».

Функции-члены (методы) класса – это то, что класс «делает».

Таким образом, определение типа, задаваемого пользователем (class), содержит спецификацию данных, требующихся для представления объекта этого типа, и набор операций (функций) для работы с подобными объектами.

Объявление класса

Приведем пример объявления класса:

```
class my_Fun
{
// компоненты-данные
        double x,y;
// компоненты-функции
public:
// функция инициализации
void set(char *c,double X)
    {
                x=X;
                y=sin(x);
     }
// функция вывода результатов
void print(void)
     {
cout << point<<y << endl;
      }
};
```

Обычно описания классов включают в заголовочные файлы (*.H), а реализацию функций-членов классов – в файлы *.CPP.

Для каждого объекта класса устанавливается область видимости либо явно – указанием уровня доступа одним из ключевых слов public, private, protected с двоеточием, либо неявно – по умолчанию. Указание области видимости относится ко всем последующим объектам класса, пока не встретится указание другой области видимости. Область видимости public разрешает доступ к объектам класса из любой части программы, в которой известен этот объект (общедоступный). Область видимости private разрешает доступ к объектам класса только из методов этого класса. Объекты с такой областью видимости называют частными. Область видимости protected определяется для защищенных объектов, она имеет смысл только в иерархической системе классов и разрешает доступ к объектам этой области из методов производных классов. В теле класса ключевое слово области видимости может использоваться неоднократно. Область видимости для объектов типа «класс» по умолчанию private.

Способы объявления и инициализации объектов и доступ к методам класса:

1 Прямой вызов

```
my_Fun Fun1;//объявление объекта1,но не инициализация
Fun1.set("Function1 = ",1.0);// инициализация данных
Fun1.print();  // прямой вызов
cout << "Input enter1..." << endl<<endl;
```

2 Косвенный вызов

```
my_Fun *p1 = &Fun1;// воспользовались объектом 1 новая инициализация
p1->set("Function1 = ",1.0);     // косвенный вызов
p1->print();// косвенный вызов
cout << "Input enter1..." << endl<<endl;
```

3 Динамическое выделение памяти

```
my_Fun *p1 = new my_Fun;
p1->set("Function1 = ",1.0);     // косвенный вызов
p1->print(); // косвенный вызов
cout << "Input enter1..." << endl<<endl;
delete p1; // удаляется динамически выделенный объект
```

Задание к лабораторной работе

Пользовательский класс должен содержать необходимые элементы-данные, а также метод установки их начальных значений:

```
Void set(double X, …);
```

метод печати:

```
Void print(void);
```

метод, решающий поставленную задачу:

```
Void Run(void);
```

Код методов – вне пространства определения класса. Программа должна включать в себя статический и динамический способы создания объектов и для каждого объекта использовать прямую и косвенную адресации при вызове методов класса.

Варианты заданий

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

При $x = 14{,}26$, $y = -1{,}22$, $z = 3{,}5\times10^{-2}$, $t = 0{,}564849$.

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

При $x = -4{,}5$, $y = 0{,}75\times10^{-4}$, $z = 0{,}845\times10^2$, $u = -55{,}6848$.

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

При $x = 3{,}74\times10^{-2}$, $y = -0{,}825$, $z = 0{,}16\times10^2$, $v = 1{,}0553$.

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

При $x = 0{,}4\times10^4$, $y = -0.875$, $z = -0{,}475\times10^{-3}$, $w = 1{,}9873$.

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

При $x = -15{,}246$, $y = 4{,}642\times10^{-2}$, $z = 20.001\times10^2$, $\alpha = -182.036$.

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

При $x = 16{,}55\times10^{-3}$, $y = -2{,}75$, $z = 0{,}15$, $\beta = -40{,}630$.

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

При $x = 0{,}1722$, $y = 6{,}33$, $z = 3{,}25 \times 10^{-4}$, $\gamma = -205{,}305$.

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

При $x = -2{,}235 \times 10^{-2}$, $y = 2{,}23$, $z = 15{,}221$, $\varphi = 39{,}374$.

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

При $x = 1{,}825 \times 10^2$, $y = 18{,}225$, $z = -3{,}298 \times 10^{-2}$, $\psi = 1{,}2131$.

10 $b = y^{\sqrt[3]{|x|}} + \cos^3(y)\frac{|x - y|\left(1 + \frac{\sin^2 z}{\sqrt{x + y}}\right)}{e^{|x-y|} + \frac{x}{2}}.$

При $x = 6{,}251$, $y = 0{,}827$, $z = 25{,}001$, $b = 0{,}7121$.

11 $c = 2^{\left(y^x\right)} + \left(3^x\right)^y - \frac{y\left(\text{arctg}z - \frac{\pi}{6}\right)}{|x| + \frac{1}{y^2 + 1}}.$

При $x = 3{,}251$, $y = 0{,}325$, $z = 0{,}466 \times 10^{-4}$, $c = 4{,}25$.

12 $f = \frac{\sqrt[4]{y + \sqrt[3]{x - 1}}}{|x - y|\left(\sin^2 z + \text{tg}z\right)}.$

При $x = 17{,}421$, $y = 10{,}365 \times 10^{-3}$, $z = 0{,}828 \times 10^5$, $f = 0{,}33056$.

13 $g = \dfrac{y^{x+1}}{\sqrt[3]{|y-2|}+3} + \dfrac{x+\dfrac{y}{2}}{2|x+y|}(x+1)^{-1/\sin z}$.

При $x = 12{,}3\times10^{-1}$, $y = 15{,}4$, $z = 0{,}252\times10^{3}$, $g = 82{,}8257$.

14 $h = \dfrac{x^{y+1}+e^{y-1}}{1+x|y-\mathrm{tg}z|}(1+|y-x|) + \dfrac{|y-x|^2}{2} - \dfrac{|y-x|^3}{3}$.

При $x = 2{,}444$, $y = 0{,}869\times10^{-2}$, $z = -0{,}13\times10^{3}$, $h = -0{,}49871$.

Контрольные вопросы

1 Что значит в ООП понятие «класс» и какой формат его объявления в программе?

2 Что такое объект класса, что он содержит?

3 Какие существуют уровни доступа к объектам и методам класса? (Дать характеристику каждому).

4 Что такое операция привязки, ее основное назначение?

Лабораторная работа №3
Классы. Программирование линейных алгоритмов с использованием конструктора, деструктора, friend-функции инициализации set() и функции вывода результатов print()

Цель работы – изучить основные способы работы по созданию конструктора класса с захватом динамической памяти и деструктора для ее освобождения, применение friend-функции и ее особенности.

Теоретические сведения

Конструктор класса

Конструктор – это метод класса, имя которого совпадает с именем класса. Конструктор вызывается автоматически после выделения памяти для переменной и обеспечивает инициализацию данных. Конструктор не имеет никакого типа (даже типа void) и не возвращает никакого значения в результате своей работы. Конструктор нельзя вызывать как обычную компонентную функцию в программе. Для класса может быть объявлено несколько конструкторов, различающихся числом и типами параметров. При этом, даже если для объектного типа не определено ни одного конструктора, компилятор создает для него конструктор по умолчанию, не использующий параметров, а также конструктор копирования, необходимый в том случае, если переменная объектного типа передается в конструктор как аргумент. В этом случае создаваемый объект будет точной копией аргумента конструктора.

```
class my_Fun
{
// компоненты-данные
        double x;
        unsigned size;
public:
// объявление конструктора 1 (с параметрами)
my_Fun (double X=0);
// объявление конструктора 2 (без параметров)
my_Fun(void);
// объявление и описание деструктора
~my_Fun ()
                {
cout<<"Destroyed object... "<<endl;
                }
// описание конструктора 1
my_Fun::my_Fun (double X)
    {
        cout<<"Constructor1...."<<endl;
                x=X;
```

```
    }
// описание конструктора 2
my_Fun::my_Fun (void)
{
        cout<<"Constructor2..."<<endl;
                x=5.0;
}
}
```

Деструктор класса

Еще одним специальным методом класса является деструктор. Деструктор вызывается перед освобождением памяти, занимаемой объектной переменной, и предназначен для выполнения дополнительных действий, связанных с уничтожением объектной переменной: например, для освобождения динамической памяти, закрытия, уничтожения файлов и т.п. Деструктор всегда имеет то же имя, что и имя класса, но перед именем записывается знак ~ (тильда). Деструктор не имеет параметров и, подобно конструктору, не возвращает никакого значения. Таким образом, деструктор не может быть перегружен и должен существовать в классе в единственном экземпляре. Деструктор вызывается автоматически при уничтожении объекта. Таким образом, для статически определенных объектов деструктор вызывается, когда заканчивается блок программы, в котором определен объект (блок в данном случае – составной оператор или тело функции). Для объектов, память для которых выделена динамически, деструктор вызывается при уничтожении объекта операцией delete.

Дружественная функция (friend)

В языке C++ одна и та же функция не может быть компонентом двух разных классов. Чтобы предоставить функции возможность выполнения действий над различными классами, можно определить обычную функцию языка C++ и предоставить ей право доступа к элементам класса типа private, protected. Для этого нужно в описании класса поместить заголовок функции, перед которым поставить ключевое слово friend.

Дружественная функция не является методом класса, не зависит от позиции в классе и спецификаторов прав доступа. Friend-функции получают доступ к членам класса через указатель, передаваемый им явно. Можно сделать все функции класса Y друзьями класса X в одном объявлении.

Задание к лабораторной работе

Общая постановка. Пользовательский класс X должен содержать необходимые элементы – данные x, y, а класс Y – переменную z, которые создаются в динамической области памяти; конструкторы для их создания (операция new) и установки их начальных значений соответственно: X(), Y(), деструкто-

ры: ~ X(), ~ Y(), friend-функция печати: friend void print(), функция, решающая поставленную задачу: friend void Run().

Код методов и функций – вне пространства определения класса.

Варианты заданий

Варианты заданий используются из лабораторной работы №2.

Контрольные вопросы

1 Что такое «конструктор», формат объявления, его особенности?

2 Формат объявления деструктора, его назначение.

3 Особенности дружественных функций, доступ к закрытой части класса.

4 Каким образом дружественная функция получает доступ к закрытой части класса?

Лабораторная работа №4
Класс «Динамическая строка» и перегрузка операций

Цель работы – изучить методику создания одномерных динамических символьных массивов при помощи конструкторов с захватом динамической памяти и деструкторов для их уничтожения, а также способ работы со строковыми объектами, познакомиться с механизмом перегрузки операций.

Теоретические сведения

Для представления символьной (текстовой) информации можно использовать символы, символьные переменные и символьные константы.

Символьная константа представляется последовательностью символов, заключенной в кавычки: «Начало строки \n». В C++ нет отдельного типа для строк. Массив символов – это и есть строка. Количество элементов в таком массиве на один элемент больше, чем изображение строки, т. к. в конец строки добавлен ‘\0’ (нулевой байт).

Присвоить значение массиву символов с помощью обычного оператора присваивания нельзя. Поместить строку в массив можно либо при вводе, либо с помощью инициализации:

char s[] = «ABCDEF»;

Для работы со строками существует специальная библиотека string.h. Примеры функций для работы со строками из библиотеки string.h приведены в таблице 4.1.

Таблица 4.1 – Функции работы со строками

Функция	Прототип и краткое описание функции
strcmp	int strcmp(const char *str1, const char *str2); Сравнивает строки str1 и str2. Если str1 < str2, то результат отрицательный; если str1 = str2, то результат равен 0; если str1> str2, то результат положительный
strcpy	char* strcpy(char*s1, const char *s2); Копирует байты из строки s1 в строку s2
strdup	char *strdup (const char *str); Выделяет память и переносит в нее копию строки str
strlen	unsigned strlen (const char *str); Вычисляет длину строки str
strncat	char *strncat(char *s1, const char *s2, int kol); Приписывает kol символов строки s1 к строке s2
strncpy	char *strncpy(char *s1, const char *s2, int kol); Копирует kol символов строки s1 в строку s2
strnset	char *strnset(char *str, int c, int kol); Заменяет первые kol символов строки s1 символом c

Строки при передаче в функцию в качестве фактических параметров могут быть определены либо как одномерные массивы типа char[], либо как указа-

тели типа char*. В отличие от обычных массивов в этом случае нет необходимости явно указывать длину строки.

Функции преобразования строки S в число:

– целое: int atoi(S);

– длинное целое: long atol(S);

– действительное: double atof(S),при ошибке возвращает значение 0.

Функции преобразования числа V в строку S:

– целое: itoa(int V,char S,int kod);

– длинное целое: ltoa(long V,char S,int kod); 2<=kod<=36, для отрицательных чисел kod=10.

Перегрузка операций

Для перегрузки операции для класса в C++ используется следующий синтаксис:

<Тип> operator <операция>(<входные параметры>)

{

<операторы>;

}

где < Тип > – тип, возвращаемый функцией;

operator – ключевое слово;

< операция > – перегружаемая операция.

В языке C++ имеются следующие ограничения на перегрузку операций:

– C++ не различает префиксную и постфиксную формы ++ и - -;

– переопределяемая операция должна присутствовать в языке (например, нельзя определить операцию с символом #);

– нельзя переопределить операторы, заданные следующими символами: . * :: ? ;

– переопределенные операции сохраняют свой изначальный приоритет.

Наличие в классе конструктора String:: String(String&) и операторов присваивания позволяет защитить объекты класса от побитового копирования.

Файловые потоки. Классы файловых потоков:

ifstream – файл ввода, производный от istream,
ofstream – файл вывода, производный от ostream,
fstream – файл ввода-вывода, производный от iostream.

Флаги режимов работы с файлом:

```
enum ios::open_mode
        {
in =   0x01,        // Открыть файл только для чтения
out =  0x02,        // Открыть файл только для записи
ate =  0x04,        // При открытии позиционироваться в конец файла
```

```
app = 0x08,          // Открыть существующий для дополнения
trunc =      0x10, // Создание нового файла взамен существующего
nocreate=0x20,     // Не создавать новый файл при его отсутствии
noreplace=0x40,    // Не создавать новый файл, если он существует
binary=      0x80  // Двоичный файл («прозрачный» ввод-вывод без
                   // преобразования символов конца строки)
    };
```

Конструкторы объектов (для классов ifstream, ofstream, fstream) и функции открытия/закрытия файлов:

```
ifstream();          // Без открытия файлов
ifstream(            // С открытием файла в заданном
    char *name,          // режиме imode
    int imode=ios::in,
    int prot=filebuf::openprot);
ifstream(int fd);    // С присоединением файла с дескриптором fd
ifstream(            // То же с явно заданным буфером
    int fd,
    char *buf, int sz);
void  ifstream::open(
    char *name, // Открытие файла в заданном режиме
    int imode=ios::in,
    int prot=filebuf::openprot);
void  close();       // Закрыть файл
void  setbuf(
    char *p,int sz);// Установить буфер потока
int   fd();          // Дескриптор открытого в потоке файла
int   is_rtl_open();       // 1 – файл открыт в потоке
```

Унаследованные переопределения операторов позволяют проверять наличие ошибок в потоках в следующем виде:

```
fstream      ss;
if (ss) ...      или   if (!ss) ...
```

Задание к лабораторной работе

Общая постановка. Пользовательский класс String должен содержать необходимые элементы-данные, которые создаются в динамической области памяти.

Конструктор для создания строк: String (…);

Деструктор: ~String();

Метод ввода исходной строки: Set();

Метод печати: void print(…);

Код методов – вне пространства определения класса. Программа иллюстрирует прямой и косвенный способы обращения к методам.

Ввести с клавиатуры строку символов S1. Признак окончания ввода строки – нажатие клавиши «Ввод». Программа должна содержать перегруженную операцию «=», использование которой скопирует S1 в S2 .

Исходную и преобразованную строки вывести в файл. В программе необходимо использовать стоковые и файловые потоки.

Варианты заданий

1 Если длина *L* нечетная, то удаляется символ, стоящий по середине строки.

2 Если длина *L* четная, то удаляются 2 первых и 2 последних символа.

3 Если длина *L* кратна 2, то удаляются все числа, которые делятся на 2.

4 Если длина *L* кратна 3, то удаляются все числа, делящиеся на 3.

5 Если длина *L* >10, то удаляются все цифры.

6 Если длина *L* >15, то удаляются все a…z.

7 Если длина *L* = 10, то удаляются все A…Z.

8 Если длина *L* кратна 4, то первая часть строки меняется местами со второй.

9 Если длина *L* кратна 5, то подсчитывается количество скобок всех видов.

10 Если длина *L* > 5, то выделяется подстрока до первого пробела.

11 Если длина *L* > 6, то выделяется подстрока { } скобках.

12 Если длина *L* > 10, то удаляется подстрока в [] скобках.

13 Если длина *L* > 12, то удаляется подстрока до первой открывающейся скобки.

14 Если длина *L* кратна 4, то выделяется подстрока после последнего пробела.

15 Если длина *L* >5, то удаляются все точки.

16 Если длина *L* четная, то выделяется подстрока до первого пробела.

17 Если длина *L* четная, то удаляется подстрока до первого пробела.

18 Если длина *L* четная, то выделяется подстрока со второго пробела.

19 Если длина *L* нечетная, то выделяется подстрока после первого пробела.

20 Если длина *L* нечетная, то удаляется подстрока со второго пробела.

21 Если длина *L* кратна 3, то удаляется каждый 3-й символ.

22 Если длина *L* четная, то удаляется каждый 2-й символ.

23 Если длина *L* нечетная, то удалить средние 3 символа.

24 Если длина *L* четная, то выделяется подстрока до последнего пробела.

25 Если длина *L* нечетная, то выделяется подстрока от последней цифры.

26 Если длина *L* =15, то удаляются все символы, кроме *A – Z*.

27 Если длина *L* делится на 5, то удаляется все символы, кроме *a* – *z*.

28 Если длина *L* четная и >= 10, то удаляются все пробелы.

29 Если длина *L* нечетная и < 12, произвести инверсию (abcdef в fedcba).

30 Если длина *L* > 5 и < 30, изменить регистр символов (aBcDeF в AbCdEf).

Контрольные вопросы

1 Как объявить динамическую строку в C++?

2 Какие вы знаете функции работы со строками?

3 Как определяются строки при передаче в функцию в качестве фактических параметров?

4 Поясните механизм перегрузки операций для объектов данного класса.

Лабораторная работа №5
Наследование классов, механизм виртуальных функций

Цель работы – изучить одну из базовых концепций ООП, наследование классов в С++, заключающуюся в построении цепочек классов, связанных иерархически; познакомиться с механизмом виртуальных функций.

Теоретические сведения

Наследование – механизм создания производного класса из базового, т.е. к существующему классу можно что-либо добавить или изменять его каким-либо образом для создания нового (производного) класса. Это мощный механизм для повторного использования кода. Наследование позволяет создавать иерархию связанных типов, совместно использующих код и интерфейс. Модификатор прав доступа используется для изменения доступа к наследуемым объектам в соответствии с правилами, указанными в таблице 5.1.

Таблица 5.1 – Доступ в классах при наследовании

Доступ в базовом классе	Модификатор прав доступа	Доступ в производном классе
private	private	недоступны
private	public	недоступны
protected	private	private
protected	public	protected
public	private	private
public	public	public

Ограничение на наследование

При определении производного класса не наследуются из базового:

1) конструкторы;
2) деструкторы;
3) операторы new, определенные пользователем;
4) операторы присвоения, определенные пользователем;
5) отношения дружественности.

Использование косвенной адресации с установкой указателей на базовый класс. Механизм косвенной адресации рассмотрим на примере:

```
class B
{
    public:
        int x;
        B() {  // Конструктор по умолчанию
```

```
        x = 4; }        };
        class D : public B {// Производный класс
                public:
                    int y;
            D()
        {// Конструктор по умолчанию
                    y = 5; }
            };
void main(void)  {
D  d;// Конструктор класса D создает объект d
B *p; // Указатель установлен на базовый касс
p = &d;// Указатель p инициализируется адресом d
// косвенное обращение к объектам базового и производного клас-
сов
// считываем их текущее состояние в переменные
int i = p -> x;         // Базовый класс виден напрямую
int j = ( ( D* ) p )p -> y;// Прямое преобразование указателя на D
// через переменные печатаем их текущее состояние
cout << " x_i= " << i << endl;
cout << " y_j= " << j << endl;
getch();
}
```

Виртуальная функция и механизм позднего связывания

Виртуальная функция объявляется в базовом или в производном классе и затем переопределяется в наследуемых классах. Совокупность классов (подклассов), в которых определяется и переопределяется виртуальная функция, называется полиморфическим кластером, ассоциированным с некоторой виртуальной функцией. В пределах полиморфического кластера сообщение связывается с конкретной виртуальной функцией-методом во время выполнения программы.

Обычную функцию-метод можно переопределить в наследуемых классах. Однако без атрибута virtual такая функция-метод будет связана с сообщением на этапе компиляции. Атрибут virtual гарантирует позднее связывание в пределах полиморфического кластера.

Часто возникает необходимость передачи сообщений объектам, принадлежащим разным классам в иерархии. В этом случае требуется реализация механизма позднего связывания. Чтобы добиться позднего связывания для объекта, его нужно объявить как указатель или ссылку на объект соответствующего класса. Для открытых производных классов указатели и ссылки на объекты этих классов совместимы с указателями и ссылками на объекты базового класса (т. е. к объекту производного класса можно обращаться, как будто это объект базового класса). Выбранная функция-метод зависит от класса, на объект которого указывается, но не от типа указателя.

C++ поддерживает virtual функции-методы, которые объявлены в основном классе и переопределены в порожденном классе. Иерархия классов, определенная общим наследованием, создает связанный набор типов пользователя, на которые можно ссылаться с помощью указателя базового класса. При обращении к виртуальной функции через этот указатель в C++ выбирается соответствующее функциональное определение во время выполнения. Объект, на который указывается, должен содержать в себе информацию о типе, поскольку различие между ними может быть создано динамически. Эта особенность типична для ООП–кода. Каждый объект «знает» как на него должны воздействовать. Эта форма полиморфизма называется чистым полиморфизмом.

В C++ функции-методы класса с различным числом и типом параметров есть действительно различные функции, даже если они имеют одно и то же имя. Виртуальные функции позволяют переопределять в управляемом классе функции, введенные в базовом классе, даже если число и тип аргументов те же самые. Для виртуальных функций нельзя переопределять тип функции. Если две функции с одинаковым именем будут иметь различные аргументы, C++ будет считать их различными и проигнорирует механизм виртуальных функций. Виртуальная функция – обязательно метод класса.

Задание к лабораторной работе

Общая постановка. Программа должна содержать:

– базовый класс X, включающий два элемента x1, x2 типа int;

– конструктор с параметрами для создания объектов в динамической области памяти;

– деструктор;

– виртуальные методы просмотра текущего состояния и переустановки объектов базового класса в новое состояние;

– производный класс У, включающий один элемент у типа int;

– конструктор с параметрами и списком инициализаторов, передающий данные конструктору базового класса;

– переопределенные методы просмотра текущего состояния объектов и их переустановку в новое состояние.

Варианты заданий

Создать в производном классе метод Run**,** определяющий:

1 сумму переменных классов;
2 произведение переменных классов;
3 сумму квадратов переменных классов;
4 значение x1+x2 – y;
5 значение (x1+x2)/y;
6 значение (x1+x2)*y;
7 значение x1*y+x2;

8 значение x1+x2*y.
9 произведение квадратов переменных класса;
10 значение x1*x2+y;
11 значение x1*x2/y;
12 значение x1*x2-y;
13 значение (x1-x2)*y;
14 значение (x1-x2)/y.

Контрольные вопросы

1 Что такое наследование, одиночное наследование, множественное наследование?

2 Какие объекты базового класса наследуются в производном, а какие не наследуются?

3 На примере своей программы поясните механизм позднего связывания.

4 В каком случае C++ проигнорирует механизм виртуальных функций?

Лабораторная работа №6
Программирование шаблона классов

Цель работы – изучить приемы создания и использования шаблонов классов.

Теоретические сведения

Достаточно часто встречаются классы, объекты которых должны содержать элементы данных произвольного типа (в том смысле, что их тип определяется отдельно для каждого конкретного объекта). В качестве примера можно привести любую структуру данных (массив указателей, список, дерево). Для этого в C++ предлагаются средства, позволяющие определить некоторое множество идентичных классов с параметризованным типом внутренних элементов. Они представляют собой особого вида заготовку класса, в которой в виде параметра задан тип (класс) входящих в него внутренних элементов данных. При создании конкретного объекта необходимо дополнительно указать и конкретный тип внутренних элементов в качестве фактического параметра. Создание объекта сопровождается созданием соответствующего конкретного класса для типа, заданного в виде параметра. Принятый в C++ способ определения множества классов с параметризованным внутренним типом данных (иначе – макроопределение) называется шаблоном (template).

Синтаксис шаблона рассмотрим на примере шаблона класса векторов, содержащих динамический массив указателей на переменные заданного типа.

```
// <class T> – параметр шаблона - класс «T», внутренний тип данных
// vector      – имя группы шаблонных классов
template <class T> class vector
{
int     tsize;       // Общее количество элементов
int     csize;       // Текущее количество элементов
T  **obj; // Массив указателей на параметризованные //объекты типа «T»
public:
T *operator[](int); // оператор [int] возвращает указатель на
                                  // параметризованный объект  класса «T»
void   insert(T*);   // включение указателя на объект типа «T»
int     index(T*);
};
```

Данный шаблон может использоваться для порождения объектов-векторов, каждый из которых хранит объекты определенного типа. Имя класса при этом составляется из имени шаблона «vector» и имени типа данных (класса), который подставляется вместо параметра «T»:

```
vector<int>  a;
vector<double> b;
extern class time;
```

```
vector<time> c;
```

Заметим, что транслятором при определении каждого вектора с новым типом объектов генерируется описание нового класса по заданному шаблону (естественно, неявно в процессе трансляции). Например, для типа int транслятор получит:

```
class vector<int>
{
        int     tsize;
        int     csize;
        int     **obj;
public:
        int     *operator[](int);
        void    insert(int*);
        int     index(int*);
};
```

Далее следует очевидное утверждение, что функции-методы шаблона также должны быть параметризированы, т. е. генерироваться для каждого нового типа данных. Действительно, это так: функции-методы шаблона классов в свою очередь также являются шаблонными функциями с тем же параметром. То же самое касается переопределяемых операторов:

```
// параметр шаблона – класс «T», внутренний тип данных
// имя функции-элемента или оператора – параметризировано
template <class T>  T* vector<T>::operator[](int n)
{
if (n >=tsize) return(NULL);
return (obj[n]);
}
template <class T> int vector<T>::index(T *pobj)
{
int     n;
for (n=0; n<tsize; n++)
        if (pobj == obj[n]) return(n);
return(-1);
}
```

Заметим, что транслятором при определении каждого вектора с новым типом объектов генерируется набор методов-функций по заданным шаблонам (естественно, неявно в процессе трансляции). При этом сами шаблонные функции должны размещаться в том же заголовочном файле, где размещается определение шаблона самого класса. Для типа int сгенерированные транслятором функции-методы будут выглядеть так:

```
int* vector<int>::operator[](int n)
```

```
{if (n >=tsize) return(NULL);
return (obj[n]);}
int vector<int>::index(int *pobj)
{int    n;
for (n=0; n<tsize; n++)
        if (pobj == obj[n]) return(n);
return(-1);
}
```

Задание к лабораторной работе

Общая постановка. Дано: число N и последовательность $a_1, a_2, \ldots a_N$. Создать шаблон класса, порождающий динамические одномерные массивы с элементами различных типов (вещественные, целочисленные, символьные). Тип данных и результат являются параметрами по отношению к классу, программа должна иметь методы инициализации, конструктор, деструктор, метод просмотра значений созданного массива согласно заданному алгоритму.

Для упрощения работы с массивами использовать возможности библиотеки шаблонов STL.

Варианты заданий

1 $a_1, (a_1+a_2), \ldots ,(a_1+a_2+\ldots+a_N)$;
2 $(a_1*a_1), (a_1*a_2), \ldots, (a_1*a_N)$;
3 $|a_1|, |a_1+a_2|, \ldots, |a_1+a_2+\ldots a_N|$;
4 $a_1, -a_1*a_2, +a_1*a_2*a_3, \ldots ,(-1)^N*a_1* a_2*\ldots a_N$;
5 $-a_1, +a_2, -a_3, \ldots , (-1)^N* a_N$;
6 $(a_1+1), (a_2+2) , (a_3+3), \ldots, (a_N +N)$;
7 a_1*1, a2*2, $a_3*3, \ldots, a_N *N$;
8 $a_1* a_2, a_2* a_3, \ldots , a_N -1* a_N$;
9 $a_1/1, a_2/2, a_3/3, \ldots , a_N /N$;
10 $(a_1+ a_2), (a_2+ a_3), \ldots ,(a_N -1+ a_N)$;
11 $(a_1+ a_2+ a_3), (a_2+ a_3+ a_4), (a_3+ a_4+ a_5), \ldots (a_N -2+ a_N -1+ a_N)$;
12 $(N+ a_1), (N-1+ a_2), \ldots ,(1+ a_N)$;
13 $(N* a_1), ((N-1)* a_2), \ldots ,(1* a_N)$;
14 $a_1/N, a_2/N, \ldots , a_N /1$.

Контрольные вопросы

1 С какой целью используются шаблоны классов?

2 Какие существуют виды параметров шаблона класса?

3 В чем заключается особенность использования функций-методов шаблона?

4 Может ли использоваться шаблон для параметризованных объектов?

Лабораторная работа №7

Множественное наследование с использованием абстрактных базовых классов, файлового ввода – вывода с применением потоков C++, функций обработки исключительных ситуаций

Цель работы – изучить методику создания множественного наследования, использование абстрактного базового класса, файловый ввод–вывод и использование функций обработки исключительных ситуаций.

Теоретические сведения

Абстрактные классы

Если базовый класс используется только для порождения производных классов, то виртуальные функции в базовом классе могут быть «пустыми», поскольку никогда не будут вызваны для объекта базового класса. Базовый класс, в котором есть хотя бы одна такая функция, называется *абстрактным*. Виртуальные функции в определении класса обозначаются следующим образом:

```
classbase
{public:
virtual print()=0;
virtual get() =0;
};
```

Определять тела этих функций не требуется.

Множественное наследование

Множественным наследованием называется процесс создания производного класса из двух и более базовых. В этом случае производный класс наследует данные и функции всех своих базовых предшественников. Существенным для реализации множественного наследования является то, что адреса объектов второго и последующих базовых классов не совпадают с адресом объекта производного класса. Этот факт должен учитываться транслятором при преобразовании указателя на производный класс в указатель на базовый и наоборот:

```
class d : public a,public b, public c { };
d      D1;
pd =    &D1;        // #define dbsizeof(a)
pa =    pd;         // #define dc sizeof(a)+sizeof(b)
pb =    pd;         // pb = (char*)pd + db
pc =    pd;         // pc = (char*)pd + dc
```

Такое действие выполняется компилятором как явно, при преобразовании в программе типов указателей, так и неявно, когда в объекте производного класса наследуется функция из второго и последующих базовых классов. Для вышеуказанного примера при определении в классе bb функции f() и ее наследовании в классе d вызов D1.f() будет реализован следующим образом:

this = &D1// Указатель на объект производного класса
this = (char*)this + db;// Смещение к объекту базового класса
b::f(this);// Вызов функции в базовом классе

Механизм виртуальных функций при множественном наследовании имеет свои особенности. Во-первых, на каждый базовый класс в производном классе создается свой массив виртуальных функций (в нашем случае – для aa в d, для bb в d и для cc в d). Во-вторых, если функция базового класса переопределена в производном, то при ее вызове требуется преобразовать указатель на объект базового класса в указатель на объект производного. Для этого транслятор включает соответствующий код, корректирующий значение this в виде «заплаты», передающей управление командой перехода к переопределяемой функции, либо создает отдельные таблицы смещений.

Обработка исключительных ситуаций

Средства обработки ошибочных ситуаций позволяют передать обработку исключений из кода, в котором возникло исключение, некоторому другому программному блоку, который выполнит в данном случае некоторые определенные действия. Таким образом, основная идея данного механизма состоит в том, что функция проекта, которая обнаружила непредвиденную ошибочную ситуацию, которую она не знает, как решить, генерирует сообщение об этом (бросок исключения). Система в свою очередь вызывает по этому сообщению программный модуль, который перехватит исключение и отреагирует на возникшее нештатное событие. Такой программный модуль называют «обработчик» или перехватчик исключительных ситуаций, и в случае возникновения исключения в его «обработчик» передаётся произвольное количество информации с контролем типа. Эта информация и является характеристикой возникшей нештатной ситуации.

Обработка исключений в C++ – это обработка с завершением. Это означает, что исключается невозможность возобновления выполнения программы в точке возникновения исключения.

Для обеспечения работы такого механизма были введены следующие ключевые слова:

– try – проба испытания;
– catch – перехватить (обработать);
– throw – бросать.

Кратко рассмотрим их назначение.

Try – открывает блок кода, в котором может произойти ошибка; это обычный составной оператор:

```
try
{   код   };
```

Код содержит набор операций и операторов, который и будет контролироваться на предмет возникновения ошибки. В него могут входить вызовы функции пользователя, которые компилятор также возьмет на контроль. Среди данного набора операторов и операций обязательно указывают операцию броска исключения: throw.

Операция броска throw имеет следующий формат:

throw выражение;

где «выражение» определяет тип информации, которая и описывает исключение (например, конкретные типы данных);

catch – сам обработчик исключения, который перехватывает информацию:

```
catch ( тип, параметр)
{код}
```

Через параметр обработчику передаются данные определенного типа, описывающие обрабатываемое исключение. Код определяет те действия, которые надо выполнить при возникновении данной конкретной ситуации. В C++ используют несколько форм обработчиков. Такой обработчик получил название «параметризованный специализированный перехватчик». Перехватчик должен следовать сразу же после блока контроля, т. е. между обработчиком и блоком контроля не должно быть ни одного оператора. При этом в одном блоке контроля можно вызывать исключения разных типов для разных ситуаций, поэтому обработчиков может быть несколько. В этом случае их необходимо расположить сразу же за контролирующим блоком последовательно друг за другом. Кроме того, запрещены переходы как извне в обработчик, так и между обработчиками. Можно воспользоваться универсальным или абсолютным обработчиком:

```
catch()
{код},
```

где (...) означают способность данного перехватчика обрабатывать информацию любого типа. Такой обработчик располагают последним в пакете специализированных обработчиков. Тогда, если исключение не будет перехвачено специализированными обработчиками, будет выполнен последний – универсальный.

В случае невозникновения исключения набор обработчиков будет обойден, т. е. проигнорирован.

Если же исключение было брошено, при возникновении критической ситуации будет вызван конкретный перехватчик при совпадении его параметра с выражением в операторе броска, т. е. управление будет передано найденному обработчику. После выполнения кода вызванного обработчика управление передается оператору, который расположен за последним перехватчиком, или проект корректно завершает работу.

Существенное отличие вызова конкретного обработчика от вызова обычной функции заключается в следующем: при возникновении исключения и передаче управления определенному обработчику система осуществляет вызов

всех деструкторов для всех объектов классов, которые были созданы с момента начала контроля и до возникновения исключительной ситуации с целью их уничтожения.

Блоки try как составные блоки могут быть вложены:

```
try {
...
        try{  ...  }
        ...}
```

Тогда, в случае возникновения исключения в некотором текущем блоке поиск обработчика последовательно продолжается в блоках, предшествующих уровням вложенности с продолжением вызова деструкторов.

Задание к лабораторной работе

Общая постановка. Создать программу с абстрактным базовым классом и множественным наследованием (либо с иерархией классов), реализовать в нем:

- конструктор;
- деструктор;
- виртуальную функцию просмотра текущего состояния объекта print();
- friend-функцию Run().

Производные классы должны содержать переопределенную функцию просмотра состояния объектов, а также при вводе–выводе данных использовать функции обработки исключительных ситуаций. Используя стандартные файловые потоки, информацию об объектах вывести в файл. Для корректной работы с файлом и корректного ввода данных использовать обработку исключительных ситуаций.

Варианты заданий

1 Список товаров, имеющихся на складе, включает в себя наименование товара, количество единиц товара, цену единицы и дату поступления товара на склад. Вывести в алфавитном порядке список товаров, хранящихся больше месяца, стоимость которых превышает 1 000 000 р.

2 Для получения места в общежитии формируется список студентов, который включает ФИО студента, группу, средний балл, доход на члена семьи. Общежитие в первую очередь предоставляется тем, у кого доход на члена семьи меньше двух минимальных зарплат, затем остальным в порядке уменьшения среднего балла. Вывести список очередности предоставления мест в общежитии.

3 В справочной автовокзала хранится расписание движения автобусов. Для каждого рейса указаны его номер, тип автобуса, пункт назначения, время отправления и прибытия. Вывести информацию о рейсах, которыми можно воспользоваться для прибытия в пункт назначения раньше заданного времени.

4 На междугородной АТС информация о разговорах содержит дату разговора, код и название города, время разговора, тариф, номер телефона в этом городе и номер телефона абонента. Вывести по каждому городу общее время разговоров с ним и сумму.

5 Информация о сотрудниках фирмы включает: ФИО, табельный номер, количество проработанных часов за месяц, почасовой тариф. Рабочее время свыше 144 ч считается сверхурочным. Вывести сотрудников фирмы, которые имеют сверхурочные часы.

6 Информация об участниках спортивных соревнований содержит: наименование страны, название команды, ФИО игрока, игровой номер, возраст, рост, вес. Вывести информацию о самой молодой, рослой и легкой команде.

7 Для книг, хранящихся в библиотеке, задаются регистрационный номер книги, автор, название, год издания, издательство, количество страниц. Вывести список книг с фамилиями авторов в алфавитном порядке, изданных после заданного года.

8 Различные цеха завода выпускают продукцию нескольких наименований. Сведения о выпущенной продукции включают: наименование, количество, номер цеха. Для заданного цеха необходимо вывести количество выпущенных изделий по каждому наименованию в порядке убывания количества.

9 Информация о сотрудниках предприятия содержит ФИО, номер отдела, должность, дату начала работы. Вывести списки сотрудников по отделам, которые проработали меньше года.

10 Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит ФИО, адрес, оценки. Вывести абитуриентов, проживающих в г. Минске и сдавших экзамены со средним баллом не ниже 8.

11 В налоговой инспекции содержатся данные о предприятиях, которые определяют название, УНП, ФИО директора, вид деятельности, дата регистрации. Вывести все предприятия по видам деятельности (исключать ликвидированные).

12 В справочной аэропорта хранится расписание вылета самолетов на следующие сутки. Для каждого рейса указаны: номер рейса, тип самолета, пункт назначения, время вылета. Вывести все номера рейсов, типы самолетов и времена вылета для заданного пункта назначения в порядке возрастания времени вылета.

13 Транспорт (наименование, тип, год выпуска, максимальная скорость, объем двигателя, расход, объем бензобака, расстояние без подзаправки – функция friend Run()).

14 Продовольственные товары (наименование, отдел магазина, дата выпуска, срок хранения, последний срок реализации – функция Run(), вес).

15 Объекты недвижимости (адрес, тип, этажность, квартир на этаже, подъездов, всего квартир – функция friend Run()).

16 Периодические издания (название, тип, страниц, частота выпуска, тираж, выпусков в год – функция friend Run()).

17 Отдел кадров (ФИО, отдел, должность, дата приема на работу, внутренний стаж – функция friend Run(), ставка).

18 Научно-исследовательские разработки (наименование, дата начала, дата завершения, срок работы – функция friend Run(), область исследования, количество сотрудников, ФИО сотрудников).

19 Программное обеспечение (наименование, тип, количество дисков, объем после установки (полной, минимальной, типичной версий), процент сжатия – функция friend Run()).

20 Комплектующие ЭВМ (наименование, тип, модель, частота, объем памяти, стоимость, количество, итоговая стоимость – функция friend Run()).

21 Аудио-студия (группа/исполнитель, количество чел., стиль, количество альбомов, стоимость записи диска [], стоимость диска [], тираж[], общая прибыль группы – функция friend Run()).

22 Мобильные телефоны (наименование, фирма, стандарт связи, заряд аккумулятора, потребление при ожидании, потребление при разговоре, время разговора –функция friend Run()).

23 Сетевое оборудование (наименование, скорость передачи данных, тип, стоимость, количество, общая стоимость – функция friend Run(), максимальная скорость передачи (байт/с)).

24 Гостиница (информация о номерах, клиентах, счетах, поиск – функция friend Run() – по номеру в гостинице).

25 Установка и обслуживание ЛВС (информация о сетевых конфигурациях, клиентах, договорах, поиск – функция friend Run() – по клиенту).

26 Фирма-поставщик медицинской техники (ассортимент, заказчики, контракты, поиск – функция friend Run() – по наименованию медицинской техники).

27 Автозаправочная станция (горюче-смазочные материалы, поставщики, накладные, поиск – функция friend Run()– по номеру накладной).

28 Студия видеозаписи (режиссеры/актеры, фильмы, продажи, поиск – функция friend Run()– по названию фильма).

29 Музей (экспонаты, авторы, экспозиции, поиск – функция friend Run()– по автору).

30 Страховая фирма (виды страховок, клиенты/объекты страховки, страховая деятельность, поиск – функция friend Run() – по клиенту).

Контрольные вопросы

1 Что такое множественное наследование?

2 Как объявляются виртуальные функции в абстрактном базовом классе?

3 Поясните механизм виртуальных функций при множественном наследовании.

4 Какие вы знаете функции обработки исключительных ситуаций (пояснить особенности каждой)?

Лабораторная работа №8
Мастера Classview и Classwizard в Visual C++

Цель работы – изучить методы работы с мастерами *Classview* и *Classwizard* и их возможности.

Теоретические сведения

При начальном знакомстве с концепциями и возможностями объектно-ориентированного стиля программирования на базе языка C++, как правило, методологию данного стиля изучают на простых, часто не несущих практической значимости примерах.

Так, например, проиллюстрировать ключевые понятия пользовательского типа данных «класс», положенного в основу ООП, можно на следующем простом примере.

Программа создает целочисленные объекты класса X, который помимо целочисленных трех закрытых элементов-данных имеет в своем составе три общедоступных метода (элементы-функции класса): для инициализации объектов, просмотра их текущего состояния, а также для суммирования значений первых двух элементов с сохранением результата в третьем.

Консольное приложение для демонстрации ключевых понятий ООП может быть, например, таким:

```
// подключили заголовочные файлы
#include <iostream.h>
#include <conio.h>
// декларируем класс»X»
class X
{// закрытая часть – реализация класса
int a, b, c; // три целочисленные элементы-данные класса (состав будущих
                    //объектов данного класса)
// открытая часть – интерфейс класса
public:// метод для инициализации класса
        void set(int i, int j)
        {a=i; b=j;
                cout<<"Object created!"<<endl;
        }
// метод для просмотра текущего состояния существующего объекта
        void print(void)
        {cout<<"for à="<<a<<" b="<<b<<"  summa="<<c<<endl;}
                // метод для взаимодействия с объектами
        void run(void)
        {
                c=a+b;
        }
```

```
};
void main(void)
{
X x1;          // объект создан
x1.set(2,3); //объект использован: проинициализирован
x1.run();      //объект использован: определено значение третьего элемента
x1.print();//объект использован: просмотр текущего состояния
        getch();}
//по окончании работы программы объект автоматически уничтожается
```

Запускаем и получаем:
Object created!
for a=2 b=3 summa=5

Еще более наглядную иллюстрацию ключевых понятий ООП можно получить, создав, например, при помощи *AppWizard* простую диалоговую панель с командной кнопкой. Участок кода по созданию и использованию объектов класса разместить в обработчике щелчка на данной кнопке, а сам класс X и его элементы добавить при помощи мастера *ClassView***.**

Рассмотрим этот мастер Visual C++ более подробно.

Используя *ClassView*, можно быстро и наглядно добавлять к существующим классам новые классы, новые для них элементы, просматривать структуру наследования, а также выполнять некоторые другие полезные операции.

Итак, создадим простую диалоговую панель (Имя проекта – CMy):

1) создать при помощи MFC *AppWizard(exe)* новый проект типа *Dialog based*;

2) убрать для упрощения на втором шаге *About box*;

3) используя панель *Controls*, нанести на заготовку формы 1 кнопку – *Button* (рисунок 8.1);

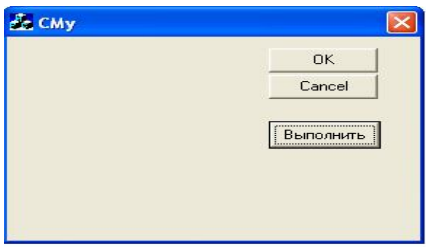

Рисунок 8.1 – Заготовка формы

4) в выпадающем меню пункт *Properties*, выбрать закладку *General* и в поле *Caption* ввести нужное имя элемента: название или надпись, в закладке *Styles* – нужные свойства элемента;

5) через *ClassWizard* связать закладку с обработчиком события одинарного щелчка *BN_CLICKED*. Для этого, щелкнув правой клавишей мыши в любом месте диалоговой панели в появившемся меню, опять выбрать пункт *ClassWi-*

zard, затем выбрать закладку *Message Maps*, связать идентификатор *IDC_Button1* с событием *BN_CLICKED* (один щелчок на кнопке) и добавить обработчик на данное событие: щелчок на кнопке *Add Function*, щелчок на «*OK*» для подтверждения, щелчок на кнопке *Edit Code* , и среда добавит в конец файла MyDlg.cpp пустой обработчик:

```
void CMyDlg::OnButton1()
{
// TODO: Add your control notification handler code here
}
```

6) теперь необходимо перейти на закладку *Class view* и выбрать необходимый класс, например просмотреть или изменить (например *CCMyDlg)*. На экране появится временное меню с пунктами, которые позволяют выполнять над классами все основные действия.

Например, строка *Go to definition* дает возможность просмотреть в окне редактирования исходный код класса. Можно редактировать класс непосредственно, а для добавления в класс новых методов и данных удобно пользоваться пунктами меню *Add Function* и *Add Variable*.

Добавление к проекту нового класса

Добавим к нашему проекту класс X**.** Для этого: выбираем пункт меню *Insert/New class*, заполняем появившуюся диалоговую панель и щелкаем «OK».

В составе классов появляется новый пользовательский класс X. Щелкаем правой клавишей мыши на нем и выбираем *Go to definition*. Имеем его первоначальный состав (файлы X.h и X.cpp):

```
// X.h: interface for the X class
class X
{
public:
        X();
        virtual ~X();
};
// X.cpp: implementation of the X class.
// Construction/Destruction
X::X()
{}
X::~X()
{}
```

Таким образом, средство *ClassView* создало пользовательский класс с декларацией в нем конструктора и деструктора. Реализация их в файле X.cpp отсутствует.

Добавление к классу элементов данных

Процедура добавления в класс новых данных довольно простая. Чтобы по условию добавить три целочисленных элемента данные, присвоим им идентификаторы m_a, m_b и m_c. Для их добавления нужно воспользоваться пунктом меню *Add Variable*.

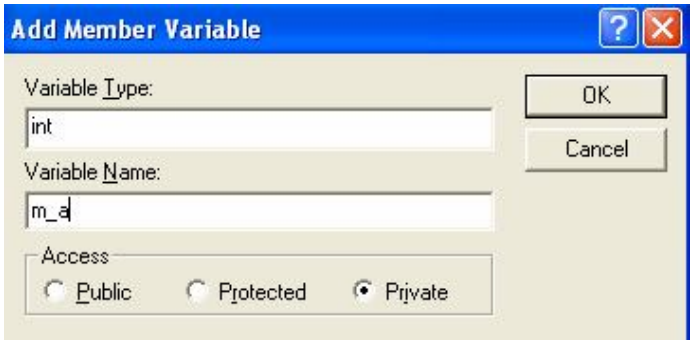

Рисунок 8.2 – Добавление данных

Щелкаем правой клавишей мыши на только что добавленном классе «X» и выбираем данный пункт меню. На экране – диалоговая панель «Add Member Variable», представленная на рисунке 8.2. Вводим тип, идентификатор, в переключателе с зависимой фиксацией Access выбираем уровень защищенности «частный» (закрытый) и нажимаем «ОК». Аналогично добавим оставшихся два элемента m_b и m_c. В файле X.h появится декларация этих трех элементов данных класса «X».

Добавление к классу методов

Методы класса добавляются аналогично добавлению элементов данных. Для того чтобы добавить в класс новый метод, нужно выбрать из временного меню строку' «Add Function». На экране теперь появится диалоговая панель «Add Member Function», представленная на рисунке 8.3.

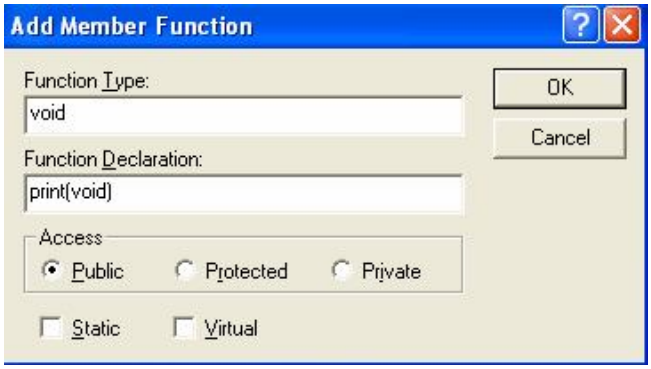

Рисунок 8.3 – Добавление функций

В поле *Function Type* следует ввести тип значения, возвращаемого методом. Как и раньше, и в этой диалоговой панели можно определить область видимости, но теперь уже метода.

Он, как и элемент-данное, может находиться в трех положениях: или public(общедоступный), или protected (защищенный), или private (закрытый). Как правило, большинство методов общедоступны из любой точки программы. Кроме того, переключатели *Static* и *Virtual* позволяют определить, как добавляемый метод должен быть объявлен соответственно: как статический или как виртуальный. Последние возможности здесь игнорируем, так как добавляются обычные методы класса.

Когда все поля диалоговой панели заполнены, щелкните на кнопке «*OK*». И в декларацию класса будет добавлен новый метод. А идентификатор метода появится в списке элементов класса в окне *ClassView*. Рисунок 8.3 иллюстрирует добавление к классу X метода print просмотра текущего состояния существующих объектов класса X. По аналогичной методике добавляем метод set для инициализации созданного объекта класса и метод run для определения суммы m_a и m_b.

Теперь в файле X.h будут добавлены и прототипы этих методов, и окончательный его вид будет таким:

```
// X.h: interface for the X class
class X
{public:
void run(void);
void set(int i, int j);
void print(void);
X();
virtual ~X();
private:
int m_c;
int m_b;
int m_a;
};
```

А в файле X.cpp *ClassView* разместит пустые определения этих добавленных в класс X методов. Заполняем их следующим образом:

```
#include "stdafx.h"
#include "CMy.h"
#include "X.h"
……..
// Construction/Destruction
X::X()
{
}
X::~X()
{
```

```
}
void X::set(int i, int j)
{
m_a=i; m_b=j;
AfxMessageBox("Объект создан и проинициализирован!");
}
void X::print(void)
{
        CString str;
// объявили строковый объект класса библиотеки MFC, и при помощи его
//вызвали метод,
// который работает как стандартная функция printf(…), но выводит значения
//объектов из
// списка вывода в указанную строку
        str.Format("Для m_a=%d, m_b=%d, \n их сумма=%d",m_a, m_b,
m_c);
AfxMessageBox(str);
}
void X::run(void)
{
m_c=m_a+m_b;
}
```

Теперь открываем файл проекта CMyDlg.cpp и в его начале подключаем заголовочный файл с декларацией построенного класса:

```
// CMyDlg.cpp : implementation file
#include "stdafx.h"
#include "CMy.h"
#include "CMyDlg.h"
#include "X.h"
……..
```

А в его конце оживляем обработчик щелчка на командной кнопке «*Выполнить*»:

```
void CCMyDlg::OnButton1()
{
X x1;
x1.set(2,3);
x1.run();
x1.print();
}
```

Запускаем, щелкаем на кнопке и получаем (рисунок 8.4):

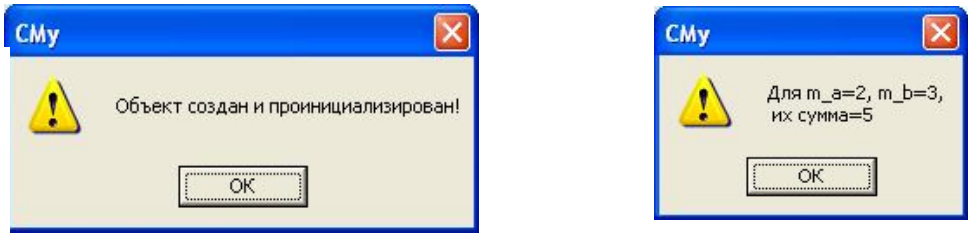

Рисунок 8.4 – Результат работы программы

Дополнительные возможности мастера

1 С помощью *ClassView* можно просмотреть список названий файлов, в которых используется данный класс. Для этого надо выбрать из временного меню строку «*References*». На экране появится диалоговая панель *Definitions and References*. Щелкаем правой клавишей мыши на классе X, выбираем вышеназванную опцию меню, получаем:

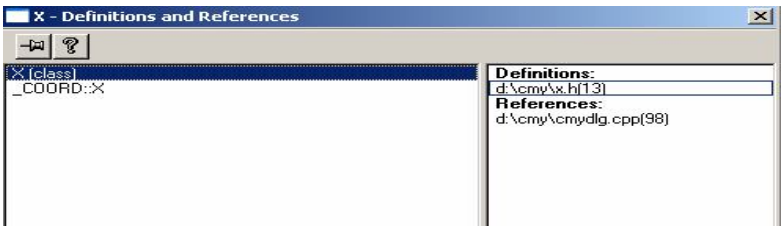

Рисунок 8.5 – Диалоговая панель Definitions and References

2 *ClassView* предоставляет пользователю полезную возможность: просмотреть деревья наследования классов приложения при наличии в нем иерархии классов, связанных между собой на основе концепции наследования. Для этого нужно выбрать название интересующего вас класса из списка классов, открыть временное меню, щелкнув, как и раньше, правой кнопкой мыши. Пункты «*Derived Classes*» и «*Base Classes*» позволяют просмотреть последовательность классов, порожденных от данного класса, и последовательность базовых для него классов.

На рисунке 8.6. показана иерархия классов для CCMyApp :

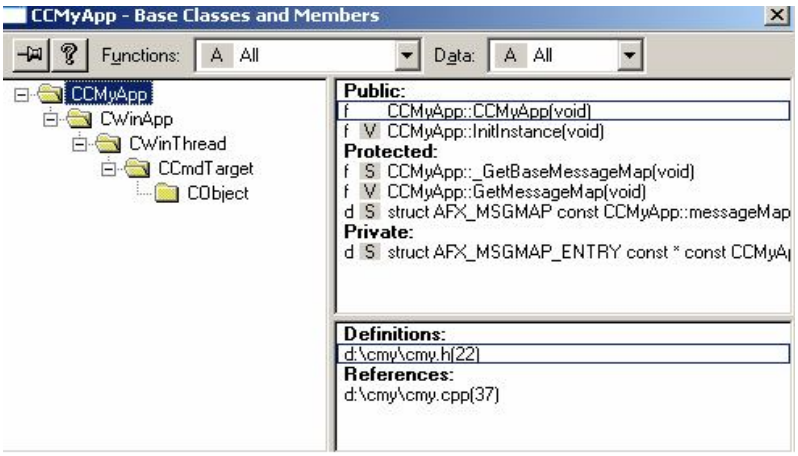

Рисунок 8.6 – Иерархия классов для CCMyApp

Данная диалоговая панель не имеет кнопок «*OK*» и «*Cancel*» и закрывается при выборе из нее какого-либо элемента класса или при переходе в другое окно. Иногда удобно, чтобы эта панель постоянно присутствовала на экране. Прикрепить ее к главному окну можно щелчком на самой левой кнопке заголовка этой панели (которая при этом изменит свой вид). В левой части панели отображается список классов в порядке наследования. Видим, что для класса CCMyApp верхушка иерархии – класс библиотеки *MFC CObject*. В правой части панели отображаются элементы-данные и элементы-функции выбранного из списка класса. Они разделены на группы в зависимости от степени защиты.

Панель *Base Classes and Members* позволяет легко перейти к редактированию любого элемента класса. Для этого нужно выбрать название этого элемента из списка и сделать по нему двойной щелчок левой кнопкой мыши. Чтобы легко различать методы и данные класса, перед методами располагается символ f, а перед данными – d. Непосредственно перед именем метода или данных расположен символ, позволяющий отличить статические данные от статических функций, а также виртуальные методы. Перед названиями статических данных и методов расположен символ S, а перед виртуальными методами – символ V. Если список методов и данных класса слишком велик, вы можете отображать в диалоговой панели только их часть. Выберите из поля «Functions» типы методов, имена которых надо отобразить. Доступны следующие типы методов:

Таблица 8.1 – Методы поля *Functions*

Поле *Functions*	Методы
All	Все
Virtual	Виртуальные
Static	Статические
Non- Static	Все, кроме статических и виртуальных
Non- Virtual	Все, кроме статических и виртуальных
None	Не отображать методы класса

Аналогично из поля «*Data*» можно выбрать типы элементов для элементов-данных класса, которые надо отображать на экране. В следующей таблице приведены типы данных, представленные в списке «*Data*».

Таблица 8.2 – Типы элементов поля *Data*

Поле *Data*	Элементы данных
All	Все
Static	Статические
Non- Static	Все, кроме статических
None	Не отображать

В нижней правой части диалоговой панели «*Base Classes and Members»* отображены названия файлов, в которых определен (группа *Definitions*) и в которых используется (группа *References*) выбранный элемент. Двойной щелчок левой кнопкой мыши позволяет открыть в редакторе соответствующий файл. Курсор при этом сразу устанавливается в то место, где объявляется или используется выбранный элемент класса.

3 Редактирование методов класса. Выбираем из списка элементов класса метод и щелкаем правую кнопку мыши. На экране – временное меню. Это меню позволяет перейти к редактированию объявления или определения метода, просмотреть те строки исходного текста приложения, в которых вызывается метод, получить список функций и методов, вызываемых выбранным методом. Например, чтобы открыть в редакторе файл, в котором объявляется метод, и перейти к его редактированию, выбираем из меню метода пункт «*Go to Definition*».

Возможности *Classview* можно использовать даже на этапе отладки приложения. Так, из временного меню метода можно установить точку прерывания непосредственно на начало метода. Для этого выбирают из меню «*Set Breakpoint*».

4 Редактирование элементов данных класса. Временное меню данных класса представляет собой сокращенный вариант временного меню метода.

Задание к лабораторной работе

По изложенной выше методике работы с *ClassView* создать проект типа *Dialog based*, позволяющий вычислять и отображать два значения *Z*1 и *Z*2 (в случае успеха они должны совпасть).

Варианты заданий

1 $z_1 = \dfrac{\sqrt{2b + 2\sqrt{b^2 - 4}}}{\sqrt{b^2 - 4} + b + 2}, \quad z_2 = \dfrac{1}{\sqrt{b+2}}.$

2 $z_1 = \dfrac{x^2+2x-3+(x+1)\sqrt{x^2-9}}{x^2-2x-3+(x-1)\sqrt{x^2-9}}$, $z_2 = \sqrt{\dfrac{x+3}{x-3}}$.

3 $z_1 = \dfrac{\sqrt{(3m+2)^2-24m}}{3\sqrt{m}-2/\sqrt{m}}$, $z_2 = -\sqrt{m}$.

4 $z_1 = \left(\dfrac{a+2}{\sqrt{2a}} - \dfrac{a}{\sqrt{2a}+2} + \dfrac{2}{a-\sqrt{2a}}\right) \cdot \dfrac{\sqrt{a}-\sqrt{2}}{a+2}$, $z_2 = \dfrac{1}{\sqrt{a}+\sqrt{2}}$.

5 $z_1 = \left(\dfrac{1+a+a^2}{2a+a^2} + 2 - \dfrac{1-a+a^2}{2a-a^2}\right)^{-1} (5-2a^2)$, $z_2 = \dfrac{4-a^2}{2}$.

6 $z_1 = \dfrac{(m-1)\sqrt{m}-(n-1)\sqrt{n}}{\sqrt{m^3 n}+nm+m^2-m}$, $z_2 = \dfrac{\sqrt{m}-\sqrt{n}}{m}$.

7 $z_1 = \dfrac{\sqrt{2m+2\sqrt{m^2-4}}}{m+\sqrt{m^2-4}+2}$, $z_2 = \dfrac{1}{\sqrt{m+2}}$.

8 $z_1 = \dfrac{(x+1)\sqrt{x^2-9}+x(x+2)-3}{(x-1)\sqrt{x^2-9}+x^2-2x-3}$, $z_2 = \sqrt{\dfrac{x+3}{x-3}}$.

9 $z_1 = \left(2 + \dfrac{1+x+x^2}{2x+x^2} - \dfrac{1-x+x^2}{2x-x^2}\right)^{-1} (5-2x^2)$, $z_2 = \dfrac{4-x^2}{2}$

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

11 $z_1 = \dfrac{\sqrt{(3x+2)^2-24x}}{3\sqrt{x}-2/\sqrt{x}}$, $z_2 = -\sqrt{x}$.

12 $z_1 = \dfrac{(a-1)\sqrt{a}-(b-1)\sqrt{b}}{\sqrt{a^3 b}+ba+a^2-a}$, $z_2 = \dfrac{\sqrt{a}-\sqrt{b}}{a}$.

13 $z_1=\dfrac{\sqrt{2\sqrt{x^2-4}+2x}}{x+\sqrt{x^2-4}+2}$, $z_2=\dfrac{1}{\sqrt{x+2}}$.

14 $z_1=\left(\dfrac{m+2}{\sqrt{2m}}+\dfrac{2}{m-\sqrt{2m}}-\dfrac{m}{\sqrt{2m}+2}\right)\cdot\dfrac{\sqrt{m}-\sqrt{2}}{m+2}$, $z_2=\dfrac{1}{\sqrt{m}+\sqrt{2}}$.

15 $z_1=\dfrac{(x-1)\sqrt{x}-(y-1)\sqrt{y}}{\sqrt{x^3y}+xy+x^2-x}$, $z_2=\dfrac{\sqrt{x}-\sqrt{y}}{x}$.

Контрольные вопросы

1 Перечислите возможности мастера ClassView.
2 Назовите этапы добавления к проекту нового класса, данных, методов.
3 Какие мастера проектов существуют в VC++?

Лабораторная работа №9
Интерфейс графических устройств Windows

Цель работы – изучить методы работы с графическим интерфейсом (*GDI*). Научиться строить простые геометрические фигуры.

Теоретические сведения

Любая программа Windows выводит информацию на доступное графическое устройство с помощью множества специальных функций, образующих так называемый GDI-интерфейс (от англ. graphics device interface). GDI-интерфейс является аппаратно-независимым. Windows абстрагирует вывод на графическое устройство от технических особенностей его работы. Таким образом, графическое устройство в Windows – это монитор, принтер, плоттер.

В MFC интерфейсу GDI соответствует класс CDC. Данный класс содержит методы для вывода разнообразной графической информации: прямоугольников, линий, эллипсов, текста. Он также позволяет устанавливать цвет и масштаб изображений. Методы, обеспечивающие вывод, например, на экран монитора, используют контекст устройства, который в приложении на базе MFC является объектом класса CDC. Таким образом, класс CDC является базовым классом контекста устройств, содержит ряд атрибутов контекста устройств и методы, управляющие ими. И для того, чтобы использовать контекст устройства, необходимо создать объект CDC.

На практике пользуются следующими классами, производными от CDC:

– CClientDC – управляет контекстом экрана монитора, связанного с клиентской областью окна;

– CWindowDC – управляет контекстом экрана монитора, связанного с клиентской областью окна и с его системной частью.

Ниже приведен список часто используемых функций класса CDC:

Ellipse – рисует эллипс;

Polygon – многоугольник;

Rectangle –прямоугольник;

RoundRect – прямоугольник со скругленными углами;

Chord – сегмент эллипса;

Pie – сектор эллипса;

LineTo – линию.

При рисовании линии при помощи функции LineTo рисование осуществляется с текущей позиции пера. Для ее изменения существует функция MoveTo.

Для управления цветом фона и цветом линий существуют специальные объекты GDI. Цветом фона управляет объект – «кисть», которому соответствует класс CBrush, а цветом линий управляет объект – «перо», которому соответствует класс CPen.

Рассмотрим создание объекта «перо». Создание может происходить в два этапа: сначала объявляется переменная типа CPen, потом вызывается функция CreatePen, в которую предают тип линии, ширину и цвет.

Для указания типов линии используют константы:

PS_SOLID – сплошная линия;

PS_DASH – штриховая линия;

PS_DASHDOT – штрихпунктирная линия.

После создания объекта «перо» его необходимо выбрать в контекст устройства рисования. Для этого вызывают метод SelctObject класса CDC, передавая ему указатель на объект «перо».

Рассмотрим типовой пример: Написать программу, после нажатия кнопки – выводящую картинку усеченного эллипса красного цвета с синими гранями (рисунок 9.1).

Для решения задачи необходимо создать приложение на основе диалогового окна.

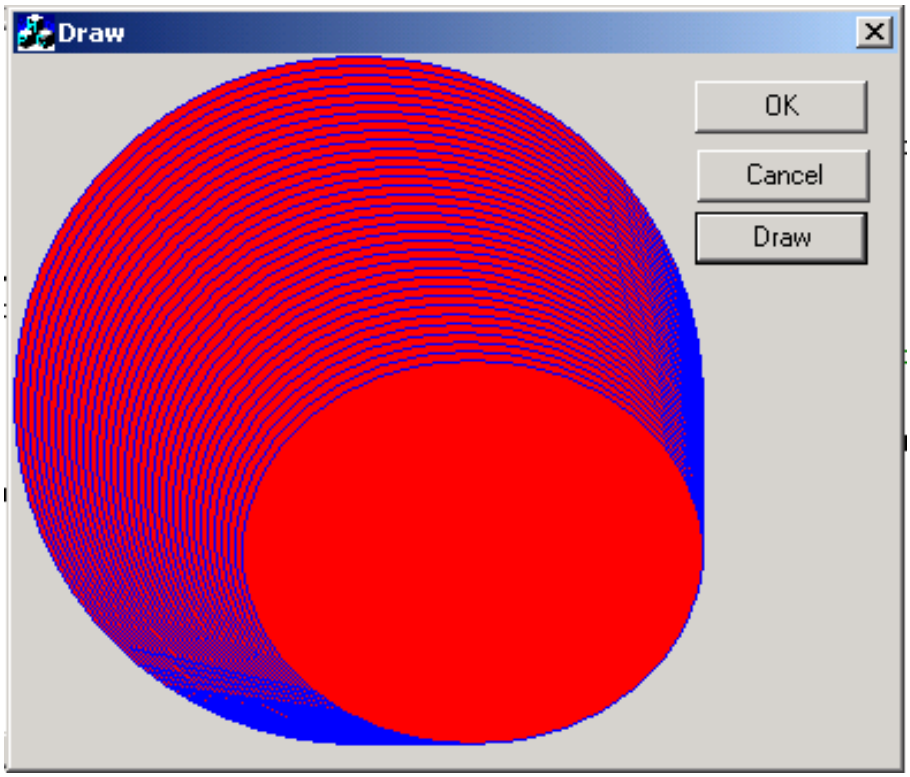

Рисунок 9.1 – Результат программы

Затем поместить на форму кнопку, при нажатии которой будет выполняться функция рисования.

Обработчик нажатия кнопки приведен ниже:

```
void CDraw::OnButton1()
{
// TODO: Add your control notification handler code here
```

```
CClientDC dc(this); //объявляем класс, производный от класса CDC
//  класс CClientDC   предназначен для рисования в диалоговом окне
        CBrush br;// объявляем класс кисти
br.CreateSolidBrush(RGB(255,0,0)); //создаем сплошную красную кисть
dc.SelectObject(&br);// выбираем кисть в контекст устройства
        CPen p; //объявляем класс пера
        p.CreatePen(PS_SOLID    ,1,RGB(0,0,255));
//создаем сплошное перо, толщиной 1, синего цвета
dc.SelectObject(&p); // выбираем перо в контекст устройства
        for(int x=0,int y=0;x<100;x+=3,y+=4)
dc.Ellipse(x,y,300,300); //рисуем эллипс с заданными координатами
}
```

Задание к лабораторной работе

Требуется нарисовать одну из фигур, представленных на рисунке 9.2. в соответствии со своим вариантом. Фигуры должны быть окрашены в произвольные цвета.

Варианты заданий

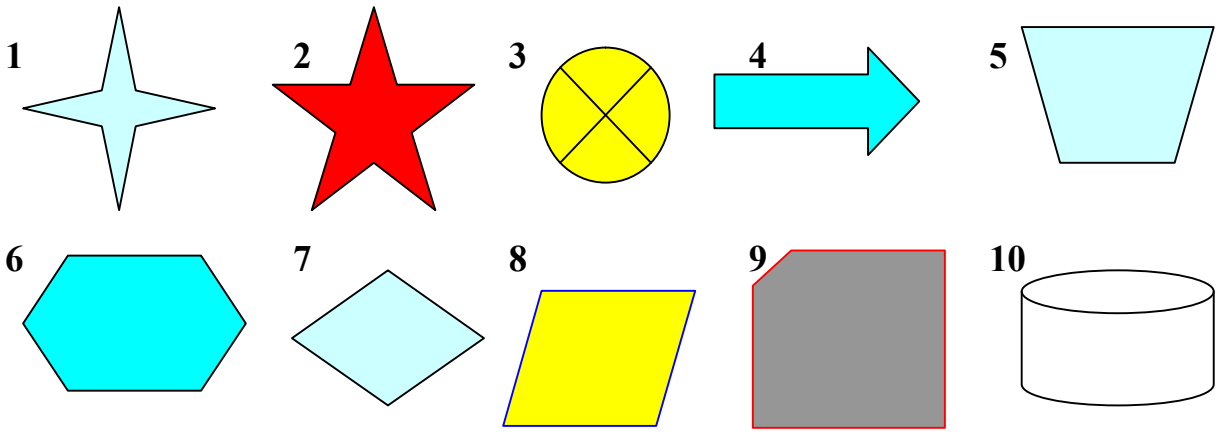

Рисунок 9.2– Заданные фигуры

Контрольные вопросы

1 Дайте характеристику GDI-интерфейсу
2 В чем заключается роль класса CDC в MFC?
3 Назовите основные функции класса CDC для рисования фигур, линий.

Лабораторная работа №10
Элементы управления радиокнопки (Radio Button) и переключатели (Check Box)

Цель работы – изучить принципы работы элементов управления: радиокнопок (*Radio Button*) и переключателей (*Check Box*), позволяющих устанавливать тот или иной режим работы.

Теоретические сведения

Элементы управления Radio Button и Check Box

Элемент *Check Box* организует кнопку независимого переключателя, с помощью которой пользователь может указать свое решение типа да/нет. В программе состояние кнопки может быть связано со значением булевской переменной, которая проверяется с помощью оператора if.

Элемент *Radio Button* организует группу кнопок – зависимых переключателей. При нажатии одной из кнопок группы все остальные кнопки отключаются. В программе элемент *Radio Button* связывается с переменной типа int, при этом в программу будет передаваться номер включенной кнопки, который анализируется с помощью оператора switch или if. Номер первой кнопки равен 0, второй –1 и т.д. Но для этого нужно в тексте программы найти объявление привязанной к этим кнопкам переменной и задать ее нулевое начальное значение, так как по умолчанию система программирования установит для нее значение –1.

Пример реализации программы

Задание: В поля ввода ввести 2 целых числа, «Радиокнопка» выбирает действие: или суммирование или разность. «Переключатель» при не активности устанавливает вывод в одну строку при многократном выполнении приложения, а при его активизации результаты выводятся списком. Интерфейс программы приведен на рисунке10.1.

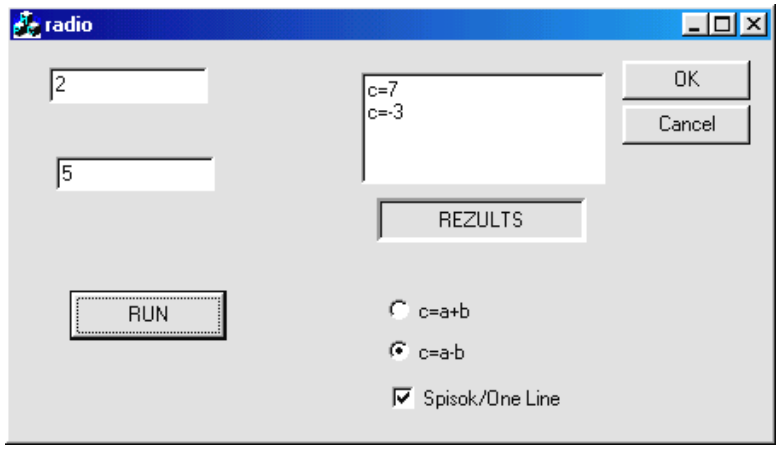

Рисунок 10.1 – Интерфейс программы

Для решения задачи необходимо сделать следующие действия:

1) создать при помощи *MFCAppWizard(exe)* новый проект (предварительно выбрав путь и имя – например *Lab2*) типа *Dialog based*;

2) используя панель *Controls*, нанести на диалоговую панель 3 элемента типа *Edit Box*, 4 – *Static Text*, 1 – *Button*, 2- *Radio Button*, 1 – *Check Box* и установить нужные свойства и надписи (для *Radio Button* с идентификатором *IDC_RADIO1* нужно установить свойство *Group*);

3) при помощи *ClassWizard* через закладку *Member Variables* связать поле *Edit1* с переменной m_a тип: int, *Edit2* - с m_b тип: int, *Edit3* - с m_c тип: *CString*. Для *Edit3* установить в закладке свойств *Styles*: свойство *Multiline*;

4) при помощи *ClassWizard* через закладку *Member Variables* связать идентификатор «переключателя» *IDC_CHECK1* с переменной m_s тип: BOOL, а радиокнопки *IDC_RADIO1* – с переменной m_r тип: int. Кроме того, в самом начале метода BOOL Clab2Dlg::OnInitDialog() нужно установить значение переменной m_r =0;

5) теперь необходимо активировать кнопку «*Выполнить*». Для этого: выполнить щелчок правой клавишей мыши в любом месте диалоговой панели и в меню выбрать *ClassWizard*, выбрать закладку *Message Maps*, связать идентификатор *IDC_Button1* с событием *BN_CLICKED* (один щелчок на кнопке) и добавить обработчик на данное событие: последовательно *Add Function/OK/ Edit Code*, среда добавит в конец файла Lab2Dlg.cpp пустой обработчик:

```
void Clab2Dlg::OnButton1()
{
        // TODO: Add your control notification handler code here
}
```

и, как и ранее, вместо этой фразы:
добавляем код решения поставленной задачи:

```
{
int r; CString str;
UpdateData(TRUE);
if (m_r==0) r=m_a+m_b;      //если выбрана первая радиокнопка –
//«суммирование»
if (m_r==1) r=m_a-m_b; // если выбрана вторая
// радиокнопка – «вычитание»
if (m_s)
// если переключатель активен – вывод списком
                        {
                                str.Format("c=%d%c%c",r,13,10);
                                m_c=m_c+str;
                        }
else
// если переключатель не активен – вывод в одну строку
```

```
        m_c.Format("c=%d",r);
UpdateData(FALSE);
}
```

Задание к лабораторной работе

В индивидуальном задании составить программу для вычисления составной функции в соответствии со своим вариантом. Требуется самостоятельно выбрать необходимое количество исходных данных для того, чтобы в программе выполнялись все возможные ветви алгоритма. Перед выводом полученного результата программа должна сообщать о ветви, для которой он получен. В качестве $f(x)$ использовать по выбору: «радиокнопки» функции $cos(x)$, $sin(x)$, $tg(x)$. Программа должна через «переключатели» запоминать или не запоминать min и max найденные значения. Вариант интерфейса приведен на рисунке 10.2.

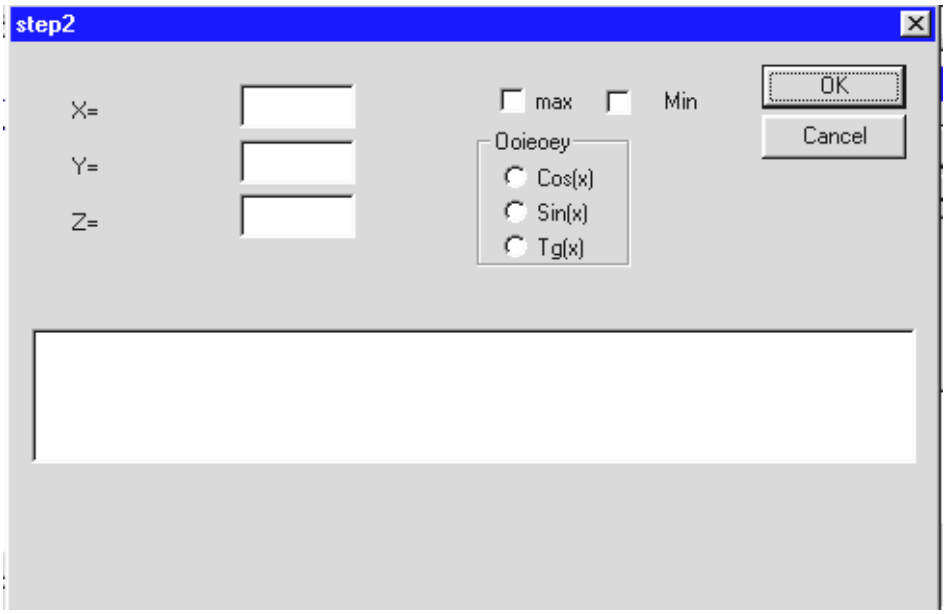

Рисунок 10.2 – Вариант интерфейса

Варианты заданий

1 $a=\begin{cases}(f(x)+y)^2-\sqrt{f(x)y}, & xy>0\\ (f(x)+y)^2+\sqrt{|f(x)y|}, & xy<0\\ (f(x)+y)^2+1, & xy=0.\end{cases}$

2 $c=\begin{cases}f(x)^2+y^2+\sin(y), & x-y=0\\ (f(x)-y)^2+\cos(y), & x-y>0\\ (y-f(x))^2+\mathrm{tg}(y), & x-y<0.\end{cases}$

3 $e=\begin{cases} i\sqrt{f(x)}, i-\text{нечетное}, x>0 \\ i/2\sqrt{|f(x)|}, i-\text{четное}, x<0 \\ \sqrt{|if(x)|}, \text{иначе}. \end{cases}$

4 $s=\begin{cases} e^{f(x)}, & 1<xb<10 \\ \sqrt{|f(x)+4*b|}, & 12<xb<40 \\ bf(x)^2, & \text{иначе}. \end{cases}$

5 $l=\begin{cases} 2f(x)^3+3p^2, & x>|p| \\ |f(x)-p|, & 3<x<|p| \\ (f(x)-p)^2, & x=|p|. \end{cases}$

6 $m=\dfrac{\max(f(x),y,z)}{\min(f(x),y)}+5.$

7 $p=\dfrac{|\min(f(x),y)-\max(y,z)|}{2}.$

8 $b=\begin{cases} \ln(f(x))+(f(x)^2+y)^3, & x/y>0 \\ \ln|f(x)/y|+(f(x)+y)^3, & x/y<0 \\ (f(x)^2+y)^3, & x=0 \\ 0, & y=0. \end{cases}$

9 $d=\begin{cases} (f(x)-y)^3+\text{arctg}(f(x)), & x>y \\ (y-f(x))^3+\text{arctg}(f(x)), & y>x \\ (y+f(x))^3+0{,}5, & y=x. \end{cases}$

10 $g=\begin{cases} e^{f(x)-|b|}, & 0{,}5<xb<10 \\ \sqrt{|f(x)+b|}, & 0{,}1<xb<0{,}5 \\ 2f(x)^2, & \text{иначе}. \end{cases}$

11 $j=\begin{cases} \sin(5f(x)+3m|f(x)|), & -1<m<x \\ \cos(3f(x)+5m|f(x)|), & x<m \\ (f(x)+m)^2, & x=m. \end{cases}$

12 $k=\begin{cases} \ln(|f(x)|+|q|), & |xq|>10 \\ e^{f(x)+q}, & |xq|<10 \\ f(x)+q, & |xq|=10 \end{cases}$

13 $n=\dfrac{\min(f(x)+y,y-z)}{\max(f(x),y)}.$

14 $q=\dfrac{\max(f(x)+y+z,xyz)}{\min(f(x)+y+z,xyz)}.$

Контрольные вопросы

1 В чем заключается принцип работы элементов управления – радиокнопок (*Radio Button*)?

2 В чем заключается принцип работы элементов управления – переключателей (*Check Box*)?

3 Для чего служит мастер *ClassWizard*?

Лабораторная работа №11
Работа с элементом управления «Окно списка» (ListBox)

Цель работы – научиться использовать элемент управления *ListBox*, основные методы класса *CListBox*, а также возможности контроля правильности ввода исходных данных.

Теоретические сведения

Класс CListBox

Класс *CListBox* обеспечивает функционирование окна списка. В окне списка могут отображаться различные элементы, которые пользователь может просматривать и выделять.

Окно списка имеет два режима выделения элементов:

– единичный выбор, т. е. может быть выделен только один элемент списка;

– множественный выбор, т. е. в списке одновременно могут быть выделены несколько элементов.

Окно списка может содержать как один столбец, так и несколько столбцов с элементами.

Основные методы данного класса

Для добавления строки в список в классе *CListBox* объявлена функция:

int AddString(LPCTSTR lpszItem);

Здесь параметр lpszItem – указатель на нуль-терминированную строку, добавляемую в список. Функция возвращает номер строки в списке, перед которой добавлялся текст, причем первая строка имеет нулевой индекс.

Кроме того, данный класс имеет следующие методы:

– int GetCount() const; при успехе метод возвращает количество элементов в окне списка;

– void SetColumn(int Wingth); метод устанавливает ширину всех колонок списка;

– int GetCutSel(); метод возвращает индекс выделенного элемента списка;

– int DeleteString(UINT n); метод удаляет строку с индексом n;

– int InsertString(int n, LPCTSTR lpszItem*)*; метод вставляет в указанное место n строку; если индекс n равен –1, то строка будет добавлена в конец списка.

Если в процессе работы функций возникла ошибка, то возвращается величина *LB_ERR*; если эта ошибка была связана с нехваткой памяти для хранения новой строки, то возвращается величина *LB_ERRSPASE*. Для очистки всего списка объявлена функция:

void ResetContent( );

Эта функция не получает и не возвращает параметры.

Пример реализации программы

Задание: Написать и отладить программу, которая выводит таблицу функций

$$S(x)=\sum_{k=0}^{n}\frac{x^i}{i!} \text{ и } Y(x)=\exp(x)$$

для *x* изменяющихся от *X*1 до *X*2 с шагом *H*. Программа должна выглядеть, как представлено на рисунке 11.1.

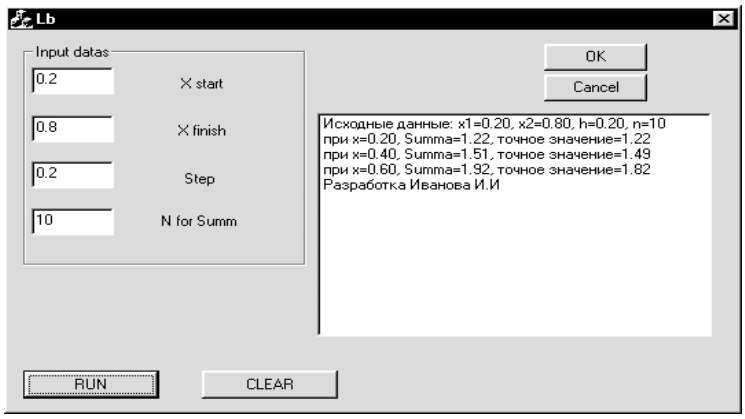

Рисунок 11.1– Результат работы программы

Для решения задачи необходимо проделать следующие действия:

1) создать при помощи *MFCAppWizard(exe)* новый проект (предварительно выбрав путь и имя) типа *Dialog based*;

2) используя панель *Controls*, нанести на Dialog 4 элемента типа *Edit Box* и *Static Text* (объединив их элементом Group Box), 2 – Buttons и 1 – *ListBox* для вывода результатов. Установить нужные свойства и надписи;

3) при помощи *ClassWizard* через закладку *Member Variables* связать поле *Edit1* с переменной m_x1, выполнив команду *ClassWizard/Member Variables*, сделать щелчок на *IDC_EDIT1*, щелчок на *Add Variable* и в появившейся закладке в поле name: m_x1 тип: double щелчок на *OK* установить диапазон: Minimum Value: 0.1; Maximum Value: 0.3;

По аналогии связать:

Edit2 – с m_x2 тип: double, диапазон: 0,7 –0,9;

Edit3 – с m_h тип: double, диапазон: 0,1 –0,2;

Edit4 – с m_n тип: int, диапазон: 5 –20.

Элемент *ListBox* связать с m_l, назначив *Category: Control*, установив тем самым для данного объекта тип *CListBox*, т. е. для данного объекта можно вызывать методы по всей цепи иерархии данного класса *MFC*. Так, для стирания

окна вывода в обработчик щелчка на кнопке «*CLEAR*» можно вызвать следующий метод: m_l.ResetContent().

В обработчике щелчка на кнопке «RUN» ввести в класс диалога переменные m_x1, m_x2, m_h, m_n, при помощи циклических алгоритмов (внешний – по x, а внутренний цикл – по параметру i для вычисления суммы для текущего аргумента) написать текст программы решения поставленной задачи.

Для вывода информации о разработчике и введенных исходных данных необходимо воспользоваться методом класса *ListBox – AddString*:

```
UpdateData(TRUE);
str.Format("Разработка Иванова И.И");
m_l.AddString(str);
UpdateData(FALSE);
str.Format("Исходные данные: x1=%.2lf, x2=%.2lf, h=%.2lf, n=%2d",
m_x1,m_x2,m_h,m_n);
m_l.AddString(str);
UpdateData(FALSE);
```

Участок вывода текущей строки таблицы в окно результатов может быть таким:

```
str.Format("при x=%.2lf, Summa=%.2lf, точное значение=%.2lf",x,s,y);
m_l.AddString(str);
UpdateData(FALSE);
```

4) находясь в режиме конструктора диалога, командой *Layout/Tab Order* установить очередность перехода между элементами управления:

1 – *Edit1*; 2 – *Edit2*; 3 – *Edit3*; 4 – *Edit4*; 5 – *Button RUN*; 6 – *ListBox*; 7 – *Button CLEAR*; 8 – *Button OK*; 9 – *Button Cancel*, а для кнопки «*RUN*» установить свойство «*Default*»;

5) работа с программой.

Запустить проект и в появившемся диалоговом окне щелкнуть на «*OK*».

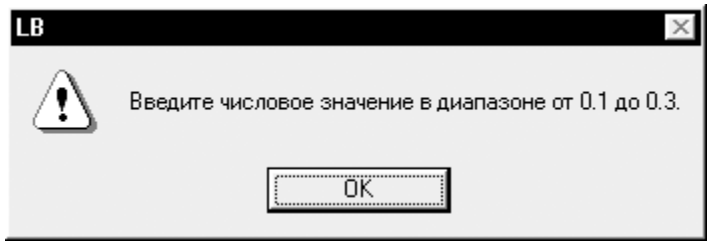

Рисунок 11.2 – Окно подсказки

Появится *MessageBox* с подсказкой о диапазоне вводимого x_1 (рисунок 11.2): Щелкнуть «*OK*», ввести начальное значение аргумента x (поле с подсветкой) и

опять щелкнуть на «*OK*». По аналогии ввести значения для x_2, h, n. Для выполнения задания щелкнуть на «*RUN*». Щелчок на кнопке «*CLEAR*» стирает значения в окне вывода.

Варианты заданий

1 $S(x)=\sum_{k=0}^{n}\frac{\ln^k 3}{k!}x^k,\ Y(x)=3^x-1.$

2 $S(x)=\sum_{k=1}^{n}\frac{\cos(kx)}{k},\ Y(x)=-\ln\left|2\sin\frac{x}{2}\right|.$

3 $S(x)=\sum_{k=0}^{n}(-1)^k\frac{x^{2k+1}}{(2k+1)!},)\ Y(x)=\sin(x).$

4 $S(x)=\sum_{k=0}^{n}\frac{x^k}{k!},Y(x)=e^x$

5 $S(x)=\sum_{k=1}^{n}(-1)^{k+1}\frac{\sin(kx)}{k},Y(x)=\frac{x}{2}.$

6 $S(x)=\sum_{k=0}^{n}\frac{\cos(\frac{k\pi}{4})}{k!}x^k,\ Y(x)=e^{x\cos\frac{\pi}{4}}\cos(x\sin(\frac{\pi}{4})).$

7 $S(x)=\sum_{k=0}^{n}(-1)^k\frac{x^{2k}}{(2k)!},\quad Y(x)=\cos(x).$

8 $S(x)=\sum_{k=1}^{n}x^k\sin^k(\frac{\pi}{4}),Y(x)=\frac{(x\sin(\pi/4))}{1-2x\cos(\frac{\pi}{4})+x^2}.$

9 $S(x)=\sum_{k=0}^{n}\frac{x^{4k+1}}{4k+1},\ Y(x)=\frac{1}{4}\ln\frac{1+x}{1-x}+\frac{1}{2}\text{arctgx}.$

10 $S(x)=\sum_{k=0}^{n}\frac{\cos(kx)}{k!},\quad Y(x)=e^{\cos x}\cos(\sin(x)).$

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

12 $S(x)=\sum_{k=1}^{n}\frac{x^k\cos\frac{k\pi}{3}}{k},Y(x)=-\frac{1}{2}\ln(1-2x\cos\frac{\pi}{3}+x^2).$

13 $S(x)=\sum_{k=0}^{n}\frac{1}{2k+1}\left(\frac{X-1}{X+1}\right)^{2K+1},\ Y(x)=\frac{1}{2}\ln(x).$

14 $S(x)=\sum_{k=1}^{n}(-1)^k\frac{\cos(kx)}{k^2},Y(x)=\frac{1}{4}\left(x^2-\pi^2/3\right).$

15 $S(x)=\sum_{k=1}^{n}(-1)^{k+1}\frac{x^{2k+1}}{4k^2-1}, Y(x)=\frac{1+x^2}{2}\text{arctg}(x)-x/2$.

Контрольные вопросы

1 В чем заключается роль класса *CListBox* в MFC?
2 Назовите основные методы класса *CListBox*.
3 Перечислите основные этапы создания списка.

Лабораторная работа №12
Работа с элементом управления «Комбинированный список» (ComboBox)

Цель работы – научиться использовать элемент управления *ComboBox*, изучить основные методы класса *CComboBox*, предназначенным для управления данным элементом.

Теоретические сведения

Класс CComboBox

Элемент управления *ComboBox* представляет собой список *ListBox* в комбинации с редактором *Edit* или надписью *Static*. Данный элемент называют комбинированным окном. Окно при этом может иметь два состояния: как распахнутое (которое отображается постоянно), так и свернутое (отображается при щелчке на элементе «стрелка вниз»). Элементы списка в окне можно выбирать, и тогда они отображаются или в окне редактирования, или в окне статического элемента.

Три модификации компонента определяются его свойством «*Type*» в закладке окна свойств *Style*. В таблице 12.1 приведено сравнение этих трех стилей.

Таблица 12.1 – Сравнение стилей

Стиль	Когда отображается *ListBox*?	Что отображается, Edit или Static?
Simple	Всегда	*Edit*
Drop-down	После нажатия кнопки со стрелкой	*Edit*
Drop-down list	После нажатия кнопки со стрелкой	*Static*

Для управления элементом *ComboBox* в библиотеке MFC существует класс CComboBox. Чтобы добавить новую строку в выпадающий список, нужно использовать функцию: int AddString(LPCTSTR lpszItem). Здесь параметр lpszItem – указатель на нуль-терминированную строку, добавляемую в список. Для очистки списка служит функция:void ResetContent( );

В списке элемента *ComboBox* пользователь может выбирать элементы, и для того, чтобы определить, какой элемент был выбран, нужно воспользоваться функцией:int GetCurSel( ) const;

Эта функция не принимает параметров и возвращает индекс выбранного элемента, причем индекс начинается с нуля.

Если необходимо из списка по заданному индексу элемента определить его содержимое, то можно воспользоваться следующей функцией:

void GetLBText(int nIndex, CString& rString) const;

Здесь nIndex – индекс элемента, rString – сторока, куда помещается текст.

Пример написания программы

Задание. В поле ввода вводятся последовательно строки и кнопкой «*Add*» добавляются в поле *ComboBox* (рисунок 12.1). Затем щелчком левой клавиши мыши выбирается соответствующая строка (выбранная строка отобразится в верхнем окошке *ComboBox*). Щелчок на кнопке «*RUN*» произведет подсчет пробелов в выделенной строке и отобразит результат в поле вывода.

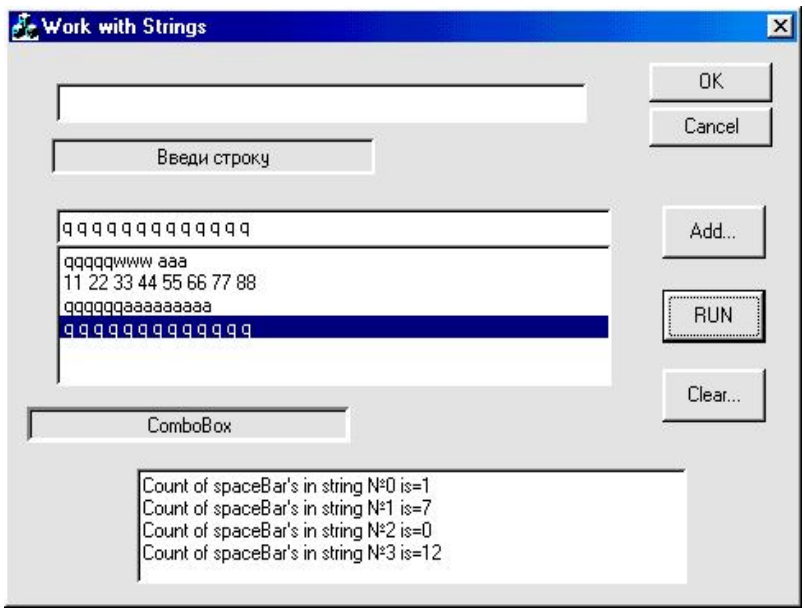

Рисунок 12.1 – Результат работы программы

Для решения задачи необходимо проделать следующие действия:

1) создать при помощи *MFCAppWizard(exe)* новый проект типа *Dialog based* (предварительно выбрав путь и имя);

2) используя панель *Controls*, нанести на *Dialog* 2 элемента типа *Edit Box* (поле ввода строки и поле вывода), используя *Static Text* , нанести на окно нужные пояснения, 3 – *Buttons* и 1 – *ComboBox* для вывода добавляемых строк. Установить нужные свойства и надписи, для *ComboBox* – в закладке *Styles: Type->Simple* и активизировать *Vertical scroll*;

3) при помощи *ClassWizard* через закладку *Member Variables*: связать поле *Edit1* с переменной m_a (тип CString): *ClassWizard/Member Variables*, сделать щелчок на IDC_EDIT1, щелчок на *Add Variable* в закладке name: m_a type: CString и нажать «*OK*»; для поля вывода *IDC_EDIT2:* name: m_a type: Cstring и в свойствах: *Multiline* (можно добавить вертикальный скроллинг). Элемент ComboBox связать с m_c, установив *Category*: *Control* для данного объекта типа *CComboBox***,** т. е. для данного объекта можно вызывать методы по всей цепи иерархии данного класса *MFC*;

4) связать через *ClassWizard/Message maps* кнопки с событием «один щелчок» и активировать обработчики. Так, для стирания окон вывода в обработчике щелчком на кнопке «Clear» можно вызвать следующий метод:

m_c.ResetContent(); и m_b=””.

Обработчик кнопки «*Add*» добавляет в поле вывода *ComboBox* строку, набранную в поле ввода, одновременно очищая последнюю.

Текст программы:

```
UpdateData(TRUE);
m_c.AddString(m_a);
m_a="";
UpdateData(FALSE);
```

Обработчик кнопки «*RUN*» выводит в поле вывода номер выбранной строки и количество пробелов в ней. Текст программы следующий:

```
CString s1;
int i,j=0;
char str[25];
CComboBox *p=(CComboBox *)GetDlgItem(IDC_COMBO1);
i=p->GetCurSel();
if (i==LB_ERR)
AfxMessageBox("String No SELECT!!!");
else
{
p->GetLBText(i, str);
AfxMessageBox(str);
}
```

Таким образом, объявляем указатель p на элемент *ComboBox*, через него вызываем метод GetCurSel() для определения номера выбранной строки. Далее контролируем: если выбор строки не сделан, то получаем *AfxMessageBox* с сообщением об этом. Если же строка выделена, методом *GetLBText* копируем выделенную строку списка в str и отображаем ее в *AfxMessageBox*.

Данный участок программы решает поставленную задачу, т.е. подсчитывает число пробелов в выделенной строке:

```
int k=strlen(str);
for(int i1=0;i1<k;i1++)
if (str[i1]==' ') j++;
s1.Format("Count of spaceBar's in string №%d is=%d%c%c",i,j,13,10);
m_b=m_b+s1;
UpdateData(FALSE);
}
```

В начале текста программы нужно подключить заголовочный файл string.h;

5) находясь в режиме конструктора диалога, командой *Layout/Tab Order* установить необходимую очередность перехода между элементами управления, поле ввода должно иметь №1.

Варианты заданий

1 Дана строка, состоящая из групп нулей и единиц. Найти и вывести на экран группу с пятью символами.

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

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

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

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

6 Дана строка, состоящая из букв, цифр, запятых, точек, знаков «+» и «-». Вывести подстроку, которая соответствует записи целого числа (т.е. начинается со знаков «+» и «-» и внутри подстроки нет букв, запятых и точек).

7 Дана строка, состоящая из букв, цифр, запятых, точек, знаков «+» и «-». Вывести подстроку, которая соответствует записи вещественного числа с фиксированной точкой.

8 Дана строка, состоящая из букв, цифр, запятых, точек, знаков «+» и «-». Вывести подстроку, которая соответствует записи вещественного числа с плавающей точкой.

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

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

Контрольные вопросы

1 Чем отличается элемент управления *ComboBox* от *ListBox*?
2 Назовите основные методы класса *CComboBox*.
3 Назовите 3 модификации элемента *ComboBox*.

Лабораторная работа №13
Элементы управления «Шкала индикации» (Progress) и «Маркер» (Slider). Модальные диалоговые окна

Цель работы – изучить принципы работы элементов управления *Progress* и *Slider*, получить навыки по самостоятельному созданию модальных диалоговых окон.

Теоретические сведения

Элементы управления Progress и Slider

Для отображения завершенности какого-либо длительного процесса в Windows существует элемент Progress (шкала индикации*).* Этот элемент управления используется для наглядного представления некоторой изменяемой характеристики. Progress представляет собой прямоугольник, который заполняется системным цветом в направлении слева направо по мере выполнения некоторой операции.

Класс CProgressCtrl обеспечивает управление состоянием индикатора. Для установки текущей позиции индикатора в классе CProgressCtrl есть следующая функция: int SetPos(int *nPos*).

Здесь nPos – это новая позиция линейного индикатора, функция возвращает его старую позицию. По умолчанию верхний предел значений равен 100 (соответствует полностью заполненному индикатору), а нижний – 0 (соответствует полностью не заполненному индикатору).

При необходимости для изменения пределов можно воспользоваться функцией: void SetRange(short nLower, short nUpper).

Здесь nLower – значение нижнего предела, nUpper – новое значение верхнего предела.

Элемент управления Slider (маркер, линейный регулятор) представляет собой линейный регулятор, содержащий бегунок и метки шкалы. Этот элемент управления может использоваться для выбора дискретных значений из некоторого диапазона, т. е. его, как правило, используют для плавного изменения значения в некотором диапазоне в зависимости от положения бегунка.

В MFC элементу линейного регулятора соответствует класс CSliderCtrl. Как и в классе CProgressCtrl для установки нового положения бегунка в классе CSliderCtrl есть функция SetPos, для установки диапазона допустимых значений служит функция SetRange. Когда требуется определить положение бегунка, вызывают функцию-член класса CSliderCtrl: int GetPos( ) const.

Кроме того, часто возникает необходимость запрограммировать обработчик сообщения *WM_HSCROLL*. Как правило, обработка данного сообщения сводится к определению нового положения бегунка (метод GetPos) и отображению его значения в другом элементе управления, например в диалоговой панели.

Модальные диалоговые окна

Существует два типа диалоговых окон: модальные и немодальные.

Модальные окна должны быть закрыты пользователем, для того чтобы приложение могло продолжать выполнять свои задачи. Немодальные окна позволяют пользователю продолжать выполнять другие задачи без закрытия диалога, поэтому модальные диалоги более распространены. Примером модального окна может служить стандартный диалог выбора файлов.

Для создания модального диалогового сначала создают ресурс диалогового окна и с помощью *ClassWizard* создают новый класс диалогового окна, порожденного от класса CDialog. Для вызова диалогового окна используют функцию DoModal();. Для возвращения значений из модального диалогового окна в программу можно в классе диалогового окна объявить поля или функции.

Пример реализации программы

Задание*:* написать программу, вычисляющую значение интеграла функции $f(x) = x^2$ по следующей приближенной формуле:

$$I = \int_a^b f(x)dx = \sum_{i=1}^{N} f(x_i) * h;$$

где a,b – границы интегрирования;

N – число разбиений;

h – шаг, $h = (b\text{-}a)/N$, $x_i = h * i$.

Число разбиений должно задаваться с помощью элемента *Slider*, а скорость вычисления интеграла контролируется элементом *Progress*. Границы интегрирования необходимо задавать в модальном диалоговом окне.

Для решения задачи необходимо создать проект на основе диалогового окна (имя: *CLabProgressBar*), на втором шаге отключив опцию *About Box*. Используя панель *Controls*, нанести на заготовку формы – 1 элемент типа *Edit Box*, 2 – *Static Tex,t*; 2 – *Button*, 1 – *Slider*, 1– *Progress*. Название кнопки «*OK*» переименовать в «*Выход*».

С помощью мастера *ClassWizard* связать компонент *EditBox* с переменной m_res типа *double*, компонент *Progress* с переменной m_progress, выбрав ее тип как *CProgressCtrl*, а элемент управления *Slider* с переменной m_slider, выбрав ее тип как *CSliderCtrl*.

Для создания модального диалогового окна в закладке *ResourseView* окна *Workspase* во всплывающем меню требуется выбрать пункт *Insert*, после чего появится диалог *Insert Resource*, изображенный на рисунке 13.1.

В диалоге *Insert Resource* следует выбрать пункт *Dialog* и нажать кнопку *New*, после чего появится заготовка нового окна. Создать окно, аналогичное снимку на рисунке 13.2.

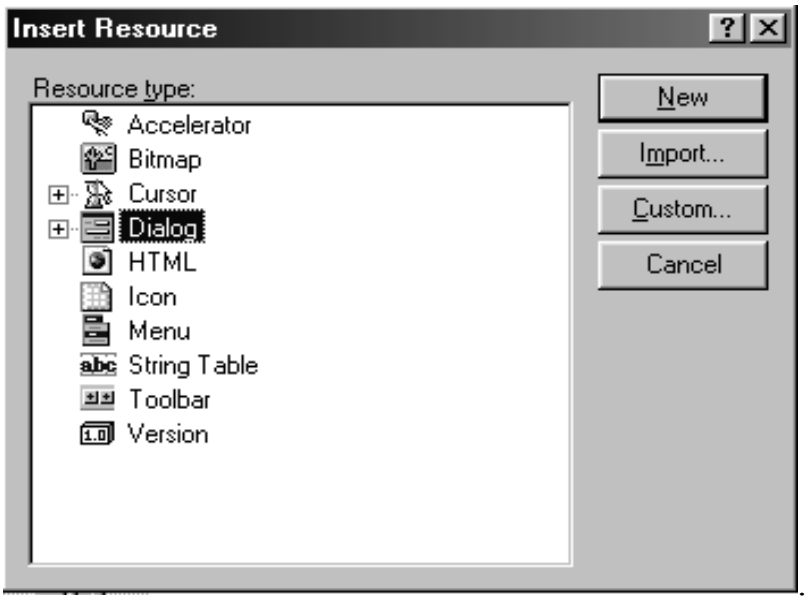

Рисунок 13.1 – Панель *Insert Resource*

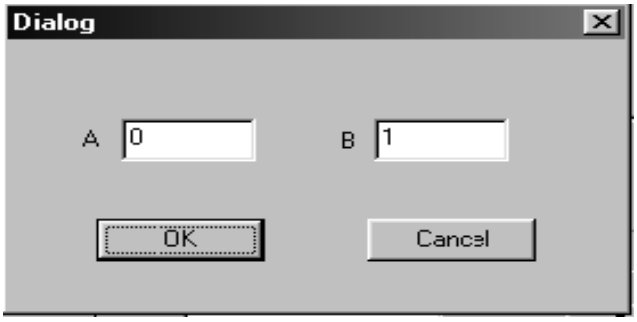

Рисунок 13.2 – Окно для ввода данных

После создания ресурса диалогового окна следует создать класс для работы с ним. Для этого в основном меню надо выбрать пункт Insert и далее *New Class*, после чего появится диалог *New Class* (рисунок 13.3).

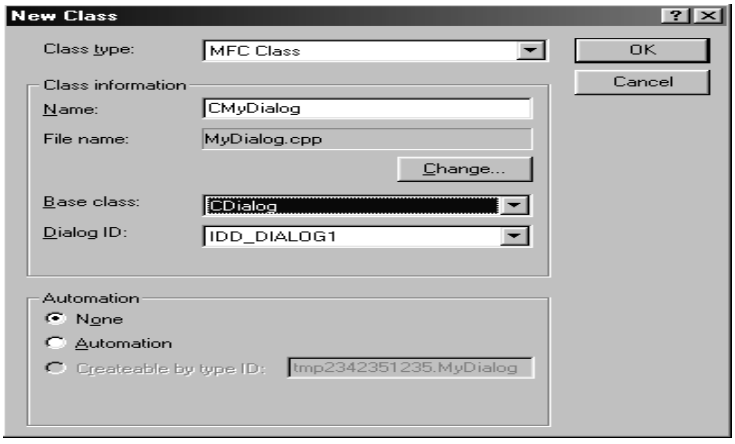

Рисунок 13.3 – Окно создания класса

В поле *Name* требуется ввести имя класса, например *CMyDialog*. В выпадающем списке *Base Class* требуется выбрать в качестве базового класса *CDialog*. В списке *Dialog ID* должен стоять идентификатор созданного ранее диалогового окна, по умолчанию для первого окна это значение *IDD_DIALOG1*.

После создания класса диалогового окна с помощью мастера Class Wizard связать поля ввода с переменными m_a и m_b типа double. Для создания экземпляра класса диалогового окна объявить поле в классе главного окна.

Далее сделать необходимые добавления в тексте файлов проекта.
Ниже приведен фрагмент заголовочного файла класса главного окна с необходимыми добавками.

```
// CLabProgressBarDlg dialog
#include "MyDialog.h" //подключили заголовочный файл класса
//модального диалога
class CLabProgressBarDlg : public CDialog
{// Construction
public:CLabProgressBarDlg(CWnd* pParent = NULL); // standard constructor
CMyDialog d; //объявили переменную класса модального диалога
// Dialog Data
        //{{AFX_DATA(CLabProgressBarDlg)
```

Далее в основной файл ClabProgressBarDlg.cpp в конце метода:

```
 BOOL CLabProgressBarDlg::OnInitDialog():
        // TODO: Add extra initialization here
        m_slider.SetRange(100,1000000);
// устанавливаем диапазон в элементе Slider
        m_slider.SetPos(50000);
//устанавливаем положение ползунка
return TRUE;  // return TRUE  unless you set the focus to a control
}
```

Тексты щелчков на кнопках:

```
void CLabProgressBarDlg::OnButton1()
{                               //обработчик кнопки Граничные условия
        d.DoModal();        //вызов модального окна
}void CLabProgressBarDlg::OnButton2()
{                                       //обработчик кнопки Пуск
        CWaitCursor w;    //класс для установки курсора – часы
        double s=0;
        double x=d.m_a;   //считывание из диалога нач. значения
        double h=(d.m_b-x)/m_slider.GetPos();  //вычисление h=(b-a)/N
        int i=0;
```

```
        while(x<d.m_b)
        {       double f=x*x;       //численение квадрата икса
                s+=f*h;              // вычисление суммы
                x+=h;
                m_progress.SetPos(100*i++/m_slider.GetPos());
        //установка положения в полосы в Progress
        }m_res=s;
        UpdateData(0);
}
```

Варианты заданий

Проверьте правильность выражений, вычислив интеграл:

1 $\int_0^{2\pi} \sin(x)dx = 0$

2 $\int_0^{1} (1+x)dx = 1,5$

3 $\int_0^{1} x^4 dx = 0,2$

4 $\int_0^{\pi} \cos(2x)dx = 0$

5 $\int_0^{1} x^4 dx = 0,2$

6 $\int_0^{1} \sqrt{x}dx = 2/3$

7 $\int_0^{1} x^3 dx = 0,25$

8 $\int_0^{1} x^3 dx = 0,25$

9 $\int_0^{1} x^9 dx = 0,1$

10 $\int_0^{1} 2xdx = 1$

Контрольные вопросы

1 Опишите принципы работы элементов управления *Progress* и *Slider*.
2 Назовите свойства модального и немодального диалоговых окон.
3 Опишите процесс создания модального диалогового окна.

Лабораторная работа №14
MDI-приложение. Работа с данными в архитектуре «Документ/Представление» («Document/View»)

Цель работы – использовать принципы архитектуры «Документ/Представление» для выборки и сохранения данных в файлах, а также взаимодействия элементов меню, панели инструментов и строки состояния в приложении.

Теоретические сведения

Архитектура «Документ/Представление»

В основе этой архитектуры лежат три глобальных понятия – фрейм, документ и представление. Под документом понимаются те данные, с которыми работает приложение. Это может быть все, что угодно, – «простой текст», картинка и т.п. Отображение этих данных на экране осуществляется во фрейме документа. Фрейм содержит специальные окна – представления (Views), в которых отображают данные документа, и осуществляет координацию между документом и различными типами их представления, направляя им команды и получая от них извещения. Представлений документа в приложении может быть несколько. Пользователь взаимодействует с документом посредством его представления.

Фрейм и представление реализованы в виде двух различных оконных классов – класса фрейма и класса представления, при этом окно представления является дочерним по отношению к фрейму, т. е. размещается в его рабочей области. Сам фрейм может быть окном SDI-приложения или дочерним окном MDI-приложения. За взаимодействие пользователя непосредственно с фреймом (изменение размеров, перемещение и т.п.) полностью отвечает сама система Windows. Управление же представлениями ложится на программиста.

Документ представлен в виде объекта, который обеспечивает пространство для хранения данных в памяти и отвечает за такие операции, как запись и чтение документа с файла. Большая часть операций с данными выполняется самим приложением, а не классами библиотеки MFC.

Таким образом, архитектура «Документ/Представление» охватывает следующие основные классы:

– CWinApp – класс для создания единственного объекта – приложения;

– CFrameWnd – класс для создания главного окна однодокументного приложения и базовый – для классов *CMDIFrameWnd* и *CMDIChildWnd*, которые отвечают за работу многодокументного приложения;

– CDocTemplate – базовый абстрактный класс для создания шаблонов документов;

– CDocument – класс для создания собственно документа;

– CView – базовый класс, который совместно со своими производными классами – CCtrlView, CEditView, CListView, CTreeView, CScrollView отвечает за отображение данных документа и за взаимодействие с пользователем.

При создании MFC *AppWizard* каркаса приложения архитектуры «Документ/Представление» создастся типовой набор классов, производных от вышеназванных классов (если на шаге 1 установить флажок использования архитектуры «Document/View») (рисунок 14.1).

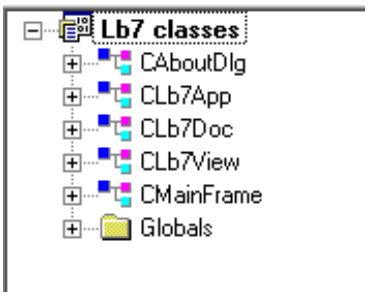

Рисунок 14.1 – Классы каркаса приложения архитектуры «Документ/Представление»

Созданный каркас приложения достаточно полно реализует функции фрейма, однако функции по описанию данных и их представлению реализовывает программист.

Рассмотрим принципы реализации данных в классе документа. Для реализации документа в типичном приложении необходимо проделать следующие действия:

– для каждого документа образовать класс на базе CDocument;
– добавить в него переменные для хранения всех данных документа;
– реализовать функции для чтения и модификации этих данных;
– переопределить функцию Cobject::Serialize() в новом классе документа для организации чтения/записи данных документа с диска.

Если используется один класс документа, то он уже создан MFC *AppWizard* (рисунок 14.1). Добавление переменных для хранения данных документа рассмотрим на следующем примере. Данными, с которыми работает приложение, будут фамилия, имя, отчество абитуриента и оценки по 3-м дисциплинам – математике, физике и русскому языку. Описать эти данные можно следующей структурой:

```
typedef struct
{       CString     name1;
        CString     name2;
        CString     name3;
        int     mark1;
        int     mark2;
        int     mark3;
}
SRTUCTRECORD;
```

В классе документа эти данные опишем в виде массива, используя шаблон для класса *CArray*:

```
class CLb7Doc : public CDocument
{
protected: // create from serialization only
        CLb7Doc();
        DECLARE_DYNCREATE(CLb7Doc)
…// Attributes
public:
  CArray <SRTUCTRECORD, SRTUCTRECORD> m_data;
```

Для занесения и извлечения данных из массива используются методы класса *CArray* , поэтому здесь не будем использовать собственные методы, тем более что член класса m_data является открытым.

Для сохранения и чтения из файла данных в документ, а также для создания документа MFC *AppWizard* реализованы команды ID_FILE_OPEN и ID_FILE_SAVE, ID_FILE_SAVE_AS и ID_FILE_NEW (рисунок 14.2).

Рисунок 14.2 – Команды для работы с документом, созданные MFC *AppWizard* для каркаса приложения архитектуры «Документ/Представление»

Для выполнения этих команд библиотекой MFC вызываются соответствующие методы, реализация которых скрыта от программиста. Только два метода могут быть изменены. В методе OnNewDocument() можно произвести инициализацию данных документа для MDI- приложения или очистку данных для SDI-приложения, т. е. уничтожение старых данных. Например, для нашего случая это может быть следующий код:

```
BOOL CLab7Doc::OnNewDocument()
{
// Всю стандартную обработку по созданию документа возлагаем
// на библиотеку MFC
if (!CDocument::OnNewDocument())
                return FALSE;
```

```
//После успешного создания проводим очистку массива  с данными
        m_data.RemoveAll();
        return TRUE;
}
```

Метод Serialize() – здесь выполняются команды сохранения данных в файл или чтения из файла данных в структуры документа. Этот метод вызывается при выполнении команд Open, Save или Save As после выбора блоком диалога имени файла.

Метод Serialize называется сериализацией – преобразование в последовательную форму и наоборот, он позволяет сохранять и восстанавливать объекты классов, созданных на базе класса CObject. В нашем случае нужно дополнить этот метод кодом по сохранению и восстановлению данных из файлового бинарного потока (данные записываются в файл без преобразования), который называют архивом. В качестве параметра в этот метод передается указатель на созданный экземпляр класса CArchiv , с которым ассоциирован объект класса CFile. Имя файла подставляется после выполнения кода диалогового окна по выбору имени файла. В классе CArchiv реализованы несколько вариантов перегружаемых операторов >> и << , которые загружают объекты (или простые типы) из архива или сохраняют их в архиве. Они имеют следующие однотипные формы:

```
friend CArchive& operator << (CArchive &ar, CObject *&pOb)
friend CArchive& operator >> (CArchive &ar, CObject *&pOb)
или для простых типов :
friend CArchive& operator << (Data_Type& data)
friend CArchive& operator >> (Data_Type& data),
```

где через Data_Type обозначены примитивные типы данных BYTE, WORD, int, DWORD, float double.
Метод CArchive::IsStoring возвращает направление потока:

```
void CLb7Doc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {               // TODO: add storing code here
                // Работа с потоком выполняется с простыми типами
                ar << m_data.GetSize();
                for (int i=0; i < m_data.GetSize(); i++)
                {
                        ar << m_data.GetAt(i).name1;
                        ar << m_data.GetAt(i).name2;
                        ar << m_data.GetAt(i).name3;
                        ar << m_data.GetAt(i).mark1;
```

```
            ar << m_data.GetAt(i).mark2;
            ar << m_data.GetAt(i).mark3;
        }
    }
    else
    {
        // TODO: add loading code here
        int size;
        SRTUCTRECORD rec;
        ar >> size;
        m_data.SetSize(size, 1);
        for (int i=0; i < size; i++)
        {
            ar >> rec.name1;
            ar >> rec.name2;
            ar >> rec.name3;
            ar >> rec.mark1;
            ar >> rec.mark2;
            ar >> rec.mark3;
            m_data.SetAt(i,rec);
        }
    }}
```

Задание к лабораторной работе

Создать MDI-приложение для формирования и просмотра данных из файла в соответствии с индивидуальным заданием.

Требования для создаваемого приложения:

1 Управление заданием имени файла, его открытием или созданием и т.п. выполняется через команды меню или кнопки панели инструментов.

2 Просмотр загруженных данных из файла, а также формируемых в приложении данных выполняется в диалоговом окне с использованием элемента управления *Listbox*. В этом списке отображаются либо все данные, либо данные, формируемые в соответствии с функцией, определенной в индивидуальном задании. Порядок вывода управляется с помощью кнопок. Исходно выводится полный список.

3 Формирование данных в приложении осуществляется в другом диалоговом окне с использованием элементов управления *Combobox* и *Edit*.

4 До задания имени файла элементы меню и соответствующие кнопки управления открытием и сохранением данных в файл должны быть не доступны.

5 После задания имени файла все команды работы с файлом должны иметь место только для этого файла.

6 Имя заданного файла должно отображаться в строке статуса.

Варианты заданий

1 Создайте БД (базу данных) фирмы по продаже стройматериалов. Ориентировочные данные: «Продукция», «Заказчики», «Контракты».

2 Создайте БД ветеринарной станции. Ориентировочные данные: «План ветеринарных работ», «Обслуживаемые хозяйства», «Проведенные работы».

3 Создайте БД фирмы по проведению буровых работ. Ориентировочные данные: «Услуги-расценки», «Клиенты», «Контракты».

4 Создайте БД загрузки аудиторий. Ориентировочные данные: «Аудитории», «Учебные дисциплины», «Группы».

5 Создайте БД трикотажной фабрики. Ориентировочные данные: «Сырье», «Изделия», «Сбыт изделий».

6 Создайте БД ателье головных уборов. Ориентировочные данные: «Изделия», «Клиенты», «Квитанции».

7 Создайте БД гостиницы. Ориентировочные данные: «Номера», «Клиенты», «Счета».

8 Создайте БД жилищного коммунального хозяйства. Ориентировочные данные: «Специалисты», «Жилищный фонд», «Мероприятия».

9 Создайте БД стоматологической поликлиники. Ориентировочные данные: «Врачи», «Пациенты», «Обслуживание».

10 Создайте БД сборочного процесса. Ориентировочные данные: «Комплектующие», «Изделия», «Технологические процессы».

11 Создайте БД мебельной фабрики. Ориентировочные данные: «Изделия», «Заказчики», «Контракты».

12 Создайте БД кабельного завода. Ориентировочные данные: «Сырье», «Продукция», «Технологические процессы».

13 Создайте БД механизированной колонны. Ориентировочные данные: «Техника», «Обслуживающий персонал», «Путевки/наряды».

14 Создайте БД санатория. Ориентировочные данные: «Оздоровительные программы», «Отдыхающие», «Диагностируемые заболевания».

15 Создайте БД геологоразведочной экспедиции. Ориентировочные данные: «Регионы», «Карты», «Экспедиции».

16 Создайте БД фирмы по автоматизации производства. Ориентировочные данные: «Каталог устройств и программного обеспечения», «Клиенты», «Договоры».

17 Создайте БД банка для кредитования физических лиц. Ориентировочные данные: «Физическое лицо», «Данные по кредиту», «Выплаты».

18 Создайте БД для торгового зала. Ориентировочные данные: «Товар», «Склад».

19 Создайте БД поликлиники. Ориентировочные данные: «Врачи», «Пациенты», «Прием».

20 Создайте БД интернет-магазина. Ориентировочные данные: «Продукция», «Продажи».

21 Создайте БД автовокзала. Ориентировочные данные: «Рейсы», «Транспорт».

22 Создайте БД видеопроката. Ориентировочные данные: «Диски», «Прокат».

23 Создайте БД склада. Ориентировочные данные: «Ячейки», «Товар».

24 Создайте БД риэлтерской компании. Ориентировочные данные: «Объекты недвижимости», «Договоры».

25 Создайте БД домашней библиотеки. Ориентировочные данные: «Книги», «Журналы».

26 Создайте БД отдела кадров. Ориентировочные данные: «Сотрудники», «Отделы».

27 Создайте БД АТС. Ориентировочные данные: «Абоненты», «Вызовы».

28 Создайте БД авиакомпании. Ориентировочные данные: «Рейсы», «Самолеты».

29 Создайте БД для централизованного тестирования. Ориентировочные данные: «Абитуриент», «Тест».

30 Создайте БД аптеки. Ориентировочные данные: «Лекарство», «Продажи».

Контрольные вопросы

1 Что составляет основу архитектуры «Документ/Представление»? Дайте характеристику каждому элементу.

2 Назовите основные классы архитектуры «Документ/Представление» и их назначение.

3 Для чего служит метод Serialize?

Литература

1 Павловская, Т. А. С++. Объектно-ориентированное программирование/ Т. А. Павловская, Ю. А. Щупак. – СПб. : Питер, 2004.

2 Карпов, Б. В. С++ – специальный справочник / Б. В. Карпов, Т. С. Баранова – СПб. : Питер, 2001.

3 Эккель, Б. Философия С++. Практическое программирование/ Б. Эккель, Ч. Эллисон. – СПб. : Питер, 2004.

4 Страуструп, Б. Язык программирования С++/ Б. Страуструп. – СПб. : Питер, 2005.

5 Бусько, В. Л. Основы ООП. С++. Лабораторный практикум для студентов всех специальностей и форм обучения БГУИР/ В. Л. Бусько, А. Г. Корбит, Т. М. Кривоносова. – Минск : БГУИР, 2005.

6 Константайн, Л. Разработка программного обеспечения / Л. Константайн, Л. Локвуд. - СПб. : – Питер, 2004.

7 Савич, В. С++ во всей полноте / В. Савич. – СПб.: – Питер, 2004.

8 Аллен, Э. Типичные ошибки проектирования / Э. Аллен. – СПб. : – Питер, 2003.

9 Пол, И. Объектно-ориентированное программирование с использованием С++/ И. Пол. – Киев : ДиаСофт, 2005.

10 Винниченко, И. В. Автоматизация процессов тестирования / И.В. Винниченко – СПб. : – Питер, 2005.

11 Богуславский, А. С. Си ++ и компьютерная графика. Лекции и практикум по программированию на Си ++ / А.С. Богуславский. – М. : Компьютер Пресс, 2003.

Св. план 2012, поз. 17

Учебное издание

Снисаренко Светлана Валерьевна
Стасевич Наталья Александровна
Капанов Николай Анатольевич

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ УПРАВЛЕНИЯ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

ПОСОБИЕ

Редактор *Т. П. Андрейченко*
Корректор *Е. Н. Батурчик*

Подписано в печать 31.05.2012. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 4,53. Уч.-изд. л. 4,5. Тираж 100 экз. Заказ 127.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6