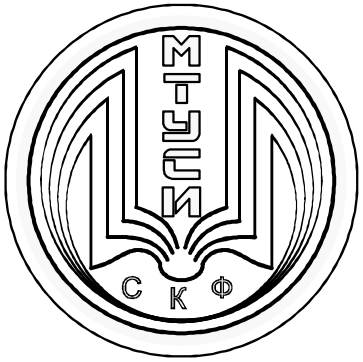
ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
СЕВЕРО-КАВКАЗСКИЙ ФИЛИАЛ
ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО
БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО
ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
СВЯЗИ И ИНФОРМАТИКИ»



КАФЕДРА ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

П.В. Лобзенко

Учебное пособие по дисциплине

РАЗРАБОТКА КРОССПЛАТФОРМЕННЫХ ПРИЛОЖЕНИЙ С++

Практикум

Ростов-на-Дону

2019 г.

УДК 004.6

ББК 32.973.2-018

Лобзенко П.В. Учебное пособие по дисциплине Разработка кроссплатформенных приложений C++: практикум. - Ростов-на-Дону:
Северо-Кавказский филиал МТУСИ, 2019. – 71 с.: 10 ил.

Учебное пособие содержит теоретический материал, снабженный примерами выполнения заданий по разработке кроссплатформенных приложений на языке программирования C++.

Рекомендовано для использования при проведении занятий по дисциплине «Разработка кроссплатформенных приложений C++», а также при самостоятельном изучении материалов по дисциплине для студентов очной и заочной форм обучения направления подготовки 09.03.01 «Информатика и вычислительная техника» (профиль «Программное обеспечение и интеллектуальные системы»)

Составитель: П.В. Лобзенко, доцент кафедры ИВТ
Рассмотрено и одобрено
на заседании кафедры ИВТ
Протокол от «26» августа 2019 г. № 1

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время основным трендом в разработках программного обеспечения (ПО) различного назначения для стационарных и мобильных устройств является его явная кроссплатформенная направленность.

Кроссплатформенность означает, что написанная и откомпилированная программа может запускаться под управлением различных операционных систем (ОС) или, как еще называют, платформ. Идея кроссплатформенности состоит в том, чтобы один раз написанный код программы запускался бы на различных платформах и не требовал переработки при переходе от платформы к платформе. Имеются ввиду такие распространенные платформы, как Mac OS, Windows, Linux, iOS, Android.

Для реализации кроссплатформенности используются соответствующие среды разработки программ, другими словами, это специфические библиотеки, которые поддерживают программирования на указанных платформах. Т.е., наличие такой библиотеки позволяет программы, написанные в одной ОС использовать в другой безо всякой адаптации.

Таких сред достаточно много. Укажем для примера две из них. Это, одна из самых распространенных сред разработки кроссплатформенных приложений - редактор Qt [1]. Другой недавней разработкой является фреймворк Juse [2].

Обе разработки являются библиотеками для языка C++ применимыми в различных ОС. Поэтому, в настоящем пособии приведены основы программирования на указанном языке и описаны подходы разработки кроссплатформенных приложений, являющихся результатом выполнения заданий практических занятий. Приведена также методика решения задач указанных занятий.

Первый раздел пособия посвящен основам написания программ на языке программирования C++. Здесь рассмотрены основные управляющие конструкции языка, работа с функциями, или как их еще называют – методами пользователя. Также, уделено внимание основам объектно- ориентированного програм-

мирования и разработке приложений с графическим интерфейсом пользователя (ГИП). Материал приведен обзорно в расчете на то, что основы языка уже изучены и освоены практически.

Во втором разделе пособия содержатся пояснения и методика выполнения типовых заданий каждого из практических занятий по дисциплине «Разработка кроссплатформенных приложений C++».

Материал описан достаточно подробно и снабжен необходимыми ссылками, иллюстрациями и примерами составления программ.

1 Основы языка C++

В этом разделе пособия приведены сведения теоретические и практические, которые необходимы для составления программ на языке C++ для выполнения заданий практических занятий по дисциплине «Разработка кроссплатформенных приложений C++».

В качестве напоминания структуры программы в C++ приведем совершенно простую программу, где вводится значение переменных с клавиатуры, производятся элементарные действия и выводится на экран результат этих действий (листинг 1.1).

Листинг 1.1- Элементарная программа на C++

```
#include <iostream>
#include <math.h>
using namespace std;
int main() {
 int x,y,z;
 cout<<"x, y";
 cin>>x>>y;
 cout<<"\n";
 z=x+y;
 cout<<"Sum of x+y = " << z<<endl;
}
```

Программа на языке C++ имеет четкую структуру, также как, например, в языке Pascal. Так, программа начинается с, так называемых, директив препроцессора, которые нужны для того, чтобы подключить нужные библиотеки или уже откомпилированные модули. Директивы начинаются специальным знаком и оператором - #include. В указанной программе их две. Вообще, в программе на C++ должны быть явно указаны все используемые библиотеки. Библиотека

iostream нужна для организации ввода с помощью инструкции cin и вывода — с помощью cout. Ниже в программе видно, в операторе ввода – cin поток ввода записывается в виде знаков «больше» >> и через запятую указываются все переменные, которые нужно ввести. В данном примере – это значения переменных x и y. Они вводятся через пробел, а в конце нажимается «Enter» и набранные на клавиатуре значения присваиваются этим переменным. Поток вывода, напротив, обозначается знаками «Меньше» << и в двойных кавычках записывается текст, который будет выведен на экран, далее через обозначения потока указываются имена тех переменных, значения которых нужно вывести. В завершении строки ставится может ставится специальный символы "\n" или endl, которые завершают строку вывода и после вывода значений на экран курсор будет переведен на новую строку.

Библиотека math.h содержит методы обработки различных математических величин и выражений. Стандартные математические функции языка C++ приведены в таблице 1.1.

Таблица 1.1- Стандартные математические функции

Обозначение	Действие
abs(x)	Модуль целого числа *x*
fabs(x)	Модуль вещественного числа *x*
sin(x)	Синус числа *x*
cos(x)	Косинус числа *x*
tan(x)	Тангенс числа *x*
atan(x)	Арктангенс числа *x*, *x* ∈(-)
acos(x)	Арккосинус числа *x*
asin(x)	Арксинус числа *x*
exp(x)	Экспонента, e^x
log(x)	Натуральный логарифм,($x > 0$)
log10(x)	Десятичный логарифм, ($x > 0$)
sqrt(x)	Корень квадратный, ($x > 0$)
pow(x,y)	Возведение числа *x* в степень *y*
ceil(x)	Округление числа *x* до ближайшего большего целого
floor(x)	Округление числа *x* до ближайшего меньшего целого

1.1 Управляющие операторы C++

Далее рассмотрим основные управляющие операторы языка C++.

Фактически, к операторам управления можно, в первую очередь, отнести те из них, которые изменяют ход вычислений при решении задачи. Это, несомненно, оператор условного перехода (конструкция if…else), или, как его еще называют: оператор выбора «один из двух». Сюда же отнесем и его модификацию – оператор выбора «один их многих» (конструкция switch…case). И, конечно же, к управляющим операторам принадлежат операторы цикла: с предусловием (while), с постусловием (do… while) и оператор цикла «со счетчиком» (for). Они обеспечивают различные варианты множественных вычислений путем их повторения до тех пор, пока условие выполнения цикла остается верным.

1.1.3 Условный оператор

При решении большинства задач порядок вычислений зависит от определённых условий, например, от исходных данных или от промежуточных результатов, полученных на предыдущих шагах программы. Для организации вычислений в зависимости от какого-либо условия в C++ предусмотрен условный оператор if, который в общем виде записывается следующим образом:

if (условие) оператор_1 ; **else** оператор_2 ;

где условие — это логическое (или целое) выражение, переменная или константа, оператор_1 и оператор_2 — любой оператор языка C++.

Работает условный оператор следующим образом. Сначала вычисляется значение выражения, указанного в скобках после **if** (если) и представляющее собой условие. Если оно не равно нулю, т.е. имеет значение «истина» (true), выполняется оператор_1. В противном случае, когда выражение в скобках равно нулю, т.е. имеет значение «ложь» (false), выполняется оператор_2.

Блок-схема алгоритма полного условного оператора if, т.е. содержащего прямую ветвь (от if до else) и побочную ветви (после else) представлен на рисунке 1.1.

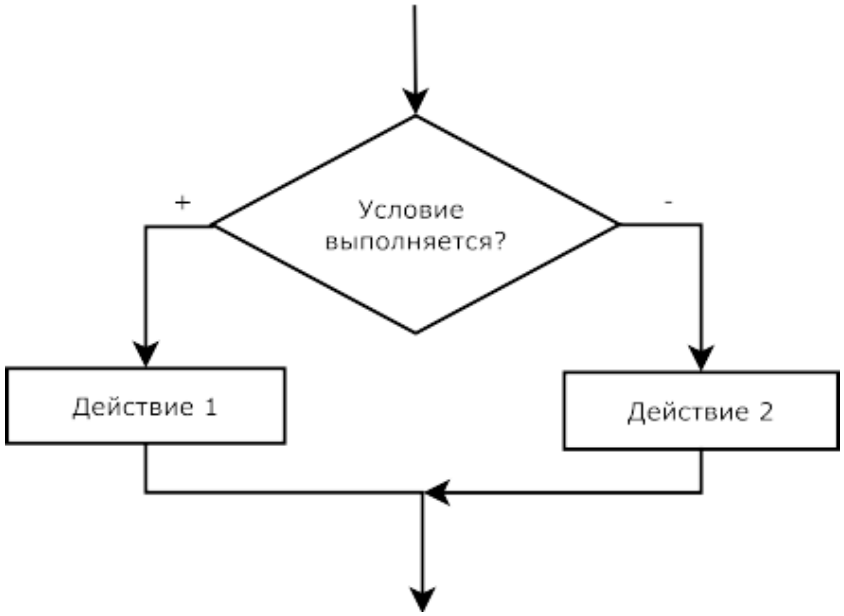


Рисунок 1.1- Алгоритм полного условного оператора
(конструкция if…else)

Например, чтобы сравнить значения переменных a и b нужно использовать полную конструкцию if…else (листинг 1.1).

Листинг 1.1- Фрагмент кода для проверки двух переменных a и b на равенство

```
cin >>a ; cin >>b ;
if ( a==b ) cout<<" a равно b " ;
else cout<<" a не равно b " ;
```

Т.к., в этом фрагменте проверяется условие равенства переменных друг другу, то использовано «==» (два знака равно). Это в корне отличается от оператора присваивания «=» (один знак равенства) потому, что если написать в условии a=b, то синтаксической ошибки не будет, однако выполнение условного оператора будет полностью зависеть только от значения переменной b. При

b=1 операция присваивания формирует результат a=1 (не зависимо от того, чему переменная a была равна до этого) и это воспринимается компилятором как результат всего условия, т.е. «истина» (true) и на экране появится сообщение «a равно b», что не соответствует действительности (значение переменной a здесь не играет никакой роли и ни с чем не сравнивается!). Если же, b=0, то сработает побочная ветвь оператора и мы увидим сообщение «a не равно b», что тоже далеко от истины. При любых других значениях переменной b проверить условие вовсе не удастся и появится сообщение о логической ошибке всей конструкции.

Помимо проверки на равенство существуют еще связки для проверки условий на не равенство «!=», больше или равно «>=» и меньше или равно «<=». Для записи нескольких условий подряд используются связки && - «И», ||- «ИЛИ». Так, проверка условия «попадания» переменной X в один из интервалов от 0 до 5 или -1 до -8 будет записано в коде программы следующим образом: if ((x>=0 && x<=5) || (x>=-8 && x<=-1)) cout<<”Сообщение”. Обратите внимание, что составное условие все записывается в круглых скобках и его отдельные части тоже.

Если в задаче требуется, чтобы в зависимости от значения условия выполнялся не один оператор, а несколько, их необходимо заключать в фигурные скобки, как составной оператор. В этом случае компилятор воспримет группу операторов как один (листинг 1.2):

Листинг 1.2- Форма записи составного оператора в условном операторе

```
if ( условие )
{
оператор_1; оператор_2;
. . .
}
else
{
оператор_3; оператор_4;
. . .
```

```
}
```

Побочная ветвь else в условном операторе может отсутствовать, если в ней нет необходимости (листинг 1.3).

Листинг 1.3- Сокращенная запись оператора if

```
if    ( условие )    оператор;
или
if ( условие )
{
оператор_1; оператор_2;
...
}
```

В таком сокращенном виде условный оператор работает так: оператор (группа операторов) либо выполняется, либо пропускается, в зависимости от значения выражения, представляющего условие (рисунок 1.2).

Рисунок 1.2- Сокращенная запись оператора if

Еще одной формой условного оператора является, как было указано выше, оператор выбора «один из многих», или «переключатель» (switch с английского это «переключатель»).

Он необходим в тех случаях, когда в зависимости от значений какой-либо переменной надо выполнить те или иные операторы (листинг 1.4):

Листинг 1.4- Конструкция оператора «переключатель»

```
switch   (выражение)
{
case   значение_1:  {Операторы_1;}   break ;
case   значение_2:  {Операторы_2;}  break ;
case   значение_3:  {Операторы_3;}   break ;
...
case   значение_n:  {Операторы_n;}   break ;
default :    {Операторы;}   break ;
}
```

Оператор работает следующим образом. Вычисляется значение выражения (оно должно быть целочисленным). Если выражение принимает значение_1, то выполняются операторы_1. Если выражение принимает значение_2, то выполняются операторы_2 и так далее. Если выражение не принимает ни одно из значений, то выполняются операторы, расположенные после ключевого слова default (этот блок может отсутствовать в конструкции).

Оператор break необходим для того, чтобы осуществить выход из оператора switch. Если оператор break не указан, то будут выполняться следующие операторы из списка, несмотря на то, что значение, которым они помечены, не совпадает со значением выражения.

В качестве примера приведем программу для вывода на экран названия дня недели по вводу его номера при условии, что 1-й это понедельник (листинг 1.5).

Листинг 1.5- Пример использования оператора switch

```
#include <iostream >
using namespace std ;
int main( )
```

```
    { unsigned int R ; //Описано целое положительное число.
    cout<<" R = " ;   c i n >>R;   //Ввод числа от 1 до 7.
    switch (R)
    {
    case 1 : cout<<"Понедельник \ n " ; break ;
    case 2 : cout<<"Вторник \ n " ; break ;
    case 3 : cout<<"Среда \ n " ; break ;
    case 4 : cout<<"Четверг \ n " ; break ;
    case 5 : cout<<"Пятница \ n " ; break ;
    case 6 : co ut<<"Суббота \ n " ; break ;
    case 7 : cout<<"Воскресенье \ n " ; break ;
    default : cout<<"Введен несуществующий номер дня недели \ n "
; break ;
    }
    return 0 ;
    }
```

1.1.2 Операторы цикла

Цикл это повторение одних и тех же действий, называемых телом цикла. Один проход цикла называют шагом или итерацией. Переменные, которые изменяются внутри цикла и влияют на его окончание, называются параметрами цикла, или переменной цикла.

При написании циклических алгоритмов следует помнить следующее. Во-первых, чтобы цикл имел шанс когда-нибудь закончиться, содержимое его тела должно обязательно влиять на условие цикла. Во-вторых, условие должно состоять из корректных выражений и значений, определённых ещё до первого выполнения тела цикла. В C++ принято, чтобы цикл мог выполниться хотя бы один раз его условие, наложенное на переменную цикла, должно быть верным, т.е. возвращать значение 1 или «true».

В C++ для удобства пользователя предусмотрены три оператора, реализующих различные циклы: while, do...while и for.

1.1.2.1 Оператор цикла с предусловием

На рисунке 1.3 изображена блок-схема алгоритма цикла с предусловием.

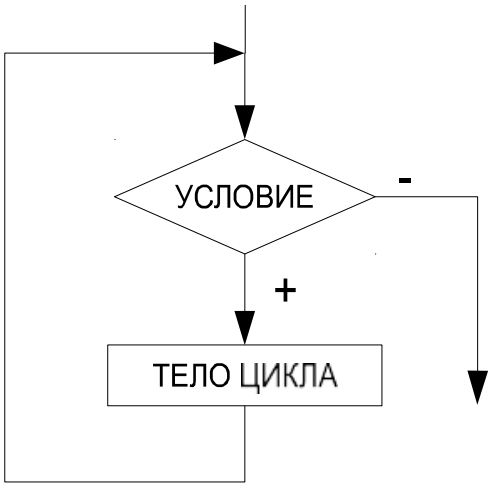


Рисунок 1.3- Блок-схема алгоритма цикла с предусловием

Оператор, реализующий этот алгоритм в C++, имеет вид:

```
while (условие)  оператор;
```

Здесь «условие» — логическое или целочисленное выражение, «оператор» — любой оператор языка C++.

Работает цикл с предусловием следующим образом. Вычисляется условие. Если оно истинно (не равно нулю), то выполняется оператор, и условие проверяется вновь. В противном случае цикл заканчивается, и управление передаётся оператору, следующему за телом цикла. Условие вычисляется перед каждой итерацией цикла. Если при первой проверке выражение равно нулю, цикл не выполнится ни разу. Тип выражения должен быть арифметическим или приводимым к нему.

Если тело цикла состоит более чем из одного оператора, необходимо использовать составной оператор (листинг 1.6):

Листинг 1.6- Конструкция цикла с предусловием с составным оператором – телом цикла

```
while   (условие)
{
оператор   1;
оператор   2;
...
оператор n;
}
```

Например, если нужно вывести на экран значения функции y=sin(2x) при x€[6; -4] с шагом 0.5, используется цикл с предусловием (листинг 1.7).

Листинг 1.7- Пример использования цикла с предусловием

```
#include <stdio.h>
#include <math.h>
using namespace std;
int main( )
{
float x, y;
x =6;        //Присваивание параметру цикла стартового значения
                        //Цикл с предусловием
while(x>=-4)//Пока параметр цикла превышает конечное значение
{                       //выполнять тело цикла
y=sin(2*x) ;            //Вычислить значение y
                        //Форматный вывод на экран пары x и y
printf("\t x =5.2f \t y =5.4f \n",x,y) ;
x -=0.5;              //Изменение параметра цикла
                    // (переход к следующему значению x)
}                  //Конец цикла
return 0;
}
```

1.1.2.2 Оператор цикла с параметром. Цикл for

Структура этого цикла включает в его заголовке все необходимые указания:

```
f o r  (начальные_присваивания; условие; последействие) опера-
тор;
```

Здесь обозначены: **начальные присваивания** — оператор или группа операторов, разделённых запятой, применяются для присвоения начальных значений величинам, используемым в цикле, в том числе параметру цикла, и выполняются один раз в начале цикла; **условие** — целое или логическое выражение, которое определяет условие входа в цикл, если условие истинно (не равно нулю), то цикл выполняется; **последействие** — оператор или группа операторов, разделённых запятой, которые выполняются после каждой итерации и служат для изменения параметра цикла; оператор — любой оператор языка, представляющий собой тело цикла. Последействие или оператор должны влиять на условие, иначе цикл никогда не закончится. Начальные присваивания, выражение или последействие в записи оператора for могут отсутствовать, но при этом «точки с запятой» должны оставаться на своих местах.

Название – «цикл с параметром» подчеркивает, что все описанное выше относится к переменной цикла, называемой «параметром», благодаря изменению которой выполняется цикл.

Если тело цикла состоит более чем из одного оператора, необходимо использовать составной оператор (рисунок):

```
f o r   (начальные присваивания;  условие;  последействие)
{
оператор_1;
...
оператор_n;
}
```

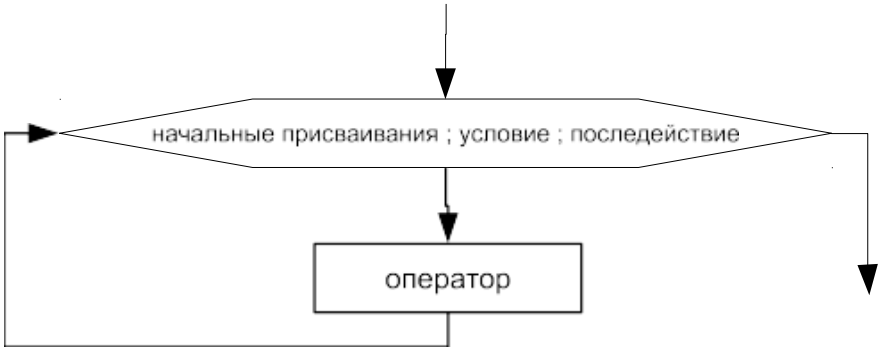


Рисунок 1.4- Блок-схема алгоритма цикла с параметром

Программа выполнения предыдущего примера (пункт 1.1.2.1) с помощью цикла с параметром приведена в листинге 1.8.

Листинг 1.8- Код программы с циклом for

```
#include <stdio.h>
#include <math.h>
using namespace std;
int main( )
{
float x, y;
                          //Цикл с параметром
for(x =6;x>=-4;x-=0.5)//Описание параметра цикла
{                        //Тело цикла
y=sin(2*x) ;             //Вычислить значение y
                         //Форматный вывод на экран пары x и y
printf("\t x =5.2f \t y =5.4f \n",x,y) ;
                       // (переход к следующему значению x)
}                   //Конец цикла
return 0;}
```

1.1.2.3 Цикл с постусловием

Цикл с постусловием реализован конструкцией:

```
do   оператор   while (условие) ;
```

здесь условие — логическое или целочисленное выражение, оператор — любой оператор языка C++. Если тело цикла состоит более чем из одного оператора:

```
do
{
оператор_1; оператор_2;
... оператор_n;
}
while(условие);
```

Работает цикл следующим образом. В начале выполняется оператор, представляющий собой тело цикла. Затем вычисляется условие. Если оно истинно (не равно нулю), оператор тела цикла выполняется ещё раз. В противном случае цикл завершается, и управление передаётся оператору, следующему за циклом. Таким образом, не трудно заметить, что цикл с постусловием всегда будет выполнен хотя бы один раз, в отличие от цикла с предусловием, который может не выполниться ни разу.

Рассмотренный выше пример вывода на экран значений функции и выполненный с помощью этого цикла показан в листинге 1.9.

Листинг 1.9- Код программы с циклом do…while

```
#include <stdio.h>
#include <math.h>
using namespace std;
int main( )
{
float x, y;
x =6;        //Присваивание параметру цикла стартового значения
                        //Цикл с постусловием
//Пока параметр цикла превышает конечное значение
do{                       //выполнять тело цикла
y=sin(2*x) ;            //Вычислить значение y
                        //Форматный вывод на экран пары x и y
```

```
printf("\t x =5.2f \t y =5.4f \n",x,y) ;
x -=0.5;              //Изменение параметра цикла
                    // (переход к следующему значению x)
} while(x>=-4)                  //Конец цикла
return 0;
}
```

1.2 Функции в программах на C++

Как указывалось выше, программа языка C++ состоит из классов, в которые включаются функции, или как их еще называют- методы.

Вообще **функция** — это поименованный набор описаний и операторов, выполняющих определённую задачу. Функция может принимать параметры и возвращать значение. Информация, передаваемая в функцию для обработки, называется параметром, а результат вычисления функции - её значением. Обращение к функции называют «вызовом». Любая программа на C++ состоит из одной или нескольких функций. При запуске программы первой выполняется функция main- главная функция программы. Если среди операторов функции main встречается вызов функции, то управление передаётся операторам функции. Когда все операторы функции будут выполнены, управление возвращается оператору, следующему за вызовом функции.

Перед вызовом функция должна быть обязательно описана. Описание функции состоит из заголовка и тела функции:

```
тип  имя_функции(список_переменных)
{
тело_функции
}
```

Заголовок функции содержит:

тип возвращаемого функцией значения, он может быть любым, если функция не возвращает значения, указывают тип void;

имя функции;

список переменных — перечень передаваемых в функцию величин (аргументов), которые отделяются друг от друга запятыми; для каждой переменной из списка указывается тип и имя; если функция не имеет аргументов, то в скобках указывают либо тип void, либо ничего.

Тело функции представляет собой последовательность описаний и операторов, заключённых в фигурные скобки.

В общем виде структура программы на C++ может иметь вид:

```
директивы компилятора
тип  имя_1(список_переменных)
{
тело_функции_1;
}

тип  имя_2(список_переменных)
{
тело_функции_2;
}

...

тип  имя_n(список_переменных)
{
тело_функции_n;
}
i n t   main ( список_переменных )
{
//Тело  функции  может  содержать  операторы  вызова  функций имя_1,  имя_2,  ...,  имя_n тело_основной_функции;
}
```

Однако, допустима и другая форма записи программного кода:

```
директивы компилятора
тип  имя_1(список_переменных); тип  имя_2(список_переменных);
...
```

```
тип  имя_n(список_переменных);
int main(список_переменных )
{
//Тело функции может содержать операторы вызова функций имя_1, имя_2, ..., имя_n тело_основной_функции;
}
тип  имя_1(список_переменных)
{
тело_функции_1;
}

тип  имя_2(список_переменных)
{
тело_функции_2;
}

...
тип  имя_n(список_переменных)
{
тело_функции_n;
}
```

Здесь функции описаны после функции main(), однако до неё перечислены заголовки всех функций. Такого рода опережающие заголовки называют прототипами функций. Прототип указывает компилятору тип данных, возвращаемых функцией, тип переменных, выступающих в роли аргументов, и порядок их следования. Прототипы используются для проверки правильности вызова функций в основной программе. Имена переменных, указанные в прототипе функции, компилятор игнорирует:

```
//Записи равносильны.
int func (int a, int b);
int func (int , int );
```

Вызвать функцию можно в любом месте программы. Для вызова функции необходимо указать её имя и в круглых скобках, через запятую перечислить имена или значения аргументов, если таковые имеются:

```
имя_функции(список_переменных);
```

Рассмотрим пример. Создадим функцию f(), которая не имеет входных значений и не формирует результат. При вызове этой функции на экран выводится строка символов "С Новым Годом, " (листинг 1.10).

Листинг 1.10- Пример обращения к функции

```
#include <iostream >
using namespace std ;
void   f ( )    //Описание функции.
{
cout  <<  "С  Новым  Годом ,   " ;
}
int main( )
{
f ( ) ;
//Вызов функции.
cout   <<"Студент!"  << endl ;
f ( ) ;
//Вызов     функции.
cout   <<"Преподаватель!"  <<  endl ; f ( ) ;//Вызов функции.
cout   <<"Народ!"  << endl ;
}
```

Результатом работы программы будут три строки:

С Новым Годом, Студент!

С Новым Годом, Преподаватель! С Новым Годом, Народ!

Если тип возвращаемого значения не void, то функция может входить в состав выражений. Типы и порядок следования переменных в определении и при вызове функции должны совпадать. Для того чтобы функция вернула какое-либо значение, в ней должен быть оператор:

```
return (выражение);
```

Далее приведён пример программы, которая вычисляет значение выражения sin2(α) + cos2(α) при заданном значении α. Здесь функция radian выполняет перевод градусной меры угла в радианную (листинг 1.11).

Листинг 1.11- Пример функции, возвращающей значение

```
#include <iostream >
#include <math.h>
#define PI 3.14159
using namespace std ;
double radian (int deg, int min, int sec)
{
return (deg * PI/180+min* PI /180/60+ sec * PI / 180/60/60) ;
}
int main( )
{
int DEG, MIN, SEC; double RAD;
//Ввод данных.
cout<<"Inpout : "<<endl ; //Величина угла:
cout<<"DEG = " ;   cin >>DEG;  //градусы,
cout<<"MIN = " ;   cin >>MIN ; //минуты,
cout<<" SEC = " ;
cin >>SEC ;                                  //секунды.
//Величина угла в радианах.
RAD=radian(DEG, MIN, SEC) ; //Вызов функции.
cout << "V a l u e in r a d i a n A = "<<RAD << endl ;
//Вычисление значения выражения и его вывод.
cout << " sin ( A ) ^ 2 + c o s ( A ) ^ 2 = " ;
cout << pow ( sin (RAD) , 2 )+pow ( cos (RAD) , 2 ) << endl ;
return 0;
}
```

Переменные, описанные внутри функции, а также переменные из списка аргументов, являются локальными. Например, если программа содержит пять разных функций, в каждой из которых описана переменная N, то для C++ это пять различных переменных. Область действия локальной переменной не вы-

ходит за рамки функции. Значения локальных переменных между вызовами одной и той же функции не сохраняются.

Переменные, определённые до объявления всех функций и доступные всем функциям, называют глобальными. В функции глобальную переменную можно отличить, если не описана локальная переменная с теми же именем.

Глобальные переменные применяют для передачи данных между функциями, но это затрудняет отладку программы. Для обмена данными между функциями используют параметры функций и значения, возвращаемые функциями.

1.2.1 Передача параметров в функцию

Обмен информацией между вызываемой и вызывающей функциями осуществляется с помощью механизма передачи параметров. Список переменных, указанный в заголовке функции, называется формальными параметрами или просто параметрами функции. Список переменных в операторе вызова функции - это фактические параметры или аргументы.

Механизм передачи параметров обеспечивает замену формальных параметров фактическими параметрами и позволяет выполнять функцию с различными данными. Между фактическими параметрами в операторе вызова функции и формальными параметрами в заголовке функции устанавливается взаимно однозначное соответствие. Количество, типы и порядок следования формальных и фактических параметров должны совпадать.

Передача параметров выполняется следующим образом. Вычисляются выражения, стоящие на месте фактических параметров. В памяти выделяется место под формальные параметры в соответствии с их типами. Затем формальным параметрам присваиваются значения фактических. Выполняется проверка типов и при необходимости выполняется их преобразование.

Передача параметров в функцию может осуществляться по значению и по адресу.

При передаче данных по значению функция работает с копиями фактических параметров, и доступа к исходным значениям аргументов у неё нет. При

передаче данных по адресу в функцию передаётся не переменная, а её адрес, и, следовательно, функция имеет доступ к ячейкам памяти, в которых хранятся значения аргументов. Таким образом, данные, переданные по значению, функция изменить не может, в отличие от данных, переданных по адресу.

Если требуется запретить изменение параметра внутри функции, используют модификатор const. Заголовок функции в общем виде будет выглядеть так:

```
тип  имя_функции ( const  тип_переменной*  имя_переменной, ...)
```

Примеры различного вида передачи и использования переменных в функциях приведены в листинге 1.12.

Листинг 1.12 - Примеры различных функций

```
#include <iostream>
using namespace std ;
int  f1( int  i )  //Данные передаются по значению
{
return ( i ++) ;
}
int f2 ( int * j ) //Данные передаются по адресу. При подстановке фактического параметра,
//для получения его значения, применяется операция разадресации * .
{
return ( ( * j )++) ;
}
int  f3 ( const  int * k) //Изменение параметра не предусмотрено .
{
return ( ( * k )++) ;
}
int main ( )
{
```

```
    int a ;
    cout<<" a = " ; cin >>a ; f1(a) ;
    cout<<" a = "<<a<<" \ n " ;
    f2(&a)  ;  //Для  передачи  фактического  параметра  используется
операция взятия адреса & .
    cout<<" a = "<<a<<" \ n " ; f3(&a) ;
    cout<<" a = "<<a<<" \ n " ;
    return 0 ;}
```

Результат работы программы:

Введено значение переменной a.

a=5

Значение переменной a после вызова функции f1 не изменилось.

a=5

Значение переменной a после вызова функции f2 изменилось.

a=6

Значение переменной a после вызова функции f3 не изменилось.

a=6

Удобно использовать передачу данных по адресу, если нужно чтобы функция изменяла значения переменных в вызывающей программе.

1.2.2 Возврат результата с помощью оператора return

Возврат результата из функции в вызывающую её функцию осуществляется оператором

```
return выражение;
```

Работает оператор следующим образом. Вычисляется значение выражения, указанного после return, и преобразуется к типу возвращаемого функцией значения. Выполнение функции завершается, а вычисленное значение передаётся в вызывающую функцию. Любые операторы, следующие в функции за оператором return, игнорируются. Программа продолжает свою работу с оператора, следующего за оператором вызова данной функции.

Оператор return может отсутствовать в функциях типа void, если возврат происходит перед закрывающейся фигурной скобкой, и в функции main.

Также функция может содержать несколько операторов return, если это определено потребностями алгоритма. Например, в следующей программе функция equation вычисляет корни квадратного уравнения. Если a = 0 (уравнение не является квадратным), то в программу передаётся значение равное 1, если дискриминант отрицательный (уравнение не имеет действительных корней), то 1, а если положительный, то вычисляются корни уравнения и в программу передаётся 0 (листинг 1.13).

Листинг 1.13 – Пример функции, возвращающей значения

```
#include <iostream >
#include <math.h>
using namespace s t d ;
int equation (float a, float b, float c , float * x1 , float *
x2 )
{ float D=b*b-4*a * c ;
if ( a==0) return -1;
else  if (D<0) return 1 ;
else
{
* x1=(-b+s q r t (D) ) /2/ a ;
* x2=(-b-s q r t (D) ) /2/ a ;
return 0 ;
}
}

int main( )
{
float A, B, C, X1 , X2 ; int P ;
cout<<" Enter the coefficient sof the equation : "<<e n d l ;
cout<<" A = " ; cin >>A;
cout<<" B = " ; c n >>B ; cout<<" C = " ;   cin >>C ;
```

```
P=equation(A, B, C, &X1 , &X2 ) ;
if (P==-1) cout<<" input Error "<<endl ;
else if   (P==1)     cout<<" N o r e a l r o o t s "<<endl ;
else  cout<<" X 1 = "<<X1<<" X 2 = "<<X2<<endl ;
return 0 ;
}
```

1.2.3 Специальное использование функций

1.2.3.1 Рекурсивные функции

Под рекурсией в программировании понимают функцию, которая вызывает сама себя. Рекурсивные функции чаще всего используют для компактной реализации рекурсивных алгоритмов. Классическими рекурсивными алгоритмами могут быть возведение числа в целую положительную степень, вычисление факториала. С другой стороны, любой рекурсивный алгоритм можно реализовать без применения рекурсий. Достоинством рекурсии является компактная запись, а недостатком — расход памяти на повторные вызовы функций и передачу параметров, существует опасность переполнения памяти.

Вычислить n-ю степень числа a (n — целое число) с помощью рекурсивной функции.

Результатом возведения числа a в целую степень n является умножение этого числа на себя n раз. Но это утверждение верно только для положительных значений n. Если n принимает отрицательные значения, то $a^{-n} = 1/a$.

В случае n = 0, $a^0 = 1$.

Для решения задачи создадим рекурсивную функцию stepen, алгоритм которой представлен на рисунке 1.5.

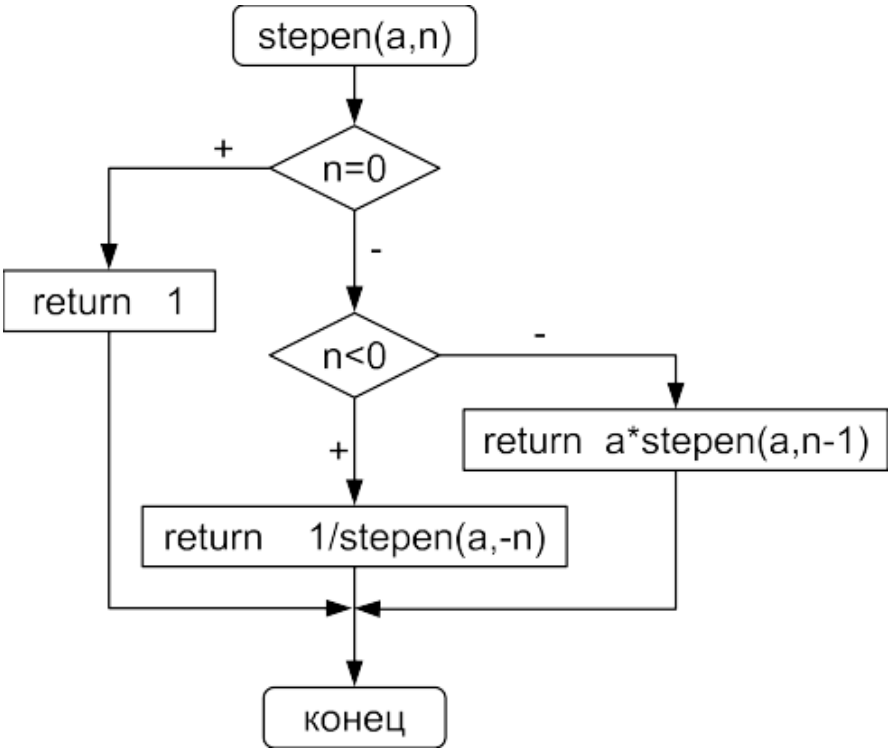


Рисунок 1.5- Алгоритм возведения в степень с помощью рекурсии

Текст программы с применением рекурсии показан в листинге 1.14.

Листинг 1.14- Текст программы рекурсивной функции

```
#include <iostream>
using namespace s t d ;
float stepen (float a, int n)
{
if ( n==0)
return 1 ;
else if ( n<0)
return 1/ stepen ( a ,-n ) ;
else
return a * stepen( a , n1) ;
}
int main( )
{
int i ; float s , b ; cout<<" b = " ; cin >>b ;
```

```
cout<<" i = " ; ci n >>i ; s=stepen(b , i) ;
cout<<" s = "<<s<<" \ n " ;
return 0 ;
}
```

1.2.3.2 Перегрузка функций

Язык C++ позволяет связать с одним и тем же именем функции различные определения, то есть возможно существование нескольких функций с одинаковым именем. У этих функций может быть разное количество параметров или разные типы параметров. Создание двух или более функций с одним и тем же именем называется перегрузкой имени функции. Перегруженные функции создают, когда одно и то же действие следует выполнить над разными типами входных данных.

В приведённом далее тексте программы три функции с именем Pow. Первая выполняет операцию возведения вещественного числа a в дробную степень n = k/m , где k и m — целые числа. Вторая возводит вещественное число a в целую степень n, а третья — целое число a в целую степень n. Какую именно функцию вызвать компилятор определяет по типу фактических параметров. Так, если a — вещественное число, а k — целое, то оператор Pow(a,k) вызовет вторую функцию, так как она имеет заголовок float Pow(float a, int n). Команда Pow((int)a,k) приведёт к вызову третьей функции float Pow(int a, int n), так как вещественная переменная a преобразована к целому типу. Первая функция float Pow(float a, int k, int m) имеет три параметра, значит, обращение к ней осуществляется командой Pow(a,k,m) (листинг 1.15).

Листинг 1.15- Пример перегрузки функций

```
#include <iostream >
using namespace std ;
#include <math . h>
float Pow( float a , int k , int m) //Первая функция
{
```

```
cout<<"Функция 1 \ t " ;
if ( a==0)
return  0 ;
else if ( k==0)
return 1 ;
else if ( a >0)
return exp ( ( f l o a t ) k/m* l o g ( a ) ) ;
else if (m%2!=0)
return (exp ( ( f l o a t ) k/m* l o g (-a ) ) ) ;

float  Pow( float  a ,  int  n )  //Вторая функция
{
float p ; int i ;
cout<<"Функция 2 \ t " ;
if ( a==0)
return 0 ;
else if ( n==0)
return 1 ;
else if ( n<0)
{
n= n ; p=1;
for ( i =1; i <=n ; i ++) p*=a ;
return ( f l o a t ) 1/ p ;
}
else
{
p=1;
f o r ( i =1; i <=n ; i ++) p*=a ;
return p ;
}
}
float Pow( int a , int n ) //Третья функция
{
int i , p ;
cout<<"Функция 3 \ t " ;
```

```
    if ( a==0)
    return 0 ;
    else if ( n==0)
    return 1 ;
    else if ( n<0)
    {
    n= n ; p=1;
    for ( i =1; i <=n ; i ++) p*=a ;
    return ( float ) 1/ p ;
    }
    else
    {
    p=1;
    for ( i =1; i <=n ; i ++) p*=a ;
    return p ;
    }
    }
    int main( )
    {
    float a ; int k , n ,m; cout<<" a = " ; cin >>a ; cout<<" k =
" ;   cin >>k ;
    cout<<" s = "<<Pow( a , k )<<" \ n " ;  //Вызов 2-й функции.
    cout<<" s = "<<Pow ( ( i n t ) a , k )<<" \ n " ; //Вызов 3-й
функции.
    cout<<" a = " ; cin >>a ;
    cout<<" k = " ; cin >>k ; cout<<" m = " ; c i n >>m;
    cout<<" s = "<<Pow( a , k ,m)<<e n d l ;//Вызов 1-й функции.
    return 0 ;
    }
```

Результаты работы программы:

a=5.2

k=3

Функция 2 s=140.608

Функция 3 s=125 a=-8

k=1 m=1

Функция 1 s=-8

a=5.2 k=-3

Функция 2 s=0.00711197

Функция 3 s=0.008 a=-8

k=1 m=3

Функция 1 s=-2

1.2.3.3 Шаблоны функций

Шаблон — это особый вид функции. С помощью шаблона функции можно определить алгоритм, который будет применяться к данным различных типов. Механизм работы шаблона заключается в том, что на этапе компиляции конкретный тип данных передаётся в функцию в виде параметра.

Простейшую функцию–шаблон в общем виде можно записать так:

template <class Type> заголовок

{

тело функции

}

Обычно в угловых скобках указывают список используемых в функции типов данных. Каждый тип предваряется служебным словом class. В общем случае в списке могут быть не только типы данных, но и имена переменных.

Рассмотрим пример шаблона поиска наименьшего из четырёх чисел (листинг 1.16).

Листинг 1.16- Пример шаблона функций

```
#include <iostream>
using namespace std ;
```

```
    //Определяем абстрактный тип данных с помощью служебного слова
Type
    template <class Type>
    Type minimum ( Type a , Type b , Type c , Type d )
    { //Определяем функцию с использованием типа данных Type .
    Type   min=a ;
    if  ( b<min )  min=b ; i f  ( c<min )   min=c ; i f ( d<min )
min=d ; return min ;
    }
    int main ( )
    {
    int ia , ib , ic , id , mini ; float ra , rb , rc , rd , minr
;
     cout<<" Vvod 4 thelih chisla \ t " ;
    cin >>ia >>ib >>ic >>id ;
    mini=minimum ( ia , ib , ic , id ) ; //Вызов функции minimum ,
в которую передаём 4  целых значения.
    cout<<" \ n "<<mini<<" \ n " ;
    cout<<" Vvod 4 veshestvenih chisla \ t " ; cin >>ra>>rb>>rc
>>rd ;
    minr=minimum ( ra , rb , r c , r d ) ; //Вызов функции minimum
, в которую передаём 4  вещественных значения.
    cout<<" \ n "<<minr<<" \ n " ;
    return 0 ;
    }
```

1.3 Объектно-ориентированное программирование в программах на C++

Объектный подход родился как следующий важный шаг на пути качественного написания больших программ. В нём предлагается разделять программу на самостоятельные части — объекты, наделённые собственными свойствами, текущим состоянием, и умеющие взаимодействовать друг с другом и с окружающей средой — примерно так, как это происходит у объектов реального мира.

В упрощённом виде такая парадигма получила название объектно- ориентированного программирования (ООП) — подхода, который позволяет использовать в программе объекты и даже поощряет эту практику, но не требует, чтобы программа состояла из одних только объектов.

Объект в самом общем смысле — это осязаемая сущность, которая чётко проявляет своё поведение.

Объект состоит из следующих трёх частей:

- имя объекта;
- состояние (значение его переменных);
- методы (функции).

Однотипные объекты образуют **класс**. Под однотипными объектами мы понимаем такие объекты, у которых одинаковы наборы методов и переменных состояния.

1.3.1 Создание классов и объектов в C++

Синтаксис описания класса во многом копирует синтаксис описания структуры. В простейшем случае класс описывается так:

```
class   имя_класса
{
закрытые члены класса
public :
открытые члены класса
} ;
```

Как и при объявлении структуры, имя класса становится новым именем типа данных, которое можно использовать для объявления переменных (объектов класса). Членами класса будут переменные и функции, объявленные внутри класса. Функции-члены класса называют методами этого класса, а переменные- члены класса называют свойствами класса.

В C++ понятия ООП используются следующим образом [3, 4]:

- «класс»: пользовательский тип данных, во многом аналогичный структуре;
- «объект класса» или «переменная-экземпляр класса»: переменная, в описании которой какой-то класс указан в качестве типа данных;
- «свойство» или «переменная-член класса»: переменная, объявленная внутри класса (как поле внутри структуры); на практике чаще говорят не о свойстве класса, а о свойстве объекта, так как для конкретных объектов переменные — члены класса обладают конкретными значениями и потому имеют конкретный смысл.
- «метод »: функция, объявленная внутри класса.

По умолчанию все функции и переменные, объявленные в классе, являются закрытыми, т. е. принадлежат закрытой секции класса. Это значит, что они доступны для обращения только изнутри членов этого класса и недоступны извне. Для объявления открытых членов класса используется ключевое слово public с двоеточием, обозначающее начало открытой секции класса. Все члены класса, объявленные после слова public, доступны для обращения как изнутри этого же класса, и для любой другой части программы, в которой доступен класс.

Открытых и закрытых секций в классе может быть несколько, и они могут произвольно чередоваться. При необходимости обозначить очередную закрытую секцию, её начало обозначается ключевым словом private.

Рассмотрим в качестве примера объект, представляющий собой геометрический вектор в трехмерном пространстве. Для простоты ограничимся хранением в объекте трёх координат и функции, вычисляющей модуль вектора (листинг 1.17).

Листинг 1.17- Пример записи класса

```
classs  patial_vector
{
public :
```

```
double abs ( ) ;
private :
double x , y , z ;
      } ;
```

Добавив в структуру или в класс какой-нибудь метод, можно вызвать его для конкретного объекта. Обращение к содержимому объекта выполняется с использованием операции «.» (либо операции «->», если нужно обратиться по указателю на объект). Например, это можно сделать в главной функции программы (листинг 1.18).

Листинг 1.18- Обращение к методу класса из функции main()

```
main ( )
{
spatial _vector  a , b ;
double d ;
. . . . . .
d = a.abs() ;
      }
```

Очевидно, что функция abs(), объявленная в классе spatial_vector, возвращает абсолютное значение вектора. Однако для того, чтобы программа скомпилировалась, после объявления функцию abs() нужно ещё определить (т. е. написать тело этой функции). Определение метода выполняется так же, как обычной функции, только в имени метода нужно указать, что он принадлежит конкретному классу. Для этого используется оператор расширения области видимости «::». Имя класса записывается перед именем функции, отделённое двойным двоеточием. Например, в следующем примере мы объявим всё тот же класс spatial_vector с двумя методами (установить значения координат вектора и посчитать его модуль) и опишем эти методы в листинге 1.19.

Листинг 1.19- Пример программы с методом класса

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

```
using  namespace   std ;
class  spatial_vector
{
double x , y , z ;
public :
void set(double a , double b , double c) ;
double abs() ;
} ;
void spatial_vector :: set(double a , double b , double c)
{
x=a ; y=b ; z=c ;
}
double spatial_vector :: abs()
{
return sqrt (x*x + y*y + z*z) ;
}
main ( )
{
spatial_vector a;
a.set (1,2,3);
cout << a.abs()<< endl ;
}
```

1.3.2 Наследование. Полиморфизм. Инкапсуляция

Понятие «инкапсуляции» было рассмотрено выше, когда указывалось, что в классе могут быть открытые и закрытые разделы. Именно закрытый раздел, начинающийся оператором «**private:**» и определяет какие переменные и функции будут недоступны к просмотру из других объектов программы. Т.е., инкапсуляция – это закрытие объектов класса от внешних, по отношению к нему объектов.

Наследование классов позволяет строить иерархию, наверху которой находятся более общие классы, а внизу - более специализированные. Попробуем

привести наглядный пример иерархии наследования. Предположим, мы создаём объектно-ориентированную систему работы с графикой, и предусмотрели класс point, описывающий отдельную двумерную точку на экране. В этом классе хранятся координаты точки, её цвет, а также методы для управления этими данными. При необходимости можно легко создать на базе класса point производный класс, хранящий трехмерную вершину (например, vertex): добавить в нём третью координату, соответствующие конструкторы, модифицировать некоторые методы. Однако не следует путать отношение наследования с отношением включения. Например, будет нелогичным строить на базе класса point или класса vertex класс region, описывающий объекты с произвольным количеством вершин: скорее, это должен быть класс-контейнер, содержащий в себе массив объектов point или vertex.

Таким образом, есть смысл создавать на базе существующего класса производный, если мы хотим получить частный случай с модифицированной функциональностью.

В C++ новый класс строится на базе уже существующего с помощью конструкции следующего вида:

```
class parent { . . . . . } ;
class child : [ модификатор наследования ] parent { . . . . . } ;
```

При определении класса-потомка, за его именем следует разделитель-двоеточие «:», затем необязательный модификатор наследования и имя родительского класса. Модификатор наследования определяет видимость наследуемых переменных и методов для класса-потомка и его возможных потомков. Таким способом определяется, какие права доступа к переменным и методам класса-родителя будут «делегированы» классу-потомку.

При реализации наследования область видимости принадлежащих классу данных и методов можно определять выбором одного из следующих модификаторов доступа:

– private (закрытый);

- public (общедоступный);
- protected (защищённый).

Эти модификаторы могут произвольно чередоваться внутри описания класса и уже использовались нами для обозначения открытых и закрытых секций класса. Модификатор private описывает закрытые члены класса, доступ к которым имеют только методы-члены этого класса. Модификатор public предназначен для описания общедоступных элементов, доступ к которым возможен из любого места в программе, где доступен объект данного класса. Модификатор protected используется тогда, когда необходимо, чтобы некоторые члены базового класса оставались закрытыми, но были бы доступны из класса-потомка.

Иными словами, одни и те же ключевые слова могут использоваться и в качестве модификаторов наследования, и в качестве модификаторов доступа.

То, как изменяется доступ к элементам базового класса из методов производного класса в зависимости от значения модификаторов наследования, показано в таблице 1.2.

Таблица 1.2- Сочетание модификаторов доступа и наследования

Модификатор доступа	Модификатор наследования		
	public	protected	private
public	public	protected	private
protected	protected	protected	private
private	нет доступа	нет доступа	нет доступа

Из таблицы видно, в производном классе доступ к элементам базового класса может быть сделан более ограниченным, но никогда нельзя сделать его менее ограниченным.

В списке базовых классов можно указывать несколько классов-родителей, через запятую, каждого со своим модификатором наследования и тем самым получить множественное наследование:

```
class  A  { . . . } ;
class  B  { . . . } ;
```

```
class C : public A, public B { . . . } ;
```

При этом класс C унаследует как содержимое класса A, так и класса B. При вызове конструктора будут сначала вызваны конструкторы базовых классов (в порядке следования). Деструкторы, как всегда, имеют противоположный порядок вызова.

При множественном наследовании автоматически включается позднее связывание.

Как уже упоминалось, перегрузка или **полиморфизм**, т. е. возможность создавать функции (например, методы класса) с одинаковыми именами и разными наборами параметров, вызываемые в разных ситуациях для решения однотипных задач — это одно из ключевых проявлений полиморфизма в C++. Однако кроме перегрузки функций C++ позволяет проделывать то же самое с большинством стандартных операторов.

На самом деле, можно считать, что перегрузка операторов для стандартных типов данных в неявном виде присутствовала ещё в языке C. Например, оператор деления может выполнять разные действия в зависимости от того, какой тип имеют его аргументы: для целочисленных аргументов будет выполнено деление нацело, а для вещественных — деление чисел с плавающей точкой. С точки зрения процессора деление чисел с плавающей точкой кардинально отличается от деления нацело: задействована другая машинная команда, операнды должны быть загружены в совсем другие регистры (ячейки памяти процессора), после чего выполняется совсем другая микропрограмма. На более высоком уровне абстракции операции целочисленного и вещественного деления могут казаться одинаковыми; однако использование для них одного и того же оператора допускают далеко не все языки.

В C++ это явление довели до логического завершения, и теперь многие встроенные операторы можно перегрузить для работы с новыми типами данных. Что- бы перегрузить оператор, программист объявляет новую функцию, имя которой состоит из ключевого слова operator и знака операции. Например,

перегрузим оператор + для сложения двух объектов класса spatial_vector. Объявление функции будет выглядеть следующим образом (листинг 1.20).

Листинг 1.20- Пример полиморфизма

spatial_vector **operator**+ (spatial_vectora , spatial _ vector b)

{. . . . . .}

Нам понадобится предусмотреть в классе spatial_vector геттеры (возвращающие методы) и сеттеры (задающие методы) для всех трёх координат, чтобы внешняя функция могла выполнить покоординатное сложение двух векторов (либо мы могли бы объявить функцию дружественной классу). Также мы предусмотрим в классе конструктор, инициализирующий координаты заданными значениями, и метод info, выводящий координаты вектора на экран (листинг 1.21).

Листинг 1.21- Пример программы с полиморфизмом

```
#include <iostream>
#include <math.h>
using  namespace   std ;
class  spatial_vector
{
double x , y , z ;
public :
spatial_vector(double x , double y , double z)
{ this ->x=x ; this ->y=y ; this ->z=z ; }
double   get_x ( )    {    return   x ;    }
double   get_y ( )    {    return   y ;    }
double   get_z ( )    {    return   z ;    }
void   set_x ( double   x )    {    this ->x=x ;    } void
set_y ( double   y )    {    this ->y=y ;    } void s e t_ z (
double z ) { this ->z=z ; }
void info(){cout << "Координаты  вектора: "<<x<<" , "<<y<<" ,
"<<z<<endl;}
```

```
    } ;
    spatial_vector operator+ (spatial_vector a , spatial_vector b)
    {
    spatial_vector c ( 0 , 0 , 0 ) ;
    c . set_x ( a . get_x ( )   +   b . get_x ( ) ) ;
    c . set_y ( a . get_y ( )   +   b . get_y ( ) ) ;
    c . s e t_ z ( a . get_z ( )   +   b . get_z ( ) ) ;
    return c ;
    }
    main ( )
    {
    spatial_vector a ( 1 , 2 , 3 ) , b ( 1 0 , 2 0 , 3 0 ) , c ( 0
, 0 , 0 ) ; c=a+b ;
    c . info( ) ;
    }
```

1.3.3 Использование шаблонов классов

Шаблоны классов во многом похожи на шаблоны функций, рассмотренные ранее. Точно так же, как и в случае шаблонов функций, шаблоны классов позволяют отделить общий алгоритм от его реализации под конкретные типы данных. Классический пример ситуации, когда выгодно применять шаблоны классов это так называемые контейнеры, т. е. классы, содержащие наборы некоторых значений (динамические списки, массивы, множества). Закладывая тип данных элемента как параметр шаблона, можно создать универсальный класс-контейнер, а на его основе порождать объекты для хранения наборов элементов конкретного типа — контейнер целых чисел, контейнер строк и т. д.

Описание шаблона класса также имеет много общего с шаблоном функции. Описание класса точно так же начинают с ключевого слова template, за которым следует список формальных параметров шаблона в угловых скобках. Этот же заголовок повторяется и перед описанием методов класса. В качестве параметров шаблона можно передавать типы данных или константы, но перед

идентификатором, обозначающим тип данных, в списке формальных параметров ставится ключевое слово class.

В отличие от шаблонов функций, для которых фактические параметры шаблона (т. е. конкретные типы данных) определяются по типам аргументов, переданных функции, для шаблонов классов фактические параметры необходимо передавать явно. Список фактических параметров шаблона указывается после имени класса в угловых скобках во всех случаях, когда имя шаблонного класса используется в программе — например, при порождении объектов, или при указании принадлежности элемента-члена класса. Дальнейшее обращение с порождёнными объектами не отличается от обычных классов.

Рассмотрим в качестве простого примера класс point, который хранит пару координат точки и имеет метод info(), выводящий координаты на экран (листинг 1.22).

Листинг 1.22- Пример шаблона класса

```
#include<iostream >
using namespace s t d ;
template <class Type>
class point
{
Type x , y ;
// . . .
public :
point (Type x , Type y) { this −>x=x ; this −>y=y ; }
void info ( ) ;
} ;
template <class Type>
void p o i n t <Type > : : i n f o ( )
{
cout << "Координаты точки: x = " << x << " , y = " << y << endl ;
```

```
}
main ( )
{
point <float> f ( 1 0 . 1 ,  2 0 . 5 ) ; f . info ( ) ;
}
```

Как видно, конкретный тип (float) указан при создании объекта в угловых скобках. Точно так же можно указать любой стандартный тип данных, или пользовательский тип, объявленный в программе. Однако необходимо помнить, что шаблоны функций и шаблоны классов могут работать только для тех типов данных (в т.ч. классов), которые поддерживают необходимые операции.

Шаблоны классов, как и классы, поддерживают механизм наследования. Все принципы наследования при этом остаются неизменными, что позволяет построить иерархическую структуру шаблонов, аналогичную иерархии классов.

1.4 Разработка интерфейсных приложений в C++

Создадим новое оконное приложение C++ в MS Visual Studio, используя Windows Forms Application, что является проще и удобней.

Необходимо указать название приложения и выбрать указанный способ создания (рисунок 1.6).

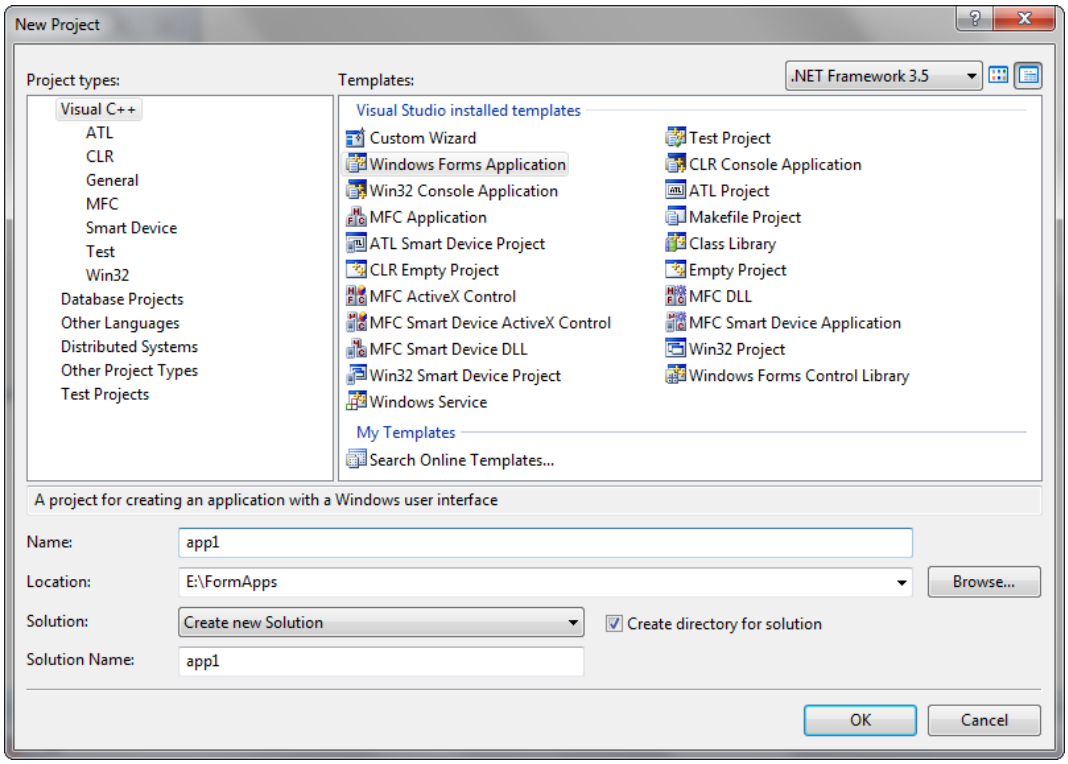


Рисунок 1.6- Начало создания оконного приложения

При этом во вкладке «Решение» («Solution») появится файл формы Form1 (рисунок 1.7).

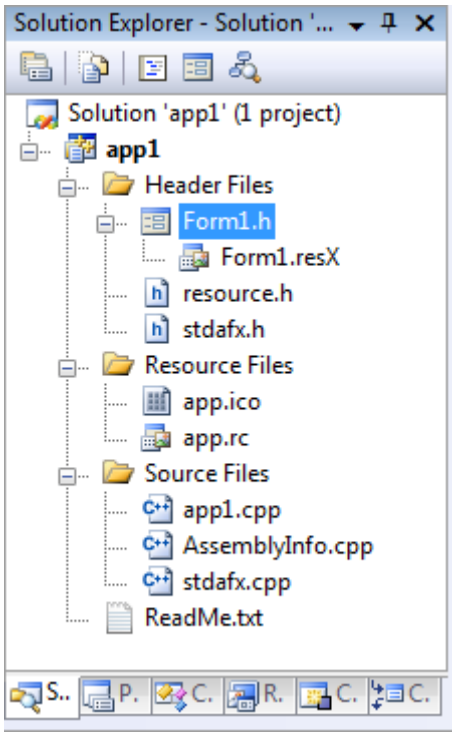


Рисунок 1.7- Вкладка «Решение»

Добавим на форму, для примера, стандартный элемент управления – текстовое поле, выбрав его из списка инструментов - Toolbox (рисунок 1.8).

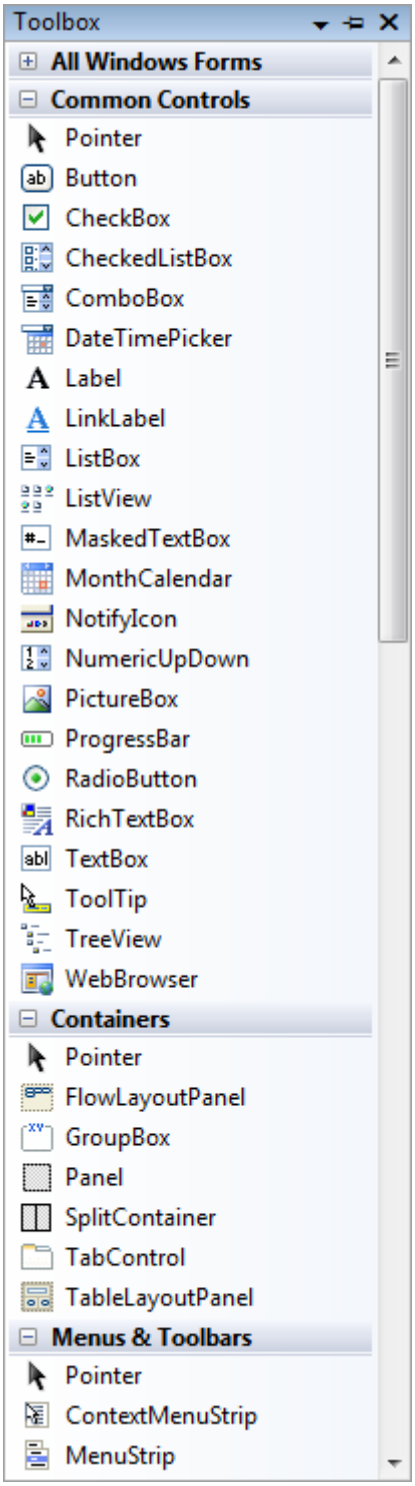


Рисунок 1.8- Список инструментов, располагаемых на форме

Далее, откроем и исправим форму, как показано на рисунке 1.9.

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) {
   textBox1->Text = "Hello, World!"; //textBox1 – имя добавленного вами текст бокса
}
```

Рисунок 1.9- Редактирование формы

После запуска приложения на экране появляется следующая форма (рисунок 1.10).

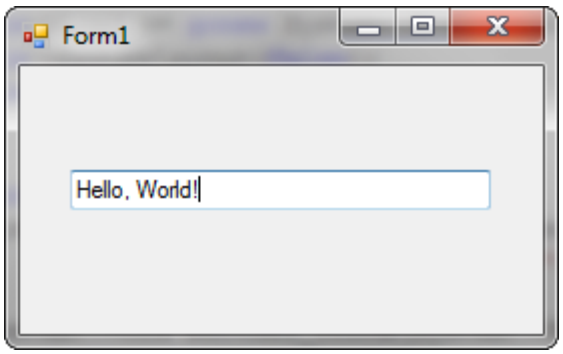


Рисунок 1.10- Отображение формы на экране

Аналогично создаются оконные приложения в C++ любой сложности.

2 Методические указания по выполнению заданий практических занятий

Далее описаны задания, которые необходимо выполнить на практических занятиях и даны методические указания для их выполнения.

2.1 ПЗ-1. Создание приложений с различными видами наследования

1) Цели занятия

Практически освоить основные варианты наследования в задачах на C++

2) Рекомендации и порядок выполнения заданий

Необходимо изучить материалы лекций по этой теме и пункты 1.1 и 1.3 настоящего пособия.

Порядок выполнения задания:

a) Составить программу решения 5 вычислительных задач по своему варианту. Определение номера 1-го задания:

- нечетные по журналу: 1-е задание – это номер по журналу;

- четные по журналу: 1-е задание – это № последнего задания- № по журналу. Если достигнут конец списка вариантов заданий, то перейти к первому варианту и далее.

b) Решить указанные задания и составить блок-схемы алгоритмов решения в электронном виде.

c) Составить отчет, включив в него задание и его выполнение. Представить отчет в электронном виде на проверку.

3) Варианты заданий

1. Даны три целых числа. Возвести в квадрат отрицательные числа и в третью степень — положительные (число 0 не изменять).

2. Из трех данных чисел выбрать наименьшее.
3. Из трех данных чисел выбрать наибольшее.
4. Из трех данных чисел выбрать наименьшее и наибольшее.
5. Перераспределить значения переменных X и Y так, чтобы в X оказалось меньшее из этих значений, а в Y — большее.
6. Даны две переменные целого типа: A и B. Если их значения не равны, то присвоить каждой переменной сумму этих значений, а если равны, то присвоить переменным нулевые значения.
7. Даны две переменные целого типа: A и B. Если их значения не равны, то присвоить каждой переменной максимальное из этих значений, а если равны, то присвоить переменным нулевые значения.
8. Даны три переменные: X, Y, Z. Если их значения упорядочены по убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное.
9. Даны три переменные: X, Y, Z. Если их значения упорядочены по возрастанию или убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное.
10. Даны целочисленные координаты точки на плоскости. Если точка не лежит на координатных осях, то вывести 0. Если точка совпадает с началом координат, то вывести 1. Если точка не совпадает с началом координат, но лежит на оси OX или OY, то вывести соответственно 2 или 3.
11. Даны вещественные координаты точки, не лежащей на координатных осях OX и OY. Вывести номер координатной четверти, в которой находится данная точка.
12. На числовой оси расположены три точки: A, B, C. Определить, какая из двух последних точек (B или C) расположена ближе к A, и вывести эту точку и ее расстояние от точки A.
13. Даны четыре целых числа, одно из которых отлично от трех других, равных между собой. Вывести порядковый номер этого числа.

14. Дан номер некоторого года (положительное целое число). Вывести соответствующий ему номер столетия, учитывая, что, к примеру, началом 20 столетия был 1901 год.
15. Дан номер некоторого года (положительное целое число). Вывести число дней в этом году, учитывая, что обычный год насчитывает 365 дней, а високосный — 366 дней. Високосным считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300 и 1900 не являются високосными, а 1200 и 2000 — являются).
16. Дано целое число, лежащее в диапазоне от –999 до 999. Вывести строку — словесное описание данного числа вида "отрицательное двузначное число", "нулевое число", "положительное однозначное число" и т.д.
17. Дано целое число, лежащее в диапазоне от 1 до 9999. Вывести строку — словесное описание данного числа вида "четное двузначное число", "нечетное четырехзначное число" и т.д.
18. Два прямоугольника заданы длинами сторон. Определите, можно ли первый прямоугольник целиком разместить во втором.
19. Решите квадратное уравнение $ax^2 + bx + c = 0$
20. Составить программу, печатающую значение true, если указанное высказывание является истинным, и false в противном случае: данная тройка натуральных чисел a, b, c не является тройкой Пифагора, т.е. $c^2 = a^2 + b^2$
21. Составить программу, печатающую значение true, если указанное высказывание является истинным, и false в противном случае: данные числа x, y являются координатами точки, лежащей в первой координатной четверти.
22. Составить программу, печатающую значение true, если указанное высказывание является истинным, и false в противном случае: данная тройка натуральных чисел a, b, c является тройкой Пифагора, т.е. $c^2 = a^2 + b^2$.

23. Составить программу, печатающую значение true, если указанное высказывание является истинным, и false в противном случае: число c является средним арифметическим чисел a и b.
24. Составить программу, печатающую значение true, если указанное высказывание является истинным, и false в противном случае: число c является средним геометрическим чисел a и b.
25. Составить программу, печатающую значение true, если указанное высказывание является истинным, и false в противном случае: сумма двух натуральных чисел кратна 2.
26. Составить программу, печатающую значение true, если указанное высказывание является истинным, и false в противном случае: произведение натуральных чисел a и b кратно числу c.
27. Составить программу, печатающую значение true, если указанное высказывание является истинным, и false в противном случае: данное натуральное число a кратно числу b, но не кратно числу c.
28. Составить программу, печатающую значение true, если указанное высказывание является истинным, и false в противном случае: числа a и b выражают длины катетов одного прямоугольного треугольника, c и d — другого. Эти треугольники являются подобными.

2.2 ПЗ-2. Составление приложений с инкапсуляцией и полиморфизмом методов классов

1) Цели занятия

Практически освоить основные способы инкапсуляции и полиморфизма методов в C++.

2) Рекомендации и порядок выполнения заданий

Необходимо изучить материалы лекций по этой теме и пункты 1.1 и 1.3 настоящего пособия.

Порядок выполнения задания:

a) Составить программу решения 5 вычислительных задач по своему варианту. Определение номера 1-го задания:

- нечетные по журналу: 1-е задание – это номер по журналу;

- четные по журналу: 1-е задание – это № последнего задания- № по журналу. Если достигнут конец списка вариантов заданий, то перейти к первому варианту и далее.

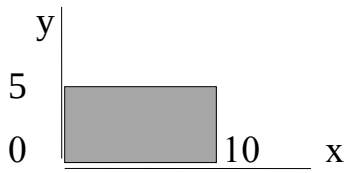
b) Решить указанные задания и составить блок-схемы алгоритмов решения в электронном виде.

c) Составить отчет, включив в него задание и его выполнение. Представить отчет в электронном виде на проверку.

3) Варианты заданий

Решить задачи, инкапсуляруя 1-й метод класса и показать полиморфизм остальных методов.

1. Даны действительные числа A,B,C,D. Выяснить, можно ли уместить прямоугольник со сторонами A,B внутри прямоугольника со сторонами C,D.
2. Даны действительные числа x, y, z. Найти минимальное из них.
3. Даны действительные положительные числа A,B,C. Выяснить, пройдет ли кирпич с ребрами A,B,C в прямоугольное отверстие со сторонами x,y.
4. Определить, лежит ли точка D(c,b), где $c=\sqrt{a1+a2}$, $b=a1+0{,}7a3$, внутри прямоугольника (a1,a2,a3 - произвольные числа)



5. Выяснить, существует ли треугольник с координатами вершин A(x1,y1), B(x2,y2), C(x3,y3), если да, то найти его площадь.
6. Даны действительные числа A,B,C. Проверить выполняются ли неравенства A<B<C, если да, то присвоить A=B+C иначе A=C-B.
7. Даны действительные числа x,y. Вычислить значение функции z=log(x-y)-x/y

8. На плоскости расположена окружность радиуса R с центром в начале координат

Определить положение точки x с координатами (A,B) относительно окружности.

9. Даны круг радиуса R и квадрат со стороной A. Определить их взаимное положение.
10. Вывести на печать переменные A,B,C в порядке их возрастания
11. Проверить, какие из чисел A,B,C,D принадлежат интервалу (1,25).
12. Даны действительные числа A,B. Если они оба отрицательные, то заменить каждое из них его квадратом, иначе положительные из них увеличить в два раза.
13. Выяснить, существует ли треугольник с координатами вершин A(x1,y1), B(x2,y2),C(x3,y3).
14. Даны действительные числа x,y. Вычислить значение функции z=log(x/y)-1/x.
15. Даны действительные числа A,B. Если они оба неотрицательные, то заменить каждое из них его кубом, иначе отрицательные из них заменить их модулями.
16. Даны площадь квадрата S1 и круга S2. Определить поместится ли круг в квадрат и наоборот.
17. На плоскости расположена окружность радиуса R с центром в начале координат. Определить, лежат ли точки A(x1,y1) и B(x2,y2) на окружности.
18. Составить программу вычисления корней системы уравнений с двумя неизвестными

$$\begin{cases} a1x + b1y = c1 \\ a2x + b2y = c2 \end{cases}$$ методом Крамера. Проверить, что главный определитель не равен 0.

19. Даны действительные числа A,B,C,D. Выяснить, можно ли уместить прямоугольник со сторонами A,B внутри прямоугольника со сторонами C,D.
20. Вывести на печать переменные A,B,C в порядке их убывания

21. Даны действительные числа x, y, z. Найти максимальное из них.
22. Проверить, какие из чисел A,B,C,D не принадлежат интервалу (3,15).
23. Даны действительные числа x,y. Вычислить значение функции z=ln(x)-x/y
24. Даны действительные числа A,B. Если они имеют разные знаки, то напечатать их произведение, иначе напечатать их квадраты
25. Выяснить, существует ли треугольник с длинами сторон A,B,C. Если да, то найти его площадь.
26. Даны действительные числа x,y. Вычислить значение функции z=arcsin(x)-y
27. Даны действительные числа x,y,z. Получить максимальное из них по модулю
28. Даны действительные числа x,y. Вычислить значение функции z=arcsin(x+y)
29. На каком из интервалов $(-\infty;k1),(k1;k2),(k2;+\infty)$ лежит точка с координатой x? k1, k2, x - произвольные числа, причем k1<k2.
30. Лежат ли обе точки D(a1;b1) и C(a2;b2) внутри круга радиуса R с центром в начале координат? Если такой точки нет, выдать соответствующее сообщение.

2.3 ПЗ-3. Создание консольных приложений в Visual Studio (Qt)

1) Цели занятия

Выработать умения и навыки по составлению консольных проектов в Visual Studio.

2) Рекомендации и порядок выполнения заданий

Необходимо изучить материалы лекций по этой теме и пункты 1.1 и 1.3 настоящего пособия.

Порядок выполнения задания:

а) Составить программу решения 5 вычислительных задач по своему варианту. Определение номера 1-го задания:

- нечетные по журналу: 1-е задание – это номер по журналу;

- четные по журналу: 1-е задание – это № последнего задания- № по журналу. Если достигнут конец списка вариантов заданий, то перейти к первому варианту и далее.

b) Решить указанные задания и составить блок-схемы алгоритмов решения в электронном виде.

c) Составить отчет, включив в него задание и его выполнение. Представить отчет в электронном виде на проверку.

3) Варианты заданий

Составить функции (методы) для решения следующих задач.

1. Составить программу для перевода длины в метрах в длину в сантиметрах, определив функцию, выполняющую это преобразование и передав длину в метрах в качестве параметра.
2. Составить программу для нахождения суммы элементов каждого из трёх массивов, введенных с клавиатуры, определив функцию, выполняющую это действие, и передавая массивы в качестве параметра.
3. Даны числа S, T. Получить с использованием функции пользователя F(T,-2S,1.17)+F(2.2,T,S-T) где F(A, B, C) = (2A-B-sin(C))/(5+C).
4. Составить программу перевода двоичной записи натурального числа в десятичную, описав соответствующую функцию с параметром. Перевод осуществлять для чисел, вводимых с клавиатуры. Признак конца ввода - число 0.
5. Даны числа S, T. Получить с использованием функции пользователя с параметрами F=G(1,sin(S))+2G(T*S,24)-G(5,-S), где G(A,B)=(2A+B*B)/(A*B*2+B*5).
6. Составить программу для расчёта значений гипотенузы треугольника, определив функцию, выполняющую этот расчёт. Катеты передаются в качестве параметров.
7. Найти периметр десятиугольника, координаты вершин которого заданы. Определить процедуру вычисления расстояния между двумя точками,

заданными своими координатами, которые передаются функции в качестве параметров из основной программы.

8. Найти периметр шестиугольника, координаты вершин которого заданы. Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами. Координаты передаются функции в качестве параметров из основной программы.
9. Найти площадь пятиугольника, координаты вершин которого заданы. Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами, и процедуру вычисления площади треугольника по трем сторонам. Описать функции с соответствующими формальными параметрами.
10. Составить программу вывода на экран всех натуральных чисел, не превосходящих N и делящихся на каждую из своих цифр. Описать соответствующую функцию, получающую из основной программы в качестве параметра натуральное число и возвращающую TRUE, если оно удовлетворяет указанному условию.
11. Используя подпрограмму - функцию, составить программу для нахождения максимального из трех чисел. Числа передаются функции в качестве параметров.
12. Используя подпрограмму - функцию, составить программу для печати знаков трех чисел, введенных с клавиатуры и передаваемых функции в качестве параметра.
13. Используя подпрограмму - функцию, составить программу для возведения чисел в целую положительную степень. Число передаются функции в качестве параметра из основной программы. Расчет вести для чисел, пока не будет введено число, равное 0.
14. Используя подпрограмму - функцию, составить программу для вычисления функции Z=(X1+Y1)/(X1*Y1), где X1 - первый корень уравнения X^2-4*X-1=0; Y1 - первый корень уравнения $2*Y^2 + A*Y - A^2 = 0$ (A - произвольное).

15. Задав функцию, вывести на печать средние арифметические двух массивов, введенных с клавиатуры. Массив передается функции в качестве параметра.
16. Задав функцию, рассчитать и вывести на печать максимальные значения в трех парах чисел, вводимых с клавиатуры. Пара чисел передается функции в качестве параметра.
17. Найти периметр восьмиугольника, координаты вершин которого заданы. Определить функцию вычисления расстояния между двумя точками, заданными своими координатами. Координаты передать функции в качестве параметров.
18. Даны четыре пары чисел. Получить с использованием функции пользователя наибольший общий делитель для каждой пары.
19. Даны числа A, B, C. Получить с использованием функции пользователя наименьшее значение. Числа передаются функции из основной программы в качестве параметров.
20. Даны числа x = 1,2,...,N. Получить с использованием функции пользователя значения 3*P(X+3)*P(X) для заданных x, где
    $P(X) = 10*X^3 - 14*X^2 + 12*X - 2$.
21. Составить программу для расчета значений катета треугольника, определив функцию, выполняющую этот расчет. Гипотенуза и второй катет передаются в качестве параметров.
22. Даны целые числа a,b,c,d. Проверить с использованием функции пользователя их четность. Число для проверки передается в функцию в качестве параметра из основной программы.
23. Для каждого из 10 введенных с клавиатуры чисел напечатать сообщение: является ли оно простым или нет, описав функцию логического типа, возвращающую значение “ИСТИНА”, если число, переданное ей в качестве параметра, является простым.
24. Даны числа S, T. Получить с использованием функции пользователя Y(T,S)=G(12,S)+G(T,S)-G(2S-1,S*T), где

G(A,B)=(2*A+B*B)/(A*B*2+B*5).

25. Определите функцию, определяющую, какой целой степенью числа 2 является ее аргумент (если число не является степенью двойки - выдать соответствующее сообщение).
26. Определите функцию, подсчитывающую сумму N первых элементов целочисленного массива A. N и массив A передать в качестве параметров.
27. Вычислить количество простых чисел, не превосходящих заданного N. Описать функцию логического типа, возвращающую значение true, если число простое и false в противном случае.
28. Используя функцию с параметрами, вычислить функцию
    $F(X,Y) = (2X^3-4*X^2+X+1)/(9*Y^3+Y+4) + 3*Y^2+5*Y$.
29. Составить программу для перевода веса в граммах в вес в килограммах, определив функцию, выполняющую это преобразование. Вес в граммах передается функции в качестве параметра.
30. Даны числа S, T. Получить с использованием функции пользователя
    P=G(12, S)+G(T, S)-G(2S-1, S*T) где
    G(A, B) = (2*A+B*B)/(A*B*2+B*5).

2.4 ПЗ-4. Реализация приложений с интерфейсами пользователя в C++

1) Цели занятия

Выработать умения и навыки составлять типовые программы с графическим интерфейсом пользователя.

2) Рекомендации и порядок выполнения заданий

Необходимо изучить материалы лекций по этой теме и пункты 1.1 и 1.4 настоящего пособия.

Порядок выполнения задания:

а) Составить программу решения 5 вычислительных задач по своему варианту. Определение номера 1-го задания:

- нечетные по журналу: 1-е задание – это номер по журналу;

- четные по журналу: 1-е задание – это № последнего задания- № по журналу. Если достигнут конец списка вариантов заданий, то перейти к первому варианту и далее.

b) Решить указанные задания и составить блок-схемы алгоритмов решения в электронном виде.

c) Составить отчет, включив в него задание и его выполнение. Представить отчет в электронном виде на проверку.

3) Варианты заданий

Составить интерфейс пользователя для следующих задач.

1. Составить программу для перевода длины в метрах в длину в сантиметрах, определив функцию, выполняющую это преобразование и передав длину в метрах в качестве параметра.
2. Составить программу для нахождения суммы элементов каждого из трех массивов, введенных с клавиатуры, определив функцию, выполняющую это действие, и передавая массивы в качестве параметра.
3. Даны числа S, T. Получить с использованием функции пользователя F(T,-2S,1.17)+F(2.2,T,S-T) где F(A, B, C) = (2A-B-sin(C))/(5+C)
4. Составить программу перевода двоичной записи натурального числа в десятичную, описав соответствующую функцию с параметром. Перевод осуществлять для чисел, вводимых с клавиатуры. Признак конца ввода - число 0.
5. Даны числа S, T. Получить с использованием функции пользователя с параметрами G(1,sin(S))+2G(T*S,24)-G(5,-S),
   где G(A,B)=(2A+B*B)/(A*B*2+B*5).
6. Составить программу для расчета значений гипотенузы треугольника, определив функцию, выполняющую этот расчет. Катеты передаются в качестве параметров.
7. Найти периметр десятиугольника, координаты вершин которого заданы. Определить процедуру вычисления расстояния между двумя точками,

заданными своими координатами, которые передаются функции в качестве параметров из основной программы.

8. Найти периметр шестиугольника, координаты вершин которого заданы. Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами. Координаты передаются функции в качестве параметров из основной программы.
9. Найти площадь пятиугольника, координаты вершин которого заданы. Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами, и процедуру вычисления площади треугольника по трем сторонам. Описать функции с соответствующими формальными параметрами.
10. Составить программу вывода на экран всех натуральных чисел, не превосходящих N и делящихся на каждую из своих цифр. Описать соответствующую функцию, получающую из основной программы в качестве параметра натуральное число и возвращающую TRUE, если оно удовлетворяет указанному условию.
11. Используя подпрограмму - функцию, составить программу для нахождения максимального из трех чисел. Числа передаются функции в качестве параметров.
12. Используя подпрограмму - функцию, составить программу для печати знаков трех чисел, введенных с клавиатуры и передаваемых функции в качестве параметра.
13. Используя подпрограмму - функцию, составить программу для возведения чисел в целую положительную степень. Число передаются функции в качестве параметра из основной программы. Расчет вести для чисел, пока не будет введено число, равное 0.
14. Используя подпрограмму - функцию, составить программу для вычисления функции Z=(X1+Y1)/(X1*Y1), где X1 - первый корень уравнения X^2-4*X-1=0; Y1 - первый корень уравнения $2*Y^2 + A*Y - A^2 = 0$ (A - произвольное).

15. Задав функцию, вывести на печать средние арифметические двух массивов, введенных с клавиатуры. Массив передается функции в качестве параметра.
16. Задав функцию, рассчитать и вывести на печать максимальные значения в трех парах чисел, вводимых с клавиатуры. Пара чисел передается функции в качестве параметра.
17. Найти периметр восьмиугольника, координаты вершин которого заданы. Определить функцию вычисления расстояния между двумя точками, заданными своими координатами. Координаты передать функции в качестве параметров.
18. Даны четыре пары чисел. Получить с использованием функции пользователя наибольший общий делитель для каждой пары.
19. Даны числа A, B, C. Получить с использованием функции пользователя наименьшее значение. Числа передаются функции из основной программы в качестве параметров.
20. Даны числа x = 1,2,...,N. Получить с использованием функции пользователя значения 3*P(X+3)*P(X) для заданных x,
    где $P(X) = 10*X^3 - 14*X^2 + 12*X - 2$.
21. Составить программу для расчета значений катета треугольника, определив функцию, выполняющую этот расчет. Гипотенуза и второй катет передаются в качестве параметров.
22. Даны целые числа a,b,c,d. Проверить с использованием функции пользователя их четность. Число для проверки передается в функцию в качестве параметра из основной программы.
23. Для каждого из 10 введенных с клавиатуры чисел напечатать сообщение: является ли оно простым или нет, описав функцию логического типа, возвращающую значение “ИСТИНА”, если число, переданное ей в качестве параметра, является простым.

24. Даны числа S, T. Получить с использованием функции пользователя Y(T,S)=G(12,S)+G(T,S)-G(2S-1,S*T), где G(A,B)=(2*A+B*B)/(A*B*2+B*5).
25. Определите функцию, определяющую, какой целой степенью числа 2 является ее аргумент (если число не является степенью двойки - выдать соответствующее сообщение).
26. Определите функцию, подсчитывающую сумму N первых элементов целочисленного массива A. N и массив A передать в качестве параметров.
27. Вычислить количество простых чисел, не превосходящих заданного N. Описать функцию логического типа, возвращающую значение true, если число простое и false в противном случае.
28. Используя подпрограмму - функцию с параметрами, составить программу для вычисления функции $F(X,Y) = (2X^3-4*X^2+X+1)/(9*Y^3+Y+4) + 3*Y^2+5*Y$.
29. Составить программу для перевода веса в граммах в вес в килограммах, определив функцию, выполняющую это преобразование. Вес в граммах передается функции в качестве параметра.
30. Даны числа S, T. Получить с использованием функции пользователя G(12, S)+G(T, S)-G(2S-1, S*T) где G(A, B) = (2*A+B*B)/(A*B*2+B*5).

ЗАКЛЮЧЕНИЕ

В учебном пособии подробно и наглядно изложены основы выполнения заданий практических занятий по дисциплине «Разработка кроссплатформенных приложений C++».

Материал пособия снабжен необходимыми примерами выполнения типовых задач разработки клиент-серверных приложений на языке C++ для интегрированной среды разработки Visual Studio. Здесь же приведены необходимые банки заданий для проведения указанных практических занятий.

Пособие соответствует требованиям, предъявляемым к учебно-методическому комплексу указанной дисциплины, обсуждено на заседании кафедры ИВТ и рекомендовано к использованию в учебном процессе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Боровский А. Н. Qt4.7+. Практическое программирование на C++. — СПб.: БХВ-Петербург, 2012. — 496 с.: ил. — (Профессиональное программирование).

2 UCE — Кроссплатформенный C++ фреймворк для разработки приложений с пользовательским интерфейсом. [Электронный ресурс] // [сайт] [2018], URL: https://habr.com/post/209956/. (дата обращения: 03.06.2018).

3 Лаптев В.В. C++. Объектно-ориентированное программирование. — СПб.: Питер, 2010. – 464 с.

4 Лаптев В.В. Морозов А.В., Бокова. C++. Объектно-ориентированное программирование. Задачи и упражнения. — СПб.: Питер, 2011. – 288 с.