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++

Лабораторный 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++: лабораторный практикум. –
Ростов-на-Дону: Северо-Кавказский филиал МТУСИ, 2019. – 48 с.

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……..4
1. Базовые конструкции С++……..6
1.1 Особенности использования функций в С++…….....6
1.1.1 Общие сведения о функциях……..….6
1.1.2 Обмен данными с функциями…….....11
1.1.3 Область видимости переменных в функциях ……..…15
1.2 Структуры языка С++……..…17
1.3 Основы ООП в С++……..…18
1.3.1 Объекты. Создание и удаление…….....18
1.3.2 Виды наследования.…….…….....22
1.3.3 Исключения и их обработка …....…….....28
2 Методические указания по выполнению заданий лабораторных работ……...33
2.1 ЛР-1. Исследование особенностей использования функций в С++……...33
2.2 ЛР-2. Исследование алгоритмов обработки структур
в С++…….....35
2.3 ЛР-3. Исследование консольных приложений в Visual Studio (Qt)……38
2.4 ЛР-4. Исследование приложений с интерфейсами пользователя
в С++……...42
Заключение……47
Список использованных источников……...48

ВВЕДЕНИЕ

Большое количество разнообразных устройств и гаджетов предусматривает обилие различных операционных систем, под управлением которых они работают. Поэтому, сейчас основным требованием к разработкам прикладного программного обеспечения (ПО) различного назначения для вычислительных систем является его кроссплатформенность.

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 основным конструкциям языка программирования C++. Здесь рассмотрены структуры языка, работа с функциями,

или методами пользователя. Также, уделено внимание основам объектно- ориентированного программирования и разработке оконных приложений, т.е. приложений, снабженных интерфейсом. Материал приведен обзорно в расчете на то, что основы языка уже изучены и освоены практически.

Во втором разделе пособия содержатся пояснения и методика выполнения исследований каждой из лабораторных работ по дисциплине «Разработка кроссплатформенных приложений C++».

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

1 Базовые конструкции C++

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

1.1 Особенности использования функций в C++

Основной составляющей любой программы, написанной на C++, являются функции, или еще говорят - методы.

1.1.1 Общие сведения о функциях

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 " Здравствуй, Галактика! Здравствуй, Вселенная!" (листинг 1.1).

Листинг 1.1- Пример обращения к функции

```
#include <iostream >
using namespace std ;
void   f ( )    //Описание функции.
{
cout  <<  "Здравствуй,   " ;
}
int main( )
{
f ( ) ;
//Вызов функции.
cout   <<"Мир!"  << endl ;
f ( ) ;
//Вызов     функции.
cout   <<"Галактика!"  <<  endl ; f ( ) ;    //Вызов функции.
cout   <<"Вселенная!"  << endl ;
}
```

Результатом работы программы будут три строки:

Здравствуй, Мир!

Здравствуй, Галактика! Здравствуй, Вселенная!

Если тип возвращаемого значения не void, то функция может входить в со-

став 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 1.2).

Листинг 1.2- 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 выходит 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 Обмен данными с функциями

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 1.3.

Листинг 1.3 - 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.

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 опера-

тором 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 1.4).

Листинг 1.4 – 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 ;
}
```

1.1.3 Область видимости переменных в функциях

По месту объявления переменные в языке С++ делятся на три класса: локальные, глобальные и переменные, описанные в списке формальных параметров функций. Все эти переменные имеют разную область видимости.

Локальные переменные объявляются внутри функции и доступны только в ней. О таких переменных говорят, что они имеют локальную видимость, то есть, видимы только внутри функции.

Глобальные переменные описывают вне всех функций. Поскольку они доступны из любой точки программы, то их область видимости охватывает весь файл. Одно и тоже имя может использоваться при определении глобальной и локальной переменной. В этом случае в теле функции локальная переменная имеет преимущество и «закрывает» собой глобальную. Вне этой функции «работает» глобальное описание переменной.

Из функции, где действует локальное описание переменной, можно обратиться к глобальной переменной с таким же именем, используя *оператор расширения области видимости* (листинг 1.5)

```
:: переменная;
```

Листинг 1.5 – Примеры классов переменных

```
#include <iostream >
using namespace std ;
float pr = 100.678 ; //Переменная pr определена глобально.
int prostoe (int n)
```

```
{
int pr =1 , i ; //Переменная pr определена локально.
if ( n<0) pr =0;
else
for ( i =2; i <=n / 2 ; i ++)
if ( n%i ==0){ pr =0; break ; }
cout<<" l o c a l  p r = "<<pr<<" \ n " ;    //Вывод локальной переменной.
cout<<" g l o b a l  p r = " <<:: pr<<" \ n " ;   //Вывод глобальной переменной.
return pr ;
}
int main ( )
{
int g ;
cout<<" g = " ; c i n >>g ;
if ( p r o s t o e ( g ) ) cout<<" g - p r o s t o e \ n " ;
else cout<<" g - n e p r o s t o e \ n " ;
return 0 ;
}
```

Результаты работы программы:

g=7

local pr=1 //Локальная переменная.

global pr=100.678 //Глобальная переменная. g - prostoe

Итак, любая программа на C++ состоит из одной или нескольких функций, причём одна из них должна обязательно носить имя main (основной, главный). Именно этой функции передаётся управление после запуска программы. Как любая функция, main может принимать параметры и возвращать значения. У функции main две формы записи:

- без параметров: тип main ( ) {тело функции };
- и с двумя параметрами: тип main (int argc, char ∗ argv[ ]) {тело функции } .

Первый параметр argc определяет количество параметров, передаваемых в функцию main из командной строки. Второй параметр argv — указатель на массив указателей типа char (массив строк). Каждый элемент массива ссылается на отдельный параметр командной строки. При стандартном запуске программы argc равно 1, argv — массив из одного элемента, этим элементом является имя запускаемого файла.

1.2 Структуры языка C++

Если возникает необходимость обрабатывать разнородную информацию как единое целое, то применяют тип данных структуры. Он позволяет сгруппировать объекты различных типов данных под одним именем.

Для того, чтобы объявить переменные структурного типа, вначале нужно задать новый тип данных, указав имя структуры и её элементы. Элементы структуры называются полями, и могут иметь любой тип данных кроме типа этой же структуры. Далее приведён пример создания структурного типа student, полями которого являются фамилия студента, шифр группы, год начала обучения и оценки по четырём предметам:

```
struct student
{
//Поля структуры:
char fio[30] ; char group[8] ; int year ;
int informatika , math , fizika , history;
}
```

На основании созданного структурного типа данных можно описать переменные типа student:

```
student Vasya ; //Переменная Vasya типа student.
student ES[50] ;  //Массив, элементы которого имеют тип student.
student*x ; //Указатель на тип данных student.
```

Обращаются к полям переменной структурного типа так:

```
имя_структуры.поле
```

Например:

```
Vasya.y e a r ; //Обращение к полю year переменной Vasya.
ES[4].math ; //Обращение к полю math элемента ES[4].
```

1.3 Основы ООП в 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.

1.3.1 Объекты. Создание и удаление

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.

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 1.6).

Листинг 1.6- 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 1.7).

Листинг 1.7- 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 1.8.

Листинг 1.8- 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 Виды наследования

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 1.1.

Таблица 1.1- 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 мо-

гут 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 1.9).

Листинг 1.9- 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 1.10).

Листинг 1.10- Код 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.set_z(a.get_z()+b.get_z());
    return c ;
    }
    main()
    {
    spatial_vector a(1,2,3),b(10,20,30),c(0,0,0); c=a+b;
    c.info();
    }
```

Шаблоны классов во многом похожи на шаблоны структур, 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 ко-

торым 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 1.11).

Листинг 1.11- 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(10.1,20.5); 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.

1.3.3 Исключения и их обработка

В C++ есть особенный механизм обработки исключительных ситуаций.

Исключительная ситуация (англ. «exception») или исключение — это что-то особенное или ненормальное, случившееся в работе программы. Чаще всего исключение — это какая-то возникшая ошибка, но не обязательно: например, это может быть нестандартное стечение обстоятельств или просто ситуация, требующая нетиповой обработки.

Если в программе (или в библиотечной функции, которую использует программа) возникает какая-то неразрешимая ситуация, то генерируется исключение. Это означает, что вместо нормального продолжения программы управление передаётся на другую ветку алгоритма, специально предназначенную для обработки такой исключительной ситуации. Эта другая ветвь программы — обработчик исключения — может просто прервать программу, а может что-то изменить, чтобы программа могла продолжить выполнение. Причём, даже если исключение возникает в библиотечной функции, обработчик исключения всё

равно находится в программе, использующей эту библиотеку — в той части кода, которая непосредственно вызвала конкретную нестандартную ситуацию и потому лучше может справиться с её обработкой.

Чтобы исключение, сгенерированное в одном блоке кода, могло найти нужный обработчик, находящийся в другом блоке, при генерации исключения выбрасывается индикатор исключения. Индикатором служит объект или переменная некоторого конкретного типа. При возникновении исключения будет выбран тот обработчик, в описании которого указан тот же тип ожидаемого индикатора. Обработчики различают исключения по типам данных индикатора, и поэтому в наиболее распространённом случае в качестве индикатора указывают объект некоторого класса, специально предусмотренного для этой цели.

Для обработки исключения, нужно поместить вызов кода, в котором исключение может возникнуть, в специальный блок try{}. Следом за этим блоком должен следовать блок catch(){}, внутрь которого помещают код, обрабатывающий исключительную ситуацию. Например, если исключение может возникнуть в некой функции f(), для его обработки нужно написать следующую конструкцию:

```
    f()
    {
    //генерируем исключение, если возникла соответствующая ситуа-
ция
    if ( . . . . ) throw индикатор ;
    . . . . .
    }
    . . . . .
    try
    {
    //вызываем код, который может сгенерировать исключение:
f() ;
    }
    catch ( индикатор )
    {
```

```
//обрабатываем  исключение
. . . . .
}
```

Обработчик исключения всегда следует сразу за блоком try{}. В круглых скобках после catch указывается индикатор исключения, которое этот блок обрабатывает. Чтобы сгенерировать исключение, нужно использовать специальную конструкцию throw, после которой указывается индикатор.

Классы-индикаторы исключения могут принадлежать к общей иерархии наследования, т. е. быть в отношениях «родитель-потомок». При этом обработчики индикаторов-родительских классов могут перехватывать исключения с индикаторами-потомками (можно считать такое поведение проявлением полиморфизма). Поэтому родительские обработчики нужно обязательно указывать после дочерних в цепочке блоков catch — иначе дочерний обработчик никогда не получит управление. В самом конце цепочки можно указать catch, у которого в круглых скобках вместо индикатора три точки. Такой блок будет перехватывать абсолютно любые исключения (листинг 1.12).

Листинг 1.12- Пример перехвата любых исключений

```
class    out_of_range :    public    general_error{ } ;
. . . . . .
try { . . . . . }
catch   ( out_of_range )
{cout<<"Выход  индекса  за  границу  массива\ n " ;    }
catch (general_error)
{cout<<"Общий  сбой  в  работе  программы\ n " ;    }
catch( . . . ){cout<<"Неизвестная  ошибка\ n " ;    }
```

В приведённом схематичном примере мы объявили два различных класса-индикатора, один базовый, для исключений общего типа, и один производный от него, для исключительной ситуации типа «недопустимый индекс при обра-

щении к в массиву». Если бы порядок следования обработчиков был другим, обработчик индикатора out_of_range никогда не смог бы активироваться.

2 Методические указания по выполнению заданий лабораторных работ

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

2.1 ЛР-1. Исследование особенностей использования функций в C++

1) Цели занятия

Практическое исследование составления программ с методами пользователя, особенностей перегрузки методов в C++.

2) Рекомендации и порядок выполнения заданий

Необходимо изучить материалы лекций по этой теме и пункт 1.1 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. Найти наибольшее значение $(x_k + y_k)$ для массивов x и y.

2. Упорядочить элементы массива x(N), расположив их в порядке возрастания в том же массиве.
3. Найти минимальный из элементов массива a(N), принадлежащий интервалу (2; 14).
4. Для массива a(к) найти наибольшее значение разности между рядом стоящими элементами.
5. Найти наибольший элемент матрицы a(к,к), расположенный на главной диагонали.
6. Упорядочить элементы массива x(N), расположив их по убыванию в том же массиве.
7. Найти наибольшие элементы каждой строки матрицы X(M,N) и записать их в массив Y.
8. Найти минимальный из положительных элементов массива a(N)
9. Вычислить суммы элементов каждой строки матрицы X(N,N), определить наименьшее значение этих сумм и номер соответствующей строки.
10. Найти три наименьших элемента массива (a(1),a(2), ..., a(N)).
11. Вычислить наибольшее значения функции Y(I)=2*B(I)2+3B(I), если B(I) задано массивом (B(1),B(2), ..., B(N)).
12. Найти наименьший элемент матрицы a(к,к) из нерасположенных на главной диагонали.
13. Найти наибольшие элементы каждого столбца матрицы X(M,N) и записать их в массив Y.
14. Упорядочить элементы массива X(N), расположив их по убыванию в массиве Y.
15. Найти наибольший элемент матрицы a(M,N) и номер строки и столбца, в которых он находится.
16. Найти наименьший элемент матрицы X(M,N) и записать нули в ту строку и столбец, где он находится.
17. Найти три наибольших элемента массива (a(1),a(2), ..., a(n)).
18. Найти максимальный из отрицательных элементов массива a(n)

19. Найти наименьшее значение (xk +yk) для массивов x и y.
20. Найти максимальный из элементов массива a(n), принадлежащий интервалу (1;25).
21. Для массива a(к) найти наименьшее значение разности между рядом стоящими элементами.
22. Найти наименьший элемент матрицы a(к,к), расположенный на главной диагонали.
23. Найти максимальный и минимальный элементы массива a(n) и поменять их местами
24. Вычислить наименьшее значения функции y(i)=5*b(i)3+3b(i)2-b(i), если b(i) задано массивом (b(1),b(2), ..., b(n)).
25. Найти наименьшие элементы каждого столбца матрицы x(m,n) и записать их в массив y.
26. Найти наибольший элемент матрицы a(к,к) из не расположенных на главной диагонали.
27. Найти наибольшее значение (xk -yk) для массивов x и y.
28. Для массива a(к) найти наибольшее значение суммы между рядом стоящими элементами.

2.2 ЛР-2. Исследование алгоритмов обработки структур в C++

1) Цели занятия

Исследовать практическое составление программы с методами для обработки таких структур, как массивы.

2) Рекомендации и порядок выполнения заданий

Необходимо изучить материалы лекций по этой теме и пункты 1.1 и 1.2 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

1. Вычислить сумму элементов каждого стролбца матрицы A(M,N).
2. Вычислить значение функции $Z = \sum_{i=1}^{10} \sum_{k=1}^{N} \frac{\sin^k x(i)}{k}$, где X(I) заданы массивом (X(1),X(2), ... ,X(10)), K=1,2, ..., N.
3. Вычислить значение функции $Z(j) = \prod_{i=1}^{20} (1 + 1 / e^i + x(j))$, где X(J) заданы массивом (x(1),x(2), .., x(N)). Результаты запомнить в массиве Z.
4. Вычислить сумму элементов матрицы A(N,N), расположенных над главной диагональю.
5. Найти сумму положительных элементов каждого столбца матрицы X(M,N).
6. Вычислить сумму элементов матрицы A(N,N), расположенных под главной диагональю.
7. Из матрицы X(M,N) построить матрицу Y, поменяв местами строки и столбцы.
8. Определить количество положительных и отрицательных элементов матрицы A(M,N).
9. Определить количество положительных элементов каждого столбца матрицы A(M,N) и запомнить их в массиве R.
10. Переписать первые элементы каждой строки матрицы a(M,N) в массив B

11. Даны элементы массива A, состоящего из n элементов. Вычислить $S = A_1^{\,1} + A_2^{\,2} + ... + A_n^{\,n}$ без операций возведения в степень

12. Вычислить значение функции $Z = \sum_{i=1}^{M}\sum_{j=1}^{N}\frac{1}{i + j^2}$.

13. Вычислить значение функции $Z = \sum_{i=1}^{M}\sum_{j=1}^{N}\sin(i^3 + j^4)$.

14. Задана матрица A(M,M) . Разделить элементы каждой строки матрицы A на соответствующий диагональный элемент.

15. Вычислить значение функции $Z = \sum_{i=1}^{M}\sum_{j=1}^{i}\frac{1}{2j + i}$.

16. Дано натуральное число N. Вычислить $\sum_{K=1}^{N} K(K+1)...(K+K)$.

17. Определить количество положительных элементов каждой строки матрицы A(M,N) и запомнить их в массиве B.

18. Дано натуральное число N. Вычислить $\sum_{k=1}^{N}\frac{1}{(k^2)!}$.

19. Дано натуральное число N. Вычислить $\sum_{k=1}^{N}(-1)^k(2k^2+1)!$.

20. Вычислить суммы элементов каждой строки матрицы X(N,N), и записать их в массив Y(N).

21. Даны натуральное число N, действительное число x. Вычислить $\frac{1}{N!}\sum(-1)^k\frac{x^k}{(k!+1)!}$.

22. Даны натуральное число N, действительное число X. вычислить $\sum_{k=1}^{N} k^k x^{2k-1}$ без операции возведения в степень.

23. Даны натуральное число N, действительное число x. Вычислить $\sum_{k=1}^{N}\sum_{m=k}^{N}\frac{x+k}{m}$.

24. Заданы матрица A(5,6) и вектор B(5). Разделить каждый элемент к - ой строки матрицы A на элемент B(K).
25. Заданы матрицы A(м,м) и B(м,м). Получить матрицу X(M,2M), состоящую из M столбцов матрицы A и M столбцов матрицы B.
26. Вычислить значение функции $Z = \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{j-1+1}{i+j}$
27. Найти сумму положительных элементов каждой строки матрицы X(M,N)

2.3 ЛР-3. Исследование консольных приложений в Visual Studio (Qt)

1) Цели занятия

Практически исследовать вопросы составления консольных приложений, вы-явить достоинства и недостатки.

2) Рекомендации и порядок выполнения заданий

Необходимо изучить материалы лекций по этой теме и пункты 1.1 и 1.2 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

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 тре-

угольника 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, за-

данными 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 ЛР-4. Исследование приложений с интерфейсами пользователя в C++

1) Цели занятия

Исследовать основы составления типовых программы с графическим интерфейсом пользователя.

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

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 лабораторных работ 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 (Qt). 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.