ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
“МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ
РАДИОТЕХНИКИ, ЭЛЕКТРОНИКИ И АВТОМАТИКИ
(ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)”

ИНФОРМАТИКА
ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ

«Основы программирования на VISUAL C++ 6.0»

Методические указания
по выполнению курсовой работы
«Разработка программы исследования функций»

для студентов, обучающихся по специальностям
210104, 210105, 210108, 210601 и 190700

МОСКВА 2009

Составители: А.Ю. Остащенко
Е.Ф. Певцов
Редактор: И.В. Гладышев

В методическом указании рассматривается курсовое проектирование в интегрированной среде Visual C++. При работе над курсовым проектом программы исследования математических функций студенты обучаются на практике применять основные алгоритмические структуры и вычислительные методы определения корней уравнений, численного дифференцирования и интегрирования.

Материал предназначен для студентов дневного отделения.

Печатается по решению редакционно-издательского совета университета.

Рецензенты: Т.Г. Ситникова
В.И. Индришенок

ВВЕДЕНИЕ

Целью курсового проектирования является приобретение навыков программирования в интегрированной среде Visual C++ и изучение численных методов расчетов. При работе над проектом программы исследования функций студенты обучаются на практике применять основные алгоритмические структуры и вычислительные методы определения корней уравнений, численного дифференцирования и интегрирования.

1. ОБЩИЕ ЗАМЕЧАНИЯ ПО ОСОБЕННОСТЯМ ПРОГРАММИРОВАНИЯ НА C++

1.1. Указатели (pointer)

Указатель – это переменная, в которой записан адрес ячейки памяти компьютера. Адрес ячейки памяти можно получить с помощью оператора (&).

Например:

```
int i = 10;
cout << &i << endl;
```

в результате на экран будет выведен адрес ячейки памяти в которой хранится значение переменной **I,** равное 10.

Символ *, стоящий после наименования типа, указывает на то, что описанная переменная является указателем.

Пример:

```
int howOld = 50;
int* pAge = &howOld;
```

В указателе содержится *адрес* переменной, а для получения доступа к *значению* переменной используется оператор разыменования *.

Например:

```
int yourAge;
yourAge = *pAge;
```

в результате переменной **yourAge** присвоится значение, хранимое по адресу, содержащемуся в указателе **pAge**.

Обратите внимание: **тип * имя** – это объявление указателя. Просто ***** **имя** – значение, хранящееся по адресу, содержащемуся в памяти под данным именем имя. Т.е. символ * указывает, что

операция должна производится не над самим *адресом*, а над *значением*, сохранённым по адресу, который хранится в указателе. Компилятор по контексту программы определяет, какой именно оператор используется в данном случае.

Инициализируйте указатель нулевым значением при объявлении, если заранее не известно для указания на какую переменную он будет использоваться. Неинициализированные указатели могут приводить к проблемам в работе программы.

Указатели используются:

- для размещения данных в свободных областях памяти и доступа к ним;
- для доступа к переменным и функциям классов;
- для передачи параметров в функции по ссылке;

Локальные переменные и параметры функций размещаются в стековой памяти. По завершению работы функции стек очищается. В результате все локальные переменные оказываются вне области видимости и их значения уничтожаются. В отличии от стека, динамическая память не очищается до завершения работы программы, поэтому её очищением должен заниматься сам программист. Выделенная память в динамической области не может использоваться, пока явно не будет освобождена. Доступ к данным в динамической памяти можно получить только из функций, в которых есть доступ к указателю, хранящему нужный адрес. Это позволяет жестко контролировать доступ к данным в динамической памяти.

Для выделения памяти в динамической области используется ключевое слово **new**. Затем указывают тип объекта, который будет размещаться в памяти. В качестве результата оператор new возвращает адрес выделенного фрагмента памяти. Этот адрес должен присваиваться указателю.

Пример:

```
int * pPointer = new int;
```

Чтобы занести в данную область памяти значение, следует записать:

```
*pPointer = 72;
```

Удаление памяти осуществляется оператором **delete**, после которого следует имя указателя.

Например:

```
delete pPointer;
pPointer = 0;
```

Рекомендуется при освобождении памяти присваивать связанному с ней указателю нулевое значение. Сам указатель при освобождении памяти не уничтожается, поэтому освобождающемуся указателю необходимо присвоить нулевое значение, чтобы обезопасить его. Важно следить, какой указатель ссылается на выделенную область динамической памяти и вовремя освобождать её. Сама она не освободится.

Аналогично, в динамической памяти можно размещать любые объекты. Например, объект класса **Cat**, для управления которым создаётся указатель pCat, в котором хранится его адрес.

Пример:

```
Cat* pCat = new Cat;
```

Доступ к членам и методам класса осуществляется через оператор прямого доступа (.) как и раньше.

Например:

```
(*pCat).GetAge();
```

Скобки указывают, что оператор разыменования должен выполняться ещё до вызова функции **GetAge()**.

Есть второй способ доступа, а именно, через оператор косвенного обращения к члену класса **–>** .

Например:

```
pCat –> GetAge();
```

Каждый метод класса имеет скрытый параметр – указатель **this**. Этот указатель содержит адрес текущего объекта. Память для указателя **this** выделяется и освобождается компилятором самостоятельно.

Указатель можно объявлять константным. В этом случае не допускается присвоения данному указателю нового адреса.

Указатель, хранящий адрес константного объекта не может использоваться для изменения этого объекта.

Один указатель можно вычитать из другого. Это может понадобиться при вычислении количества элементов массива, если эти указатели указывают на разные элементы массива.

1.2. Ссылки

Ссылка – это альтернативное имя данного объекта (по сути псевдоним). Для объявления ссылки нужно указать тип объекта адресата, за которым следует оператор ссылки (&), а за ним – имя ссылки.

Например:

```
int &rSomeRef = someInt;
// rSomeRef - ссылка на целочисленное значение.
```

Обратите внимание на то, что оператор ссылки выглядит так же, как оператор взятия адреса, который используется для возвращения адреса при работе с указателями. Однако это *разные* операторы.

Если использовать ссылку для получения адреса, то она будет показывать тот же адрес, что и объект. Т.е. ссылка – полный синоним адресата (объекта), даже если применяется оператор адреса.

В качестве упражнения, иллюстрирующего сказанное, выполните такую программу:

```
#include <iostream.h>
int main()
    {
        int intOne;
        int &rRef=intOne; // создание ссылки
        intOne=5;
        cout<< “intOne: “ << intOne<< endl;
        cout<< “rRef: ”<< rRef<< endl;
        rRef=7;
        cout<< “intOne: “ << intOne<< endl;
        cout<< “rRef: ”<< rRef<< endl;
        cout<< “&intOne: “ << &intOne<< endl;
```

```
        cout<< “&rRef: ”<< &rRef<< endl;
        return 0;
    }
```

Ссылаться можно на любой объект, включая объекты классов. Но нельзя ссылаться только на тип.

Например, запись: ***int &rRef = int;*** вызовет ошибку.

Правильно будет:

```
int B=200;
int &rRef = B;
```

Ссылки на объекты используются точно также, как сами объекты. Доступ к данным-членам и методам осуществляется с помощью обычного оператора прямого доступа **(.)**. В отличие от указателей ссылке не может быть присвоено нулевое значение.

Чтобы лучше понять, для чего нужны ссылки и указатели, выполните такой пример:

```
#include <iostream.h>
void swap (int x, int y); // прототип функции

int main()
    {
    int x=5, y=10;
    cout << "Main.Before swap. x=" << x << " y="<< y
        <<"\n";
    swap(x, y); // вызов функции в основной программе
    cout << "Main.After swap. x=" << x << " y="<< y
    <<"\n";
    return 0;
    }
void swap(int x, int y)
    {
    int temp; // создание промежуточной переменной
    cout << "Swap.Before swap. x=" << x << " y="<< y
    <<endl;
    temp = x;
    x=y;
    y=temp;
```

```
        cout << "Swap.After swap. x=" << x << " y="<< y
                <<endl;
        }
```

Результаты работы программы будут такими:

Main. Before swap. x=5 y=10
Swap. Before swap. x=5 y=10
Swap. After swap. x=10 y=5
Main. After swap. x=5 y=10

Т.е. в функции произошла перемена местами значений **x** и **y**, а их исходные значения не изменились. В функцию отправляются копии значений аргументов, а сами аргументы защищены от изменений. Такая передача аргументов в функцию называется передачей *по значению*.

Иногда необходимо, чтобы функция меняла значения аргументов на самом деле. Для этого и служат ссылки и указатели. Если функции передаётся значение как ссылка, то в стек помещается не значение объекта, а его адрес и все изменения в функции отражаются на объекте.

Пример использования указателей:

```
#include <iostream.h>
void swap (int *, int *); // аргументы - это указатели
        int main()
        {
                int x=5, y=10;
                cout << "Main.Before swap. x=" << x << " y="<<
                        y <<endl;
                swap(&x,&y); //передаются адреса переменных
                cout << "Main.After swap. x=" << x << " y="<< y
                        <<endl;
                return 0;
        }
void swap(int*  px, int*  py)
        {
                int temp;
                cout << "Swap.Before swap. *px=" << *px << "
                        *py="<<*py<<endl;
```

```
        temp = *px;             /* значение по адресу px (а в
данном случае px = &x, т.е. адрес переменной x ) присваива-
ется temp */
        *px=*py;
        *py=temp;
        cout << "Swap.After swap. *px=" << *px << "
            *py="<<*py<<endl;
    }
```

Пример использования ссылок в функциях:

```
#include <iostream.h>
void swap (int &, int &); // аргументы - это ссылки
    int main()
    {
        int x=5, y=10;
        cout << "Main.Before swap. x=" << x << " y="<<
            y <<endl;
        swap(x,y); // передаются переменные x и y
        cout << "Main.After swap. x=" << x << " y="<< y
            <<endl;
        return 0;
    }
void swap(int& rx, int& ry)
    {
        int temp;
        cout << "Swap.Before swap. rx=" << rx << "
        ry="<<ry<<endl;
        temp = rx; //  ссылка rx присваивается temp
        rx=ry;
        ry=temp;
        cout << "Swap.After swap. rx=" << rx << "
            ry="<<ry<<endl;
    }
```

Обратите внимание, что для указателей передаются в функцию адреса переменных (**&x**, **&y**), а для ссылок сами переменные **x** и **y**. Таким образом, благодаря использованию ссылок функция

может изменять исходные данные. При этом сам вызов функции ничем не отличается от обычного.

Ссылки проще использовать, но ссылки нельзя переназначать. Если вам нужно сначала указывать на один объект, а потом на другой, придётся использовать указатели. Ссылки не могут быть нулевыми. Поэтому если объект может быть нулевым, то вам нельзя использовать ссылку. Только указатель.

В качестве примера рассмотрим оператор new. Если оператор new не сможет выделить память для нового объекта, он возвратит нулевой указатель. А поскольку ссылка не может быть нулевой, вы не должны инициализировать ссылку на эту память до тех пор, пока не проверите, что она не нулевая.

Пример:

```
int * pInt = new int;
if( pInt != NULL )
int & rInt = *pInt;
```

1.3. Наследование

Наследование – это механизм, позволяющий строить иерархии, в которых производные (последующие) классы получают элементы базовых (родительских) классов и могут их дополнять или изменять их свойства.

Для создания производного класса используется ключевое слово **class**, после которого указывается имя нового класса, двоеточие тип объявления класса, а затем имя базового класса. Базовый класс должен быть объявлен ранее.

Пример:

```
class Grey : public Cat
{ тело класса }
```

Члены класса, объявленные как **private**, недоступны для наследования. Если же их объявить как **protected**, то данные-члены класса станут доступными для всех производных классов, но недоступными для всех внешних классов.

Если создаётся объект производного класса, то сначала вызывается конструктор базового класса, а затем конструктор производного класса, который завершает создание объекта. При уда-

лении объекта из памяти сначала вызывается деструктор производного класса, например: **~Grey( )**, а затем деструктор базового класса, например: **~Cat( )**.

Объект производного класса имеет доступ ко всем методам базового класса, а также ко всем своим методам. Кроме того, методы базового класса могут быть замещены. Под замещением понимается изменение действий функции, обладающей тем же именем и тем же типом и набором аргументов. В случае замещения одного из методов остальные варианты этого метода (т.е. имеющие то же имя, но другие типы и количество аргументов) оказываются скрытыми. Если вы хотите их использовать в производном классе, то их также нужно будет заместить в этом классе.

Обратиться напрямую к методу базового класса можно так:

Murzik.Cat :: Meow( );

т.е.: имя объекта производного класса, *точка*, имя базового класса, *два двоеточия и* имя вызываемого метода.

1.4. Массивы

Массив – это последовательность элементов одного типа, имеющая общее имя. Доступ к каждому элементу осуществляется указанием индекса (номера) данного элемента.

Объявление массива:

тип имя [количество элементов];

Например:

```
int Mas [30];
string Names [15];
```

Размерность массива, указанная в квадратных скобках – целое число, причем если объявлено [n] элементов, то они будут иметь номера начиная с нулевого, т.е. 0, 1, 2 … n-1. Если при обращении к элементу массива указать номер вне промежутка 0, 1, … n-1, то произойдёт выход за пределы массива. Компилятор этого не отслеживает, и это может приводить к ошибками в работе программы.

Инициализация массива осуществляется списком значений:

```
int mas [3] = {82, 3, 18};
```

Можно не указывать размер массива при инициализации:

int mas [ ] = {100, 8, 36,1,0};

в этом случае компилятор сам присвоит размерность массиву исходя из списка инициализации.

Пример: программа, которая заменяет в массиве из 10 элементов все элементы, большие 7, - на нули.

```
#include <iostream.h>
 int main()
      {
      int mas [10]; // объявляем массив из 10 элементов
            // Вводим данные в массив (целые величины)
            for (int i=0; i<10; i++)
                  {
                  cout << "Enter a number: ";
                   cin >> mas[i];
                  }
            for (int j=0; j<10; j++)
                  {
                  if(mas[j]>7)
                  mas [j] =0;
                  }
            cout<< "Result is: " << endl;
            for (int k=0; k<10; k++)
                  cout << "Element number "<<k<<" has a
                        value "<<mas[k]<<endl;
            return 0;
      }
```

Необходимо также отметить, что имя массива является указателем на нулевой элемент данного массива. Т.е. записи:

int * p = &mas[0]; и **int* p=mas;**

эквиваленты.

Пример: вывести все элементы массива до первого отрицательного.

1) Решение без использования указателей:

```
int i = 0;
      while (mas[i]>=0)
```

```
{
        cout<< mas[i];
        i++;
}
```

2) Решение с указателями:

```
int* p = mas; /*указатель p равен адресу начала
                массива */
 while (*p>=0)  /* пока значение по адресу p не
                меньше нуля */
{
        cout<<*p; // выводим значение p
        p++;  // адрес p увеличиваем на 1
}
```

Упражнение 1. Выполните самостоятельно по вариантам в табл.1.

Замечание: для вычисления модуля используйте функцию **fabs(double)**; а для округления до целого числа функцию **ceil(double).** Обе эти функции возвращают значения типа **double** и для их использования необходимо подключить библиотеку **<math.h>**.

Таблица 1

Варианты для выполнения упражнения 1

1	Дана целочисленная прямоугольная матрица. Определить: 1) количество строк, не содержащих ни одного нулевого элемента; 2) максимальное из чисел, встречающихся в заданной матрице более одного раза.
2	Дана целочисленная прямоугольная матрица. Определить: 1) количество столбцов, не содержащих ни одного нулевого элемента; 2) Характеристикой строки называется сумма её положительных чётных элементов. Переставляя строки матрицы, расположить их в соответствии с ростом характеристик.
3	Дана целочисленная прямоугольная матрица. Определить: 1) количество столбцов, содержащих хотя бы один нулевой элемент; 2) номер строки, в которой находится самая длинная серия одинаковых элементов.

4	Дана целочисленная квадратная матрица. Определить: 1) произведение элементов в тех строках, которые не содержат отрицательных элементов; 2) максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.
5	Дана целочисленная квадратная матрица. Определить: 1) сумму элементов в тех столбцах, которые не содержат отрицательных элементов; 2) минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы
6	Дана целочисленная прямоугольная матрица. Определить: 1) сумму элементов в тех строках, которые не содержат хотя бы один отрицательный элемент; 2) номера строк и столбцов всех Седловых точек матрицы (точка A_{ij} является Седловой, если она является минимальным элементом в i-ой строке и максимальным элементом в j-ом столбце).
7	Дана целочисленная прямоугольная матрица. 1) Привести систему к треугольному виду; 2) Найти количество строк, среднее арифметическое элементов которых меньше заданной величины.
8	Дана целочисленная прямоугольная матрица. 1)Уплотнить заданную матрицу, удаляя из неё строки и столбцы, заполненные нулями; 2) Найти номер первой из строк, содержащих хотя бы один положительный элемент.
9	Дана целочисленная прямоугольная матрица. Определить: 1) номер первого из столбцов, содержащих хотя бы один нулевой элемент; 2) Характеристикой строки называется сумма её отрицательных чётных элементов. Переставляя строки матрицы, расположить их в соответствии с убыванием характеристик.
10	1) Упорядочить строки целочисленной прямоугольной матрицы по возрастанию количества одинаковых элементов в каждой строке. 2) Найти номер первого из столбцов, не содержащих ни одного отрицательного элемента.
11	Дана целочисленная прямоугольная матрица. Определить: 1) количество строк, содержащих хотя бы один нулевой элемент; 2) номер столбца, в котором находится самая длинная серия одинаковых элементов.

12	Дана целочисленная квадратная матрица. Определить: 1) сумму элементов в тех строках, которые не содержат отрицательных элементов; 2) минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.
13	Дана целочисленная прямоугольная матрица. Определить: 1) количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент; 2) номера строк и столбцов всех Седловых точек матрицы (точка A_{ij} является Седловой, если она является минимальным элементом в i-ой строке и максимальным элементом в j-ом столбце).
14	Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. 1) Подсчитать количество локальных минимумов заданной матрицы размером 10 на 10. 2) Найти сумму модулей элементов, расположенных выше главной диагонали.
15	Осуществить циклический сдвиг элементов прямоугольной матрицы на *n* элементов вправо или вниз (в зависимости от введённого режима). *n* может быть больше количества элементов в строке или столбце.
16	Соседями элемента A_{ij} в матрице назовём элементы A_{kl}, где $i-1 \le k \le i+1$, $j-1 \le l \le j+1$, $(k,l) \ne (i,j)$. Операция сглаживания матрицы даёт новую матрицу того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы. 1) Построить результат сглаживания заданной вещественной матрицы размером 10 на 10; 2) В сглаженной матрице найти сумму модулей элементов, расположенных ниже главной диагонали.
17	В одномерном массиве, состоящем из *n* вещественных чисел, вычислить: 1) сумму отрицательных элементов массива; 2) произведение элементов массива, расположенных между максимальным и минимальным элементами. Упорядочить элементы массива по возрастанию.
18	В одномерном массиве, состоящем из *n* вещественных чисел, вычислить: 1) сумму положительных элементов массива; 2) произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами. Упорядочить элементы массива по убыванию.

19	В одномерном массиве, состоящем из *n* вещественных чисел, вычислить: 1) сумму элементов массива с нечётными номерами; 2) сумму элементов массива, расположенных между первым и последним отрицательными элементами. Сжать массив, удалив из него все элементы, модуль которых не превышает 1. Освободившиеся в конце массива элементы заполнить нулями.
20	В одномерном массиве, состоящем из *n* вещественных чисел, вычислить: 1) максимальный элемент массива; 2) сумму элементов массива, расположенных до последнего положительного элемента. Сжать массив, удалив из него элементы, модуль которых находится в интервале [a,b]. Освободившиеся в конце массива элементы заполнить нулями.

2. ЗАДАНИЕ ПО КУРСОВОЙ РАБОТЕ «ПРОГРАММА ИССЛЕДОВАНИЯ ФУНКЦИЙ»

Разработайте и протестируйте программу, выводящую в консоли меню, при выборе одного из пунктов которого над функцией в заданном интервале значений аргумента выполняются следующие операции:

1. Вывод значений аргумента и заданной функции F(x) с шагом h, начиная от начального значения a до конечного значения аргумента b.
2. Вычисление корней уравнения F(x)=0 методом дихотомии с точностью 0,0001.
3. Вывод на экран количества экстремумов функции F(x).
4. Вычисление интеграла функции методом прямоугольников с точностью 0,001 на отрезке между вторым и третьим корнем функции.
5. Вычисление интеграла функции методом трапеций с точностью 0,001 на отрезке между вторым и третьим корнем функции.
6. Вычисление методом Монте-Карло интеграла функции на отрезке между вторым и третьим корнем функции, задав не менее 100 точек для усреднения значения интеграла.

Варианты заданий по курсовой работе приведены в табл. 2.

2.1. Указания по выполнению курсовой работы

2.1.1. Определение корней уравнения методом дихотомии (деления отрезка пополам)

Пусть задано уравнение

$F(x) = 0$,

причём функция $F(x)$ непрерывна на отрезке $[a,b]$ и $F(a)* F(b)<0$. Из этого следует, что на этом отрезке она пересекает ось абсцисс нечётное число раз. Пусть корень только один. Разделим отрезок $[a,b]$ пополам: $x=(a+b)/2$. Далее возможны три случая:

- если $F(x) = 0$, тогда x есть искомый корень
- если $F(a)* F(x)>0$, то перемена знака имеет место в правой половине отрезка и следует изменить левую границу отрезка $[a,b]$: $a = x$ и продолжить процесс деления пополам;
- если $F(a)* F(x)<0$, то перемена знака функции имеет место в левой половине отрезка и следует изменить правую границу отрезка $[a,b]$, задав $b = x$, опять продолжить процесс деления пополам.

Процесс деления отрезка пополам следует завершаеть, когда выполнится одно из условий:

$F(x) < eps$, либо $b-a < eps$,

где eps – заданная погрешность вычислений.

Отыскание корней уравнения следует оформить отдельной функцией.

Для определения интервалов изоляции корней необходимо просмотреть последовательно пары соседних значений функции при заданном шаге разбиения области задания аргумента функции. При выполнении условия отрицательности произведения соседних значений функции, следует обратиться к функции отыскания корня методом дихотомии.

2.1.2. Определение количества экстремумов

Экстремумом функции называют точку локального минимума или максимума функции. В этих точка производная функции обращается в ноль. Таким образом задача сводится к предыдущей: следует найти количество нулей производной функции. Величину самой производной следует в данном случае вычислять

приближенно как отношение приращения значения функции к приращению аргумента в данной точке при достаточно малом шаге приращения аргумента (не более 1% от интервала задания функции).

2.1.3. Вычисление определенных интегралов

Метод прямоугольников. Отрезок интегрирования делится на *n* равных частей длины $h= (b – a)/n$. Обозначим:

$x_0=a,$

$x_1=a+h,$

…

$x_i=a+i*h,$

$x_n=b,$

$F(x_i) = F_i.$

В каждом интервале разбиения площадь под функцией заменяется на прямоугольный отрезок $F(x_i)*h$. Чем меньше шаг *h*, тем точнее выражение для интеграла. Затем все площади суммируются, и получается значение интеграла функции:

$$\int_a^b F(x)dx = h(\sum_{i=1}^{n} F_i)$$

Метод трапеций. Отрезок интегрирования делится на n равных частей длины $h= (b – a)/n$. Обозначим аналогично предыдущему: $x_0=a$, $x_1=a+h$, … $x_i=a+i*h$, $x_n=b$, $F(x_i) = F_i$. Заменяя на каждом отрезке разбиения длиной *h*, подынтегральную функцию F(x) отрезком прямой, проходящей через крайние точки отрезка разбиения x_i и x_{i+1} получаем формулу трапеций:

$$\int_a^b F(x)dx = \frac{h}{2}(F_0 + 2\sum_{i=1}^{n-1} F_i + F_n)$$

<u>Замечание.</u> О точности вычислений определенных интегралов. Оценка погрешности вычисления определенного интеграла делается по формулам Рунге. На практике требуется выбрать зна-

чение шага разбиения h. Алгоритм его вычисления сводится к следующему. Выбираются шаги h_1 и h_2 , причем $h_2<h_1$. Используя эти шаги, вычисляются каким-либо вышеописанным методом значения интегралов $A_1=A(h_1)$ и $A_2=A(h_2)$, которые являются приближёнными значениями истинного значения величины интеграла A. Если A_1 и A_2 оказываются близкими по некоторому критерию точности, то за искомое значение принимается A_2. В противном случае выбирается новый шаг $h_3<h_2$ и снова вычисляется значение $A_3=A(h_3)$, которое сравнивается с A_2. Удобно выбирать $h_{i+1}=h_i/2$ (метод двойного пересчета). Значение A считается найденным, если выполняется одно из условий:

$$|A_{i+1} - A_i| \le \varepsilon \text{ при } |A_{i+1}| \le 1, \text{ или}$$

$$\left| \frac{A_{i+1} - A_i}{A_{i+1}} \right| \le \varepsilon \quad \text{при } |A_{i+1}| > 1.$$

Если эти условия не выполняются, то процесс уменьшения шага продолжается. Таким образом, реализация указанного метода сводится к вычислению определённых интегралов при заданных шагах разбиения h_1 и $h_2=h_1/2$, что удобно оформить в виде отдельной функции. Полученные значения сравниваются и, если условия достижения заданной точности не выполняются, то цикл **do { } while (условие)** продолжается.

Метод Монте-Карло. Данный метод относится к статистическим методам. В качестве узла разбиения выбирается случайное число, выбранное из участка интегрирования *[a,b]*. Проводится N вычислений со случайными узлами x_i, результат усредняется и принимается за приближённое значение интеграла:

$$\int_a^b F(x)dx = \frac{b-a}{N}\left(\sum_{i=1}^{N} F(x_i)\right)$$

Погрешность вычисления интеграла зависит от числа испытаний N и пропорциональна $N^{1/2}$.

Генерирование псевдослучайных чисел осуществляется командой **rand().** Для этого необходимо подключить библиотеки **stdlib и time:**

```
#include <stdlib.h>
#include <time.h>

srand  ( time ( NULL ) );
double j = (  (double)  rand() / RAND_MAX);
```

Данные строки позволяют сгенерировать псевдослучайное вещественное число из диапазона от 0 до 1. Его затем следует перевести в случайное число заданного диапазона, умножив на соответствующий нормирующий множитель.

3. ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ: ОСНОВЫ РАБОТЫ С ГРАФИЧЕСКИМИ СРЕДСТВАМИ VISUAL C++ НА ПРИМЕРЕ ПОСТРОЕНИЯ ГРАФИКА ФУНКЦИИ

Построение графика функции выполните, используя встроенные шаблоны и библиотечные функции Visual C++. Для этого выполните следующие действия.

Создайте новый проект, выбрав меню: ***File->New…*** и выбрав затем во вкладке ***Project*** обычное приложение ***Win32 Application.*** Задайте имя проекта, в данном случае это имя ***GraficFunction*** и далее нажмите *OK* и выберите ***Typical "Hello World" application.*** Затем нажмите *Finish* и *OK*. Далее щёлкните по любому из классов папки ***Globals*** во вкладке ***ClassView*** и увидите содержание файла ***GraficFunction.cpp***, текст которого приводится ниже. Жирным шрифтом в программном коде выделены те изменения и дополнения, которые необходимо внести в шаблон (исходный код графика) для построения графика функции. Комментарии в тексте кода проясняют суть операций.

```
// GraficFunction.cpp : Defines the entry point for the application.
#include "stdafx.h"
#include "resource.h"
#include "Math.h"

#define MAX_LOADSTRING 100
```

```
// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];      // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];
                                    // The title bar text
// Foward declarations of functions included in this code module:
ATOM            MyRegisterClass(HINSTANCE hInstance);
BOOL                    InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT,
WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM,
LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
        {// TODO: Place code here.
              MSG msg;
              HACCEL hAccelTable;
                    // Initialize global strings
              LoadString(hInstance, IDS_APP_TITLE, szTitle,
                    MAX_LOADSTRING);
              LoadString(hInstance, IDC_GRAFICFUNCTION,
                    szWindowClass, MAX_LOADSTRING);
              MyRegisterClass(hInstance);
              // Perform application initialization:
              if (!InitInstance (hInstance, nCmdShow))
                    {
                    return FALSE;
                    }
              hAccelTable = LoadAccelerators(hInstance,
              (LPCTSTR)IDC_GRAFICFUNCTION);
  // Main message loop:
              while (GetMessage(&msg, NULL, 0, 0))
              {
```

```
                if (!TranslateAccelerator(msg.hwnd,
                        hAccelTable, &msg))
                {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
                }
            }
        return msg.wParam;
        }
//  FUNCTION: MyRegisterClass()
//  PURPOSE: Registers the window class.
//  COMMENTS:
//    This function and its usage is only necessary if you want this
code
//    to be compatible with Win32 systems prior to the 'Register-
ClassEx'
//    function that was added to Windows 95. It is important to call
this function
//    so that the application will get 'well formed' small icons asso-
ciated
//    with it.
  ATOM MyRegisterClass(HINSTANCE hInstance)
        {
            WNDCLASSEX wcex;
            wcex.cbSize = sizeof(WNDCLASSEX);
            wcex.style = CS_HREDRAW | CS_VREDRAW;
            wcex.lpfnWndProc  = (WNDPROC)WndProc;
            wcex.cbClsExtra = 0;
            wcex.cbWndExtra = 0;
            wcex.hInstance = hInstance;
            wcex.hIcon = LoadIcon(hInstance,
                    (LPCTSTR)IDI_GRAFICFUNCTION);
            wcex.hCursor = LoadCursor(NULL,
                    IDC_ARROW);
            wcex.hbrBackground =
                    (HBRUSH)(COLOR_WINDOW+1);
```

```
            wcex.lpszMenuName =
                  (LPCSTR)IDC_GRAFICFUNCTION;
            wcex.lpszClassName = szWindowClass;
            wcex.hIconSm  = LoadIcon(wcex.hInstance,
                  (LPCTSTR)IDI_SMALL);
            return RegisterClassEx(&wcex);
      }
//   FUNCTION: InitInstance(HANDLE, int)
//   PURPOSE: Saves instance handle and creates main window
//   COMMENTS:
//       In this function, we save the instance handle in a global
variable and
//       create and display the main program window.
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
      {
            HWND hWnd;
            hInst = hInstance;
// Store instance handle in our global variable
            hWnd = CreateWindow(szWindowClass, szTitle,
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT, 0,
                  CW_USEDEFAULT, 0, NULL, NULL,
                  hInstance, NULL);
            if (!hWnd)
                  {
                  return FALSE;
                  }
            ShowWindow(hWnd, nCmdShow);
            UpdateWindow(hWnd);
            return TRUE;
      }
// Вычисляемая функция
      double y(double x)
            {
            return sin(x)*x;
            }
```

```
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//  PURPOSE:  Processes messages for the main window.
//  WM_COMMAND   - process the application menu
//  WM_PAINT    - Paint the main window
//  WM_DESTROY      - post a quit message and return
        LRESULT CALLBACK WndProc(HWND hWnd, UINT
              message, WPARAM wParam, LPARAM lParam)
        {
              int wmId, wmEvent;
              PAINTSTRUCT ps;
              HDC hdc;
              TCHAR szHello[MAX_LOADSTRING];
              LoadString(hInst, IDS_HELLO, szHello,
                    MAX_LOADSTRING);
#define xLow -5
// Нижняя граница аргумента функции
#define xHigh 5
// Верхняя граница аргумента функции
#define Step 0.1 // Шаг
#define Scale 100 // Приближение графика
        static int CenterX, CenterY; // Центр экрана
        double NumberOfPoint = (xHigh - xLow)/Step;
       // вычисляем количество точек
        double x;
        POINT * pPt;
        DWORD i;
        switch (message)
        {
              case WM_COMMAND:
                    wmId    = LOWORD(wParam);
                    wmEvent = HIWORD(wParam);
                    // Parse the menu selections:
                    switch (wmId)
                          {
                                case IDM_ABOUT:
```

```
                    DialogBox(hInst,
                    (LPCTSTR)IDD_ABOUTBOX,
                    hWnd, (DLGPROC)About);
                    break;
                    case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                    default:
                    return DefWindowProc(hWnd,
                    message, wParam, lParam);
                }
        break;
// необходимо добавить для задания центра
        case WM_SIZE:
            CenterX = LOWORD(lParam) / 2;
            CenterY = HIWORD(lParam) / 2;
            break;
        case WM_PAINT:
        // Выделяем память
            pPt = (POINT*) malloc(NumberOfPoint
                *sizeof(POINT));
        // просчитываем значение функции в точках
            x=xLow;
            for (i = 0; i< NumberOfPoint; i++)
                {
                pPt[i].x = (long) (CenterX + x *
                    Scale);
                pPt[i].y = (long) (CenterY + y(x) *
                    Scale);
                x+=Step;
                }
        hdc = BeginPaint(hWnd, &ps);
// Рисуем оси координат
        MoveToEx(hdc, CenterX, 0, NULL);
        LineTo(hdc, CenterX, CenterY*2);
        MoveToEx(hdc, 0, CenterY, NULL);
```

```
            LineTo(hdc, CenterX*2, CenterY);
        // Рисуем функцию
            Polyline(hdc, pPt, NumberOfPoint);
            EndPaint(hWnd, &ps);
        // Очищаем память
            free(pPt);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
        return DefWindowProc(hWnd, message, wParam,
            lParam);
            }
            return 0;
        }
// Mesage handler for about box.
        LRESULT CALLBACK About(HWND hDlg, UINT
            message, WPARAM wParam, LPARAM lParam)
        {
            switch (message)
                {
                    case WM_INITDIALOG:
                        return TRUE;
                    case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK ||
                        LOWORD(wParam) ==
                        IDCANCEL)
                {
                EndDialog(hDlg, LOWORD(wParam));
                return TRUE;
        }
            break;
  }
    return FALSE;
}
```

ЗАКЛЮЧЕНИЕ

По результатам выполнения курсовой работы необходимо оформить отчёт в виде текстового документа Word, который должен включать следующие пункты:

1. Формулировка задания с указанием номера варианта и исходных данных.
2. Копии экранов с графиками функции, производной и интеграла функции и вычисленными значениями корней и определенного интеграла, выполненным заданием в MathCad.
3. Полный программный код на Visual C++ с комментариями.
4. Копия экрана консоли с работающей программой.
5. Результаты тестирования и результаты работы программы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Д. Либерти «Освой самостоятельно C++ за 21 день». СПб.: Диалектика, 2006
2. Т.А. Павловская «C/C++. Программирование на языке высокого уровня». СПб.: Питер, 2007.
3. С.В. Глушаков, А.В. Коваль, С.В. Смирнов «Язык программирования C++» Изд-во Фолио, 2001.

Подписано в печать 00.00.2009. Формат 60х84 1/16.
Бумага офсетная. Печать офсетная.
Усл. печ. л.00,00 Усл. кр.-отт. 00,00. Уч.-изд. л. 00,00
Тираж 000 экз. С 00

Государственное образовательное учреждение
высшего профессионального образования
“Московский государственный институт радиотехники,
электроники и автоматики (технический университет)”
119454, Москва, пр. Вернадского, 78

Таблица 2

№	Аргумент	Функция (N=1,2…5)
1.	$x := -10, -9.9 .. 10$	$y(x) := N - \frac{((2 - 18 \cdot N \cdot x) + 16 \cdot ((N \cdot x)^2))}{1 + N \cdot (x)^4}$
2.	$x := -3, -2.99 .. 1$	$y(x) := \frac{\sqrt{N}}{2} - \left[\frac{\left[\left[1 + 2 \cdot (x)^4 - 8 \cdot (x)^2 \right]^2 - 4 \cdot N \cdot x^3 + N^2 \cdot x^4 \right]}{(1 - N \cdot x)^4} \right]$
3.	$x := -5, -4.99 .. 2$	$y(x) := 5 + \frac{\left[\left[\left(-5 + 2 \cdot x + 1 \cdot x^2 \right) + 4 \cdot N \cdot x^3 \right] + N \cdot x^4 \right]}{(1 - N \cdot x)^{\frac{4}{5}}}$
4.	$x := -1, -0.99 .. 4$	$y(x) := \left[(((2 \cdot N \cdot x - 1) \cdot (2 \cdot x - 2)) \cdot (x - 3))^2 \right]^{\frac{2}{3}} - N$
5.	$x := -4, -3.99 .. 2$	$y(x) := -2 \cdot \sqrt{N} + \frac{\left[\left(0.5 \cdot x^5 - 8 \cdot x^3 - 8 \cdot x^2 \right) - N \cdot x + 10 \right]^3}{100 \cdot (x - 1)^4}$

6.	$x := 0, 0.01 .. 2$	$y(x) := \sin\left(N \cdot x^{\frac{3}{4}} + 4 \cdot x^2\right) + 2 \cdot x^{\frac{3}{4}} - 2$
7.	$x := -1.9, -1.89 .. 3$	$y(x) := e^{-2x+1} \cdot \sqrt{N} \cdot (\sin(4 \cdot x - 1))^2 - 8 \cdot (x)^2$
8.	$x := 0.2, 0.21 .. 10$	$y(x) := -3 \cdot \sqrt{N} + \left[-\frac{\left[x^5 - (3x)^3 - N^2 \cdot x + 10 \right]}{(x+1)^4} \right]^2$
9.	$x := 0, 0.01 .. 10$	$y(x) := 0.2 \cdot (x+2) + \sqrt{N} \cdot \sin(x) + \frac{4 \cdot \sin(3 \cdot x)}{3}$
10.	$x := 0, 0.01 .. 2$	$y(x) := \frac{\ln\left(8 \cdot \cos\left(4x^2\right)^2 + 1\right) - x^3 + 0.4}{N \cdot (\lvert x \rvert)^{\frac{1}{5}} - 4}$
11.	$x := -4, -3.99 .. 4$	$y(x) := 0.25 \cdot x^2 - 0.8 \cdot \sqrt{N} + (\sin(2 \cdot x - 1))^4 + N \cdot (\cos(x))^4$
12.	$x := -3, -2.99 .. 3$	$y(x) := 0.25 \cdot x^2 + 0.1 \cdot x - 0.8 \cdot \sqrt{N} + N \cdot (\sin((2 \cdot x)))^4 + N \cdot (\cos(x))^4$

13.	$x := 2, 2.01 .. 18$	$y(x) := \left\lvert \left(x^3 - 2\,x^2 + 1\right)^{\frac{(2\cdot N\cdot \sin(x))}{6}} \right\rvert - \frac{x\cdot N}{3}$
14.	$x := -2, -1.99 .. 5$	$y(x) := \left[\left[\cos\left[\frac{\left[x^3 - N\cdot(x)^2 + x - 1\right]}{20}\right]\right]^4 + N\cdot\left(\frac{x-1}{10}\right)^2\right] - 1 + \frac{x}{20}$
15.	$x := 0, 0.01 .. 3$	$y(x) := -6 + \left[10\cdot(N\cdot(e))^{\left[\left(x^5 + 6\cdot x\right)^{\frac{1}{2}} - x^2 - x - 1\right]} - (x)^2\right]$
16.	$x := 0, 0.01 .. 6$	$y(x) := N\cdot(e)^{\frac{x-4}{8}} - N\cdot x + \left[4\cdot\sin\left[\left(\frac{x}{4}\right)^2 + x\right]\right]^2$
17.	$x := -4, -3.99 .. 5$	$y(x) := N\cdot(e)^{\frac{x}{8}} - x^2 + \left[4\cdot\sin\left[\left(\frac{x}{4}\right)^2 + \sqrt{N}\cdot x\right]\right]^2$

18.	$x := -2, -1.99 .. 16$	$y(x) := \left[\left[x^5 - N\cdot(x)^4 - 4\cdot x^3 + (N\cdot x)^2 - 6\cdot N\cdot x\right] - 10\right]\cdot e^{-(x)} - 1$
19.	$x := -10, -9.99 .. 1$	$y(x) := 0.2\cdot(x+1) + N\cdot\sin(x) + \frac{2\cdot\sin(3\cdot x - 1)}{3}$
20.	$x := -4, -3.99 .. 4$	$y(x) := (2\cdot x)^2 - N^2\cdot(\sin((2\cdot x)))^4 + 4\cdot x + \left(\cos\left(\frac{x^2}{4}\right)\right)^4$
21.	$x := -4, -3.99 .. 4$	$y(x) := -(2\cdot x)^2\cdot N\cdot\sin((2\cdot x))^4 + 4\cdot N\cdot x + \left(\cos\left(\frac{x^2}{4}\right)\right)^4$
22.	$x := 0, 0.01 .. 10$	$y(x) := 0.2\cdot(x+2) + \sin(x) + \frac{4\cdot\sqrt{N}\cdot\sin(3\cdot x)}{3}$
23.	$x := -4, -3.99 .. 4$	$y(x) := 0.25\cdot x^3 - 0.8\cdot\sqrt{N} + (\sin(2\cdot x - 1))^4 + N^2\cdot(\cos(x))^4$
24.	$x := 0, 0.01 .. 6$	$y(x) := e^{\frac{x^{x-4}}{8}} - N\cdot x + \left[4\cdot\sin\left[\sqrt{N}\cdot\left(\frac{x}{4}\right)^2 + x\right]\right]^2$