



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
"Московский государственный технический университет радиотехники,
электроники и автоматики"
МГТУ МИРЭА

Факультет электроники

Подлежит возврату
№ 0806

**ИНФОРМАТИКА
ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ**

«Основы программирования на VISUAL C++ 6.0»

Методические указания
по выполнению лабораторных работ
для студентов, обучающихся по специальностям

Для студентов специальностей
210100, 210100 (550700), 222900,
210600, 210104, 210601

МОСКВА 2013

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

В методическом указании рассмотрены запуск и отладка консольных приложений в среде программирования Visual C++ 6.0, основные понятия языка программирования C++: переменные, условия, функции, циклы, классы, а также приводится список рекомендованной литературы.

Материал предназначен для студентов дневного отделения.

Печатается по решению редакционно-издательского совета университета.

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

ВВЕДЕНИЕ

Целью работы является знакомство с интегрированной средой разработки Visual C++ для операционной системы Windows и приобретение навыков работы по созданию проектов приложений с использованием переменных, условий, функций, циклов, классов. В заданиях и примерах звучат основные синтаксические конструкции языка C++, понятия переменных, использование функций, условий, циклов и классов, как основы объектно-ориентированного программирования.

Лабораторная работа №1. Основы разработки проектов в Visual C++6.0. Типы переменных

Программой называется конечная последовательность предписаний, сформулированных с помощью определённых синтаксических конструкций и служащая для выполнения каких-либо действий ЭВМ. Для перевода программы с алгоритмического языка в машинный код используются специальные программы-трансляторы: интерпретаторы и компиляторы. Интерпретатор обрабатывает программу пользователя поэлементно: преобразует в машинный код и сразу же его исполняет, т.е. программа выполняется ЭВМ только в среде интерпретатора. С помощью компилятора создаются файлы-программы (exe-файлы), которые могут самостоятельно выполняться в среде операционной системы ЭВМ.

Нами будет использован компилятор *Visual C++ 6.0*, входящий с пакет программ *Visual Studio* от компании *Microsoft*. Все программы данной лабораторной работы запускаются в консольном окне. Для запуска *Visual C++6.0* щёлкните дважды по ярлычку *Microsoft Visual C++ 6.0*. Появится среда разработки. Если открылось окошко с подсказками «Did you know...», то закройте его. В меню **File** выберите пункт **New**. Проследите, что открылась именно вкладка *Projects*, и выберите опцию **Win32 Console Application**. В поле **Project name** наберите имя проекта и щёлкните **OK**. Далее выберите опцию **An empty project** и нажмите **Finish**. В окошке *New project Information* просто нажмите **OK**. Затем снова выберите меню **File** – **New** и появится окошко *New* с открытой вкладкой *Files*. В ней выберите опцию **C++ Source File** и введите имя файла в поле *file name*, например **ex1**. Далее введите про-

граммный код и сохраните его. Для компиляции программы выберите меню **Build** → пункт **Build название_программы.exe** или нажмите **F7**. Если имеются ошибки, то исправьте их. При отсутствии ошибок запустите программу в консольном приложении: меню **Build** → пункт **Execute название_программы.exe (CTRL+F5)**.

Для завершения программы в консольном окне нажмите пробел. Вот типичная последовательность написания, отладки и запуска программы в консоли в среде Visual C++ 6.0, которая будет использована в данной лабораторной работе. Примечания обозначаются как `//` (действует до конца строки), либо `/* текст */`. Кроме разрыва строки существуют следующие управляющие символы: `\t` – табуляция, `\"` – двойная кавычка, `'` – одинарная кавычка, `\?` – знак вопроса, `\\` – обратный слеш, `\n` – новая строка (разрыв строки).

Напишем первую программу на C++, выводящую в консоль сообщение «Hello World!». В приведенном ниже программном коде команда **#include <iostream>** обеспечивает подключение файла ввода/вывода сообщений в консольном окне. Управляющий символ `\n` обозначает разрыв строки. Команда **cout <<** выводит данные на экран. Функция **main()** вызывается автоматически при запуске программы и должна присутствовать в каждой программе, т.е. в ней содержится основной код программы (внутри фигурных скобок `{ }`).

```
#include <iostream.h> /* директива включения файла ввода/вывода сообщений */
int main() // функция вызова основного содержания программы
{
cout << "Hello World! \n";
return 0; /* значение, возвращаемое функцией. Для main() - это условность */
}
```

Теперь коротко объясним, что такое функция. Программа выполняется по строкам, пока не встретится вызов функции. Это приводит к передаче управления функции. После выполнения функции управление возвращается строке программы, следующей за строкой вызова функции. Функция возвращает значение

или ничего не возвращает. Для последнего случая есть специальный тип функции **void**, которая не возвращает значение из функции. Рассмотрим на практике пример работы программы:

```
#include <iostream.h>
void Demka() /* функция, не возвращающая никакого значения, и выводящая фразу на экран*/
{
    cout<< "Demo funktion \n"; // тело функции
}
int main()
{
    cout<< "In Main!"<< 1 << endl;
    Demka(); // вызов функции в теле основной программы
    cout << "Back in main \n";
    return 0;
}
```

Из новых команд здесь появилась **cout<< "In Main!"<< 1 << endl;** Это позволяет вывести вслед за фразой «In Main!» значение 1 на экран. Команда **endl** означает конец строки (end of line). Вся суть работы программы и функции в ней приведена в комментариях программы.

Как можно видеть на данном примере, функция состоит из заголовка и тела. Заголовок содержит тип возвращаемого значения, имя функции и параметры. В языке C++ можно присваивать значения только самим данным и нельзя - типам данных. Например, запись: **int = 5;** неверна, правильно будет записать: **int x = 5.** Параметры передают в функцию значения определённых типов. Рассмотрим на практике пример программы, приведённой ниже:

```
#include <iostream.h>
int Add(int x, int y) // int – целочисленный тип значения
{
    return (x+y); // return – команда возврата значения из функции
}
int main()
{
    int a,b,c; // создание целочисленный переменных a,b,c
    cout << "Enter value of x: \n";
```

```

cin >> a; // ввод с клавиатуры значения переменной a
cout << "Enter value of y: \n";
cin >> b;
c=Add(a,b); // вызов функции с аргументами a и b
cout << "Total sum of " <<a<<" and "<<b<< " is:" << c << endl;
cout << "Size of variable C is:"<< sizeof(c) << " bytes \n";
return 0;
}

```

В данном примере заголовок функции - это: **int Add(int x, int y)**. Функция с именем **Add** возвращает значение типа *int* и содержит два параметра *x* и *y*, также типа *int* (целочисленный тип). Тело функции содержится между фигурными скобками и содержит обязательную команду **return**, которая возвращает из функции значение указанного типа. В самой программе создаётся три целочисленных переменных. Двум из них значения присваиваются с клавиатуры (оператор **cin >>**), а третьей переменной присваивается возвращённое из функции значение **c=Add(a,b)**, где **a** и **b** уже являются аргументами, т.е. параметрами, имеющими конкретные значения.

Команда **sizeof()** позволяет узнать размер переменной в байтах. Кроме целочисленного типа **int** существуют и переменные других типов, представленные в таблице 1.

Таблица 1

Основные типы данных

Тип	Размер в байтах	Принимаемое значение
char	1	256 значений символов
float	4	1,2e-38 до 3,4e38
double	8	2,2e-308 до 1,8e308
bool	1	true или false
unsigned short int	2	0 до 65535
short int	2	-32768 до 32767
unsigned long int	4	0 до 4294967295
long int	4	-2147483648 до 2147483647

Переменной называется поименованная ячейка памяти для хранения данных, значения которых могут изменяться. Как уже было видно из предыдущего примера, для создания переменной записывается её тип и имя. Для инициализации можно сразу присвоить ей значение. Пример:

```
int MyAge=3, yourAge, hisAge=40;
```

Создано три переменные целочисленного типа, первая и третья из которых инициализированы значениями 3 и 40.

Чтобы не писать длинные названия типов можно создать псевдоним типа данных командой **typedef**. Пример:

```
typedef unsigned short int US;
```

и далее в программе создаём переменные с новым типом:

```
US Width=5;
```

(переменная с именем **Width** типа **US** инициализирована значением 5).

Константой называется поименованная ячейка памяти для хранения не изменяющихся данных. Константа вводится ключевым словом **const**. Пример:

```
const int students = 15;
```

(создаётся целочисленная константа **students** со значением 15).

Для создания новых типов данных используется перечислимый тип данных (константы перечисления). Они вводятся ключевым словом **enum**. Пример:

```
enum COLOR {Red, Blue, Green, White, Black};
```

создан перечислимый тип данных с именем **COLOR**, состоящий из указанных в фигурных скобках значений. При этом **Red** присваивается значение 0, **Blue** – 1, и т.д.

Рассмотрим пример программы:

```
#include <iostream.h>
int main()
{
    enum Days {Monday=1, Tuesday, Wednesday,
    Thursday, Friday, Saturday, Sunday};
    int a; // переменная проекта
    cout << "Enter a day (1-7): \n";
    cin >> a; // ввод с клавиатуры номера дня
    if (a == Sunday || a == Saturday)
```

```

        cout << "\n It's a WeekEnds!\n";
    else
        cout<< "\n OK. Job'll not wait \n";
    return 0;
}

```

В данной программе заданы данные для перечислимого типа **Days**. Далее с клавиатуры вводится значение переменной проекта. Исходя из значения переменной, происходит ветвление программы (подробнее об этом ниже в работе). В программе значение для **Monday** инициировано 1 для большего удобства, т.к. принято считать понедельник первым днём недели, а не нулевым. Соответствующие значения для других переменных перечислимого типа сдвинулись на единицу автоматически.

В C++ применяются следующие основные операторы и выражения. Следует отметить, что в C++ существует приоритет одних операторов по отношению к другим. Чтобы избежать ошибок рекомендуется применять блоки и скобки.

1. Оператор присваивания: **=**. Пример: **x = a + b;**
2. Структурная единица – блок **{ }**. Пример:

```

{ temp=a;
  a=b;
  b=temp;}

```

3. Математические операторы: сложение **+**, вычитание **-**, умножение *****, целочисленное деление **/** и деление по модулю **%**. Пример: **5/3** равно **1** (ответ тоже целого типа). Чтобы получить вещественное число необходимо записать: **5,0/3,0**, что даст **1,66667**. Также существуют операции присвоения с суммой: **+=**. Пример: **Age +=2;** что эквивалентно записи **Age=Age+2;**. Аналогично применяются операторы: **-=**, **/=**, ***=** и **%=**.
4. Операции инкремента и декремента. Операция увеличения (уменьшения) переменной на единицу называется инкрементом (декрементом) и обозначается **a++**, что равносильно записи **a = a + 1;** или **a += 1;**. Соответственно, декремент обозначается: **a- -**.

Инкремент (декремент) бывает префиксный и постфиксный. Пример префиксного инкремента:

int a = ++x; /*если x=5, то сначала увеличиваем x (т.е. x=6), а затем присваиваем её переменной a (т.е. a=6)*/

Пример постфиксного инкремента:

int b = x++; /*если x=5, то сначала присвоить b (т.е. b=5), а затем увеличить на единицу x (т.е. x=6) */

5. Операторы отношения: равенства **=**, больше **>**, меньше **<**, больше или равно **>=**, меньше или равно **<=**, не равно **!=**.

6. Логические операторы: И **&&**, ИЛИ **||**, НЕ **!**.

7. Оператор условия **?:** имеет следующий синтаксис:
(выражение1) ? (выражение2) : (выражение3).

Если **выражение1** возвращает значение **true** (т.е. верно), то выполняется **выражение2**. В противном случае выполняется **выражение3**.

8. Оператор ветвления **if**. В самом простом виде оператор ветвления **if** записывается так:

```
if (условие)
    выражение1;
else
    выражение2;
```

Т.е. если условие верно, то выполняется **выражение1**, в противном случае выполняется **выражение2**. Выполняемых выражений может быть несколько:

```
if (условие)
    { выражение1;
      выражение2;
      .....
      выражение N;}
else
    выражениеM;
```

Возможно создание вложенных конструкций типа ветвление с последующим ветвлением, по иному условию:

```
if (условие1)
    {if (условие2)
      выражение1;
    else
      {if ()
        выражение2;
```

```

else
    выражение3;
}
else
    выражение4;

```

Пример программы по заданию условий. Напишите и протестируйте программу, которая вычисляет значение функции

$Y = a + \frac{a}{a+1} + 2.5 * x$; где a – параметр, который зависит от аргумента x : $a = 3.5 * x$, если $x \geq -2$, и $a = x + 1.5$, если $x < -2$.

```

#include <iostream.h>
int main()
{
    double x,a, answer;
    cout << "Enter a value of x: ";
    cin >> x;
    if (x >= - 2)
    { a = 3.5*x;
      answer = a + (a/(a+1)) + 2.5*x;
    }
    else
    { a = x + 1.5;
      answer = a + (a/(a+1)) + 2.5*x;
    }
    cout << "Value of fuction Y is " << answer << endl;
    return 0;
}

```

9. Оператор ветвления **switch**

Позволяет осуществлять ветвление программы с количеством ветвей больше двух за один раз.

```

switch( выражение )
{
    case первое_значение: оператор;
                        break;
    case второе_значение: оператор;
                        break;
}

```

```

case значение_N : оператор;
                    break;
default : оператор;
}

```

Оператор **switch** сравнивает выражение, указанное в скобках, со значениями, приводимыми после ключевого слова **case**. В случае совпадения выполняются операторы, следующие после двоеточия до оператора **break** с последующим выходом из **switch**. В случае отсутствия оператора **break** после выражения, следующего за **case**, будет выполняться переход к выражению очередного блока **case**. Так бывает, когда оператор **break** пропущен по ошибке. Если же это делается умышленно, то вставляйте комментарии во избежание недоразумений.

Если выражение не совпадает ни с одним из случаев, указанных в **case**, то выполняется оператор по умолчанию, следующий после ключевого слова **default**.

Лабораторная работа №2. Программирование условий

Создайте самостоятельно программу вычисления значения функции в соответствии с вводимым значением переменной x в соответствии с вашим вариантом (см. Табл.2).

Примечания: 1) если в задании имеются тригонометрические функции и возведение в степень, то необходимо в начале программы подключить стандартную библиотеку математических операций, записав: **#include <math.h>** после **#include <iostream.h>**; 2) возведение числа a в степень b осуществляется командой **pow (a,b)**.

Таблица 2

№	Y=	a=
1	$Y = \frac{2a^2 + 7a - 2}{x - 2.5} + X^3$	$a = \begin{cases} x + 1.5x^3 + e^{x+1} & (-2 \leq x \leq 5) \\ 4.5x + 1.5 & (5 < x \leq 7.5) \\ x + 1 & (x > 7.5) \\ \text{не определено} & (x < -2) \end{cases}$

2	$Y = \frac{2}{x} + a^3$	$a = \begin{cases} 2 + 2x & (x \leq 4) \\ 3 & (4 < x \leq 5) \\ x + 1 & (x > 5) \end{cases}$
3	$Y = \frac{2.2a + 3.2a^2 - 2}{x - 1.5}$	$a = \begin{cases} 2 \sin x + 4 \cos^2 x^2 & (-1 \leq x \leq 1) \\ x + 3.5 & (1 < x \leq 5) \\ -4.6 & (x > 5) \\ \text{не определено} & (x < -1) \end{cases}$
4	$Y = \frac{x + (2.2a + xa)^2}{x + 1}$	$a = \begin{cases} 5 + 2x & (x \leq 3) \\ 4 & (3 < x \leq 7) \\ x - 1 & (x > 7) \end{cases}$
5	$Y = 3x + a^2 - \frac{x}{2ax}$	$a = \begin{cases} \sin x + 2 \cos x & (-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}) \\ \frac{x - 2.5}{2 + x} & \text{для остальных } x \end{cases}$
6	$Y = 7.3 - \frac{a}{(1 + x)} + x^3 a$	$a = \begin{cases} 2x + 1 & (x \leq 2) \\ \sqrt{x + 3} & (x > 2) \end{cases}$
7	$Y = a + 2x + \frac{3x^2 + a^2}{a + x}$	$a = \begin{cases} 3.2x + 1 & x < 3.14 \\ x \sin x & (x = 3.14) \\ x & (x > 3.14) \end{cases}$
8	$Y = \frac{2}{x} + a^3 + e^x$	$a = \begin{cases} x^2 + \frac{4}{\sin x + 2} & (-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}) \\ 5 & (\frac{\pi}{2} < x \leq 2.5) \\ x + 1 & (x > 2.5) \\ \text{не определено} & (x < -\frac{\pi}{2}) \end{cases}$
9	$Y = a + 2.8x + \frac{x + a}{3 - x}$	$a = \begin{cases} 4x + 2.5 & (x \leq 2.5) \\ 1.5x + 8 & \text{для остальных } x \end{cases}$

10	$Y = a + \frac{a}{a+1} + 2.5x$	$a = \begin{cases} 3.5x & (x \geq -2) \\ x+1.5 & (x < -2) \end{cases}$
11	$Y = 2.5a^2 + \sqrt{a+x}$	$a = \begin{cases} 5 & (x < 3) \\ 1.5x+8 & (x \geq 3) \end{cases}$
12	$Y = a - \frac{x^2}{a}$	$a = \begin{cases} 3.5x & (x < 4) \\ x-5 & (x \geq 4) \end{cases}$
13	$Y = \frac{2+x}{x} + a^3 - 3x$	$a = \begin{cases} 3.7x & (-6 < x \leq -3) \\ x^2 + 3x - 3 & (-3 < x \leq 8) \\ \text{не определено для остальных } x \end{cases}$

Лабораторная работа №3. Программирование циклов

Циклом называется алгоритмическая структура, выполняющая какие-либо повторяющиеся операции. Для организации циклов в C++ используются операторы:

1. **while**

```
while ( условие )
{
    тело цикла;
}
```

Пока условие истинно, выполняется тело цикла. Условие может быть сложным, например: **a>b && b>0 && a<MAX**. Если нужно перейти на следующую итерацию цикла до завершения выполнения всех операторов тела цикла. Для этого используют оператор **continue**. Чтобы выйти за пределы цикла, даже если условие продолжения цикла выполняется, используют оператор **break**.

2. **do ... while**

При использовании этой конструкции условие проверяется после выполнения тела цикла. Это гарантирует выполнение операторов цикла как минимум один раз.

```
do
{
    тело цикла;
} while (условие);
```

3. for

Оператор for объединяет сразу три операции: инициализация, проверка условия, приращение счётчика цикла. Счётчиком называется переменная, подсчитывающая количество выполнений цикла.

```
for (int i=0; i<5; i++)
{
    действие 1;
    действие N;
}
```

В случае множественной инициализации и приращения запись оператора будет выглядеть следующим образом:

```
#include <iostream.h>
int main()
{
    for (int i=0, j=0; i<3; i++, j++)
        cout << "i: " << i <<"j: " << j << endl;
    return 0;
}
```

Проверьте на компьютере, что выполняет данная функция.

Кроме того, циклы могут быть вложенными:

```
for (int i=0; i<row; i++)
{ for (int j=0; j<column; j++)
    {
        действие 1;
        действие 2;
    }
    действие 3;
}
```

Замечание 1. Функции в языке Visual C++. Функция, по сути своей, это подпрограмма, которая манипулирует с данными и возвращает некоторое значение. Каждая программа на C++ имеет как минимум одну функцию **main()**, которая при запуске вызывается автоматически. Функция **main()** может вызывать другие функции, которые могут вызывать следующие функции и т.д.

Существует два вида функций: функции, определяемые пользователем и встроенные (стандартные), которые являются составной частью пакета компилятора. Для использования функции в программе требуется, чтобы функция сначала была объявлена (прототип функции), а затем определена (тело функции). Существует три способа объявления функции: 1) запись прототипа функции в файл, а затем с помощью команды **#include имя_файла** включить его в свою программу; 2) запись прототипа функции в файле, в котором эта функция и используется; 3) определение функции одновременно с её объявлением. Пример объявления функции:

```
int FindArea (int length, int width);
```

сначала указывается тип возвращаемого значения *int*, затем имя функции *FindArea*, затем в скобках типы и имена параметров функции через запятую. Значения параметров можно инициализировать в объявлении прототипа. Например:

```
int Area (int W=25, int H);
```

Допускается задание прототипа функции без имён параметров, а только указывая их типы, например:

```
long Area (int, int);
```

Определение функции состоит из заголовка и тела функции. В заголовке функции указывается тип возвращаемого значения, имя функции и в скобках - типы и имена параметров через запятую. Обратите внимание что в конце заголовка функции **не ставится** точка с запятой (;) как это обычно принято в C++. Далее открывается фигурная скобка и записывается тело функции, которое определяет то, что данная функция выполняет. Оператор **return** возвращает значение функции. Необходимо следить, чтобы тип возвращаемого значения соответствовал типу, указанному в объявлении. После закрывающей тело функции фигурной скобки также **отсутствует** точка с запятой. Пример:

```
int FindArea (int length, int width) // заголовок функции  
{  
    return (length*width);  
}
```

В функцию можно не только передавать значения переменных, но и объявлять переменные внутри тела функции. Эти пере-

менные существуют только внутри самой функции и называются локальными. Когда выполнение программы передаётся обратно из функции к основному коду, локальные переменные удаляются из памяти. Глобальные же переменные имеют глобальную область видимости и доступны из любой точки программы. Пример: создание функции, переводящей температуру из шкалы по Фаренгейту в градусы Цельсия.

```
#include <iostream.h>
float Convert(float); // прототип функции
int main()
{
    float TempFar;
    float TempCel;
    cout<< "Enter the Temperature in Fahrenheit: ";
    cin >> TempFar;
    TempCel = Convert (TempFar);
    cout << "\nThe Temperature in Celsius is: ";
    cout << TempCel << endl;
    return 0 ;
}
float Convert(float Fer) /* заголовок функции. Обратите
                           внимание на отсутствие ; в конце */
{
    float Cel; // тело функции
    Cel = ((Fer - 32) * 5)/9; // тело функции
    return Cel; // тело функции
} // конце строки тоже нет точки с запятой
```

Возврат значения из функции осуществляется оператором **return**. Например, **return (x>5)** будет возвращать **false**, если x не больше 5, или **true**, если x больше 5. Функция может содержать несколько операторов return. Например:

```
int Doubler (int origin)
{
    if (origin <= 1000)
        return origin*2;
    else
```



```

    return -1;
}

```

В языке C++ предусмотрена возможность создания нескольких функций с одинаковым именем (полиморфизм функций или перезагрузка функций). Перегруженные функции должны отличаться друг от друга списком параметров: либо типом параметров, либо их количеством. Например:

```

int myFunction (int, int);
int myFunction (long, long);
int myFunction (long);

```

Если функция объявлена с ключевым словом **inline**, компилятор не будет записывать функцию в отдельном месте в памяти компьютера, а копирует её строки непосредственно в код программы по месту вызова. Это приводит к увеличению скорости выполнения программы и одновременно увеличивает объем памяти программы. Для объявления встраиваемой функции (**inline функции**) необходимо в прототипе добавить слово **inline** перед типом возвращаемого значения.

```

inline int Doubler (int);

```

всё остальное, а именно: заголовок и тело функции, и её вызов в основной программе остаются прежними.

Замечание 2. Рекурсия. Стоит отметить, что внутри функции нельзя определить другую функцию, но из одной функции можно вызвать сколько угодно других функций. Функции могут даже вызывать самих себя, что именуется рекурсией. Рекурсия бывает двух видов: прямая, когда функция вызывает сама себя, и косвенная, когда функция вызывает другую функцию, которая вызывает затем первую функцию. Пример: программа вычисления чисел, образующих ряд Фибоначчи: первые два числа ряда равны единице и каждый последующий член ряда является суммой двух предыдущих (1, 1, 2, 3, 5, 8, 13, 21, 34 ... и т.д.). Для вычисления очередного члена ряда Фибоначчи используем рекурсию:

```

#include <iostream.h> // пример рекурсии
int fib(int n); /* прототип функции вычисления члена ряда
Фибоначчи */
int main()

```

```

{
    int n, answer;
    cout<< "Enter number to find: ";
    cin >> n;
    cout << "\n\n";
    answer = fib(n);
    cout<< answer<< " is the " <<n <<"-th member of Fi-
bonacci series\n";
    return 0 ;
}
int fib(int n) /* заголовок функции вычисления члена ряда
Фибоначчи */
{
    if(n<3)
        return 1;
    else
        return( fib(n-2) + fib(n-1));
}

```

***Пример программирования циклов по заданиям к лабора-
торной работе №3.***

Используя циклы, функции и рекурсии вычислим с точно-
стью $eps=0,0001$ сумму $\sum_{i=1}^{\infty} (-1)^{i+1} \cdot \frac{x^{2(i-1)}}{2(i-1)!}$, которая является раз-
ложением в ряд Тейлора функции $\cos(x)$, при изменении x в диа-
пазоне $[0.1, 1]$ с шагом 0.1.

```
#include <iostream.h>
```

```
#include <math.h>
```

```
double factorial(int); /* прототип функции вычисления фак-
ториала */
```

```
int main()
```

```
{
```

```
double a,b,h,eps,S;
```

```
a=0.1; b=1; h=0.1; eps=1e-4;
```

```
for (double i=a; i<=b; i+=h)
```

```
{
```

```

int j=1;
double st,f;
S=0;
do
{
f = factorial(2*(j-1));
st = pow(-1,(j+1))*pow(i,(2*(j-1)))/f;
j++;
S += st; // равнозначно S = S+st
}while(st>=eps);
cout << "Sum of series is:" << S << " Value of
cos("<<i<<") is: "<< cos(i) << endl;
}
return 0;
}
double factorial(int a)
{
if (a==0||a==1) /* заметьте, что здесь стоит
знак равенства == , а не присвоить = */
return 1;
else
return (a*factorial(a-1));/* для вычисле-
ния использована рекурсия */
}

```

Задания. Вычислить сумму S при изменении аргумента x в указанном диапазоне $[a,b]$ с шагом h ., прекращая суммирование, когда очередной член станет меньше 0.0001. Для проверки в каждой точке вычислите также функцию $y=f(x)$, представляющую собой аналитическое выражение ряда. Варианты приведены в Табл.3.

Таблица 3

Варианты заданий к лабораторной работе №3

№ варианта	S=	y=	a=	b=	h=
1	$\sum_{i=1}^{\infty} i^2 \cdot x^{i-1}$	$\frac{1+x}{(1-x)^3}$	0	0.9	0.1

2	$\sum_{i=1}^{\infty} (2i-1)^2 \cdot x^{i-1}$	$\frac{1+6x+x^2}{(1-x)^3}$	0	0.9	0.1
3	$\sum_{i=1}^{\infty} (-1)^{i+1} \cdot \frac{x^{2i-1}}{(2i-1)!}$	$\sin x$	0.1	0.9	0.1
4	$\sum_{i=1}^{\infty} (-1)^{i+1} \cdot \frac{x^{2(i-1)}}{2(i-1)!}$	$\cos x$	0.1	1	0.1
5	$\sum_{i=0}^{\infty} \frac{(2x)^i}{i!}$	e^{2x}	0.1	1	0.05
6	$\sum_{i=0}^{\infty} \frac{x^{2i}}{(2i)!}$	$chx = \frac{e^x + e^{-x}}{2}$	0.1	1	0.05
7	$\sum_{i=0}^{\infty} \frac{x^{(2i+1)}}{(2i+1)!}$	$shx = \frac{e^x - e^{-x}}{2}$	0.1	1	0.05
8	$\sum_{i=0}^{\infty} (-1)^i \cdot \frac{x^{(2i+1)}}{(2i+1)}$	$arctgx$	0.1	0.5	0.05
9	$\sum_{i=0}^{\infty} (2i+1) \cdot x^i$	$\frac{1+x}{(1-x)^2}$	0	0.9	0.05
10	$\sum_{i=1}^{\infty} (-1)^i \frac{\cos(ix)}{i^2}$	$\frac{x^2 - \frac{\pi^2}{3}}{4}$	-0.5	0.5	0.1
11	$\sum_{i=1}^{\infty} (-1)^{i+1} \frac{(x)^i}{i}$	$\ln(1+x)$	-0.5	0.5	0.1
12	$\sum_{i=1}^{\infty} (-1)^{i+1} \cdot \frac{x^{2i}}{2i(2i-1)}$	$x \cdot arctg(x) - \ln \sqrt{1+x^2}$	0.1	0.8	0.05
13	$\sum_{i=0}^{\infty} \frac{(2i+1)x^{2i}}{i!}$	$(1+2x^2) \cdot e^{x^2}$	0.1	1	0.1
14	$\sum_{i=0}^{\infty} \frac{\left(\frac{x-1}{x+1}\right)^{2i+1}}{(2i+1)}$	$\frac{\ln x}{2}$	0.2	1	0.08

Лабораторная работа №4. Понятие об объектно-ориентированном программировании. Базовые классы

Класс – это набор переменных различных типов, скомбинированный с набором связанных функций. Инкапсуляцией называется соединение переменных и функций в один объект класса. Части программы, работающие с классом, могут использовать ваш объект, не беспокоясь о том, что находится в нём или как оно работает.

Переменные в классе принадлежат только своему объекту и называются ещё переменными-членами или данными класса.

Функции в классе выполняют действия с переменными-членами и называются методами класса.

Например, объявление класса **Cat** делается так:

```
Class Cat
{
int Age; // Данные класса
int Weight; // Данные класса
void Meaw(); // Метод класса
};
```

Объект нового типа определяется также как и любая переменная. Например:

```
int Skill;
Cat Murzik;
```

После этой записи **Murzik** является объектом класса **Cat**. Доступ к членам объекта (как переменным, так и методам) осуществляется оператором прямого доступа (**.**). Чтобы присвоить число переменной-члену **Weight** объекта **Murzik** запишем:

```
Murzik.Weight = 12;
```

Аналогично, чтобы вызвать метод **Meaw**, запишем:

```
Murzik.Meaw();
```

Определенные значения должны присваиваются объектам, а не классам. Поэтому запись: **Cat Age=5;** - ошибочна. Синтаксически правильно будет:

```
Cat Murzik;
Murzik.Age = 5;
```

При объявлении класса используются ключевые слова **pub-**

lic (открытый) и **private** (закрытый), определяющие порядок доступа к членам класса. Все члены класса (переменные и методы) являются закрытыми по умолчанию. К закрытым членам можно получить доступ только с помощью методов самого класса. Открытые члены доступны для всех других функций программы. Например, при записи:

```
Cat Barsik;  
Barsik.Age = 3;
```

компилятор выдаст ошибку, т.к. нельзя обращаться к закрытым переменным объекта. Выходом из положения является объявление класса открытым:

```
Class Cat  
{  
    public:  
    int Age;  
    int Weight;  
    void Meaw();  
};
```

Общая стратегия использования классов состоит в том, чтобы оставлять переменные-члены класса закрытыми. Это позволяет инкапсулировать данные внутри класса. Доступ следует открывать только методам класса, которые обеспечивают доступ к закрытым данным. Эти методы можно вызвать из любого места в программе для возвращения или установления значения закрытых данных. Такая стратегия позволяет скрыть детали хранения данных в объектах, снабжая пользователя методами их использования. Можно модернизировать способы хранения и обработки данных внутри класса, не переписывая при этом методы доступа и вызова их во внешнем программном коде. Например:

```
class Cat  
{  
    public:  
        int GetAge();  
        void SetAge(int Age);  
        void Meaw();  
    private:
```

```

    int itsAge;
};

```

Чтобы установить возраст кота Мурзика теперь нужно записать:

```

Cat Murzik;
Murzik.SetAge(5);

```

Как и для функций, после объявления класса нужно дать определение класса, т.е. определить, что он выполняет. Чтобы разобраться, выполним такой пример:

```

#include <iostream.h> // пример класса
class Cat // объявляем класс Cat
{
    public:
    int GetAge(); // метод доступа
    void SetAge(int age); // метод доступа
    void Meow(); // обычный метод
    private:
        int itsAge; // переменная-член (данные) метода
};
int Cat::GetAge() // определение метода GetAge,
{
    return itsAge; /* который возвращает значение переменной-члена метода */
}
void Cat::SetAge(int age) // определение метода SetAge,
{
    itsAge = age; /* который инициализирует переменную-член метода */
}
void Cat::Meow() // метод выводит на экран тест сообщения
{
    cout<< "Meow\n";
}
int main()
{

```

```

Cat Murzik; // создаём объект Мурзик
Murzik.SetAge(5); // устанавливаем его возраст
Murzik.Meow(); /* заставляем его мяукнуть и выводим
сообщение: */
cout<< "Murzik has a " << Murzik.GetAge() <<" years
old.\n";
return 0;
}

```

Для инициализации и разрушения переменных-членов класса используются специальные функции: конструктор и деструктор. Существуют специальные методы класса, имена которых совпадают с именем самого класса и которые обязательно должны присутствовать в коде и добавляются компилятором по умолчанию. Это конструктор и деструктор класса. Причём конструктор будет пустым, если инициализация переменных – членов класса не проводится. Деструктор служит для удаления из памяти отработавших объектов, а также других действий при модификации объекта или класса. Он не принимает никаких аргументов и не возвращает никаких значений. Объявляется с помощью значка тильда (~), например: **~Cat()**;

Пример, дополняющий предыдущий и позволяющий понять роль конструктора:

```

#include <iostream.h> // классы и конструктор/деструктор
class Cat // объявляем класс Cat
{
    public:
        Cat(int initAge); // конструктор
        ~Cat(); // деструктор
        int GetAge();
        void SetAge(int age);
        void Meow();
    private:
        int itsAge; // переменная-член метода
};
Cat::Cat(int initAge) /* конструктор инициализирует пере-
менную-член значением */

```



```

{
    itsAge = initAge;
}
Cat::~~Cat() // деструктор не выполняет действия
{}
int Cat::GetAge()
{
    return itsAge;}
void Cat::SetAge(int age)
{
    itsAge = age; }
void Cat::Meow()
{
    cout<< "Meow\n";}
int main()
{
    Cat Murzik(5); /* создаём объект Мурзик с инициализированным возрастом 5 */
    Murzik.Meow(); /*заставляем его мяукнуть и выводим сообщение*/
    cout<< "Murzik has a " << Murzik.GetAge() <<" years old.\n";
    Murzik.Meow();
    Murzik.SetAge(7); /* переустанавливаем его возраст и выводим сообщение: */
    cout<< "Now Murzik has a " << Murzik.GetAge() <<" years old.\n";
    return 0;
}

```

В языке C++ предусмотрена возможность объявить метод класса таким образом, что такому методу будет запрещено изменять значение переменных-членов класса. Для этого используется ключевое слово `const` после круглых скобок, но перед точкой с запятой. Например: **void Function() const;**

Использование `const` везде в объявлениях методов, если они не изменяют переменные объекта, позволяет лучше отслеживать

ошибки. Если метод объявлен как `const`, а в его выполнении происходит изменение переменной-члена объекта, то компилятор выдаст ошибку.

Т.к. функция **GetAge()** прошлой программы просто возвращает текущее значение переменной **itsAge**, то правильнее будет записать: **int GetAge() const;**

Можно также помещать определение методов класса непосредственно в объявлении класса. Обратите внимание на отсутствие точки с запятой в этом случае после определения.

Например:

```
class Cat
{
public:
int GetAge() {return itsAge;} // определение в объявлении
void SetAge(int age);
};
```

Можно строить сложные классы путём объявления более простых классов и последующего их включения в объявление сложного класса. Т.е. классы могут содержать другие классы в качестве переменных-членов.

Очень близкими родственниками ключевого слова **class** являются структуры, объявляемые ключевым словом **struct**. В C++ структура – это тот же класс, но с открытыми по умолчанию членами. Структуры перешли в C++ по наследству из языка C.

Пример, включающий изученные выше понятия языка C++.

Разработаем программу, позволяющую вычислять объём полого цилиндра при разных исходных данных.

```
#include <iostream.h> /* подключение стандартной библиотечки ввода/вывода */
#include <math.h> /* подключение стандартной библиотеки математических операций для функции возведения в степень b числа a: pow(a,b) */
enum choice {Size=1, Volume, Thickness, Quit};
/* перечислимый тип переменной */
class Cylinder // объявление класса
{ public:
```

```

    Cylinder(float R1, float R2, float H); // конструктор класса
    ~Cylinder() {} // деструктор
    float GetExtRadius() const {return itsR1;} /* метод доступа
класса сразу с его определением */
    float GetIntRadius() const {return itsR2;} /* метод доступа
класса сразу с его определением */
    float GetHeight() const {return itsH;} /* метод доступа клас-
са сразу с его определением */
    float GetVolume() const; // метод расчета объёма цилиндра
    float GetThickness() const {return (itsR1 - itsR2);} /* метод
класса сразу с его определением */
    void SetSize (float newR1, float newR2, float newH); /* ме-
тод задания размеров цилиндра */
    private: // закрытые члены класса (переменные)
    float itsR1;
    float itsR2;
    float itsH;
};
// определение методов класса:
Cylinder::Cylinder (float R1, float R2, float H)
/* определение конструктора*/
{
    itsR1 = R1;
    itsR2 = R2;
    itsH = H; }
void Cylinder::SetSize(float newR1, float newR2, float
newH) /* определение метода класса */
{itsR1 = newR1;
itsR2 = newR2;
itsH = newH; }
float Cylinder::GetVolume() const
// определение метода класса
{
    float V;
    // формула расчёта объёма цилиндра
    V = 3.1415 * (pow(itsR1,2) - pow(itsR2,2)) * itsH;
    return V;
}

```

```

    }
// прототипы используемых функции
    int DoMenu(); // функция вывода меню на экран
    // функция вызова метода расчёта объёма
    void DoVolume(Cylinder);
    // функция вызова метода расчёта толщины стенки^
    void DoThickness(Cylinder);
    int main() // основное содержание программы, функция main
    {
    //инициализация конструктором параметров цилиндра
    Cylinder theCyl(0,0,0);
    int choice = Size;
    int fQuit = false;
        while(!fQuit)
            {
                choice = DoMenu();/* функция DoMenu выводит
Меню и возвращает сделанный там выбор пункта */
                if (choice<Size || choice > Quit)
                    {cout<<"\nInvalid choice. Try again.\n\n";
                    continue;} /* переход на следующую итера-
цию тела цикла без выполнения тела цикла */
                switch(choice)
                {
                case Size:
                    float L1,L2,Hi;
                    cout<<"\nEnter external radius of Cylinder: ";
                    cin>> L1;
                    cout<<"Enter internal radius of Cylinder: ";
                    cin>> L2;
                    cout<<"Enter height of Cylinder: ";
                    cin>> Hi;
                    theCyl.SetSize(L1, L2, Hi);
                    break; /* выход из выполнения оператора
                    switch */
                case Volume:
                    DoVolume(theCyl);

```

```

        break;
        case Thickness:
            DoThickness(theCyl);
            break;
        case Quit:
            fQuit = true;
            cout<<"\nExiting...\n\n";
            break;
        default:
            cout<< "Error in choice!\n";
            fQuit=true;
            break;
    } // конец switch
} // конец while
return 0;
}
// определение тел функций

int DoMenu() // тело функции вывода меню и выбора его пункта
{
    int choice;
    cout<< "\n ***Menu***\n";
    cout<< "(1) Set Size\n";
    cout<< "(2) Count Volume\n";
    cout<< "(3) Count Thickness\n";
    cout<< "(4) Quit\n";
    cout<< "Choice a number from Menu: ";
    cin>> choice;
    return choice;
}

void DoVolume(Cylinder theCyl) /* тело функции вызова метода
расчёта объёма */
{cout<<"The Volume of Cylinder: "<<the-
Cyl.GetVolume()<< endl;}
void DoThickness(Cylinder theCyl) /* тело функции вызова ме-
тода расчёта толщины стенки */

```

{cout<<"The Thickness of Cylinder Wall: "<< the-
Cyl.GetThickness()<<endl;}

Задание. Самостоятельно разработать программу, позволяющую многократно вычислять некоторую физическую величину по заданной формуле и одновременно вычислять значения каких-либо параметров формулы. Варианты заданий по лабораторной работе приведены в Табл.3.

Таблица 4

Варианты заданий по лабораторной работе №4

№	Наименование	Формула	Параметр
	Объём:		
1	тора	$V = 2\pi^2 Rr^2$	r
2	усеченного круглого цилиндра	$V = \pi R^2 \frac{h_1 + h_2}{2}$	R
3	усеченного прямого конуса	$V = \frac{\pi h}{3} (R^2 + Rr + r^2)$	h
4	шарового сектора	$V = \frac{2\pi R^2 h}{3}$	R
5	шарового сегмента	$V = \frac{1}{3} \pi h (3R - h)$	h
6	полого цилиндра (Oz – ось цилиндра)	$J_Z = \frac{1}{2} (R^2 + r^2)$	R-r
	Центральный момент инерции:		
7	полого цилиндра (Ox – ось цилиндра)	$J_X = \frac{1}{2} (3R^2 + 3r^2 + H^2)$	R-r
8	полого шара	$J = \frac{2}{5} \frac{(R^5 - r^5)}{R^3 - r^3}$	R-r
9	шарового сектора	$J = \frac{h}{5} (3R - h)$	h

10	усеченного прямого конуса	$J = \frac{3}{10} \frac{(R^5 - r^5)}{R^3 - r^3}$	h
11	тора	$J_Z = (R^2 + \frac{3}{4}r^2)$	r
12	Импеданс последовательной цепи RLC	$Z_- = \sqrt{R^2 + (\omega L - \frac{1}{\omega C})^2}$	R
13	Импеданс параллельной цепи RLC	$Z_{//} = \frac{1}{\sqrt{\frac{1}{R^2} + (\omega C - \frac{1}{\omega L})^2}}$	R
14	Емкость плоского конденсатора $\epsilon_0 = 8,85 \cdot 10^{-12}$ Ф/м	$C = \frac{\epsilon \epsilon_0 S}{d}$	d
15	Сила отталкивания двух электронов $e = 1,6 \cdot 10^{-29}$ Кл	$F = \frac{1}{4\pi \epsilon_0} \frac{e^2}{r^2}$	r

ЗАКЛЮЧЕНИЕ

В конце лабораторной 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. Полный программный код с комментариями
3. Результаты тестирования и результаты работы программы

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

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

Подписано в печать 00.00.2009. Формат 60x84 1/16.
Бумага офсетная. Печать офсетная.
Усл. печ. л.00,00 Усл. кр.-отг. 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