Міністерство освіти і науки України

НТУУ „КПІ”

ТЕФ

**Кафедра АПЕПС**

***Звіт***

***До лабораторної роботи №8***

***З дисципліни***

***"Об’єктно-орієнтоване програмування"***

Виконав: **Артамонов О.Ю.**

Студент групи: **ТВ-61**

Перевірив викладач: **Карпенко С.Г.**

**Київ – 2017**

Код програми:

#pragma warning(disable : 4996)

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <iostream>

#include <fstream>

#include <iomanip>

#include <vector>

#include <algorithm>

#include <conio.h>

#include <ctime>

#include <string>

using namespace std;

namespace myexeption {

class MyException {

const char \*message;

public:

MyException(const char \*message = "Unknown exception!") : message(message) {}

const char\* what() const {

return message;

}

};

struct Wrong\_Argument\_Exeption : MyException {

Wrong\_Argument\_Exeption(const char \*message = "Wrong argument exception!") : MyException(message) {}

};

struct Out\_Of\_Range\_Exception : MyException {

Out\_Of\_Range\_Exception(const char \*message = "Out of range exception!") : MyException(message) {}

};

struct Bad\_File\_Exeption : MyException {

Bad\_File\_Exeption(const char \*message = "Bad file exception!") : MyException(message) {}

};

struct Bad\_File\_Write\_Exeption : MyException {

Bad\_File\_Write\_Exeption(const char \*message = "Bad file write exception!") : MyException(message) {}

};

struct Bad\_File\_Read\_Exeption : MyException {

Bad\_File\_Read\_Exeption(const char \*message = "Bad file read exception!") : MyException(message) {}

};

struct Invalid\_File\_Type\_Exeption : MyException {

Invalid\_File\_Type\_Exeption(const char \*message = "Invalid file type exception!") : MyException(message) {}

};

}

class Figures abstract {

public:

virtual void Set() = 0;

virtual inline double Square() = 0;

virtual void Text\_Write(char\*) = 0;

virtual long int Text\_Read(long int, char\*) = 0;

virtual ~Figures() {}

};

class Point {

private:

double x, y;

public:

Point() {

x = 0;

y = 0;

}

Point(double x1, double y1) {

x = x1;

y = y1;

}

Point(const Point &p) {

x = p.x;

y = p.y;

}

void operator() (double x1, double y1) {

x = x1;

y = y1;

}

void Text\_Write(char\*path = "EllipseToRectangle(text).txt") {

ofstream Text\_output\_file(path, ios::app);

if (!Text\_output\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

Text\_output\_file << x << " " << y << endl;

Text\_output\_file.close();

}

long int Text\_Read(long int pos = 0, char \*path = "EllipseToRectangle(text).txt") {

ifstream Text\_input\_file(path);

Text\_input\_file.seekg(pos);

if (!Text\_input\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

double tmpX, tmpY;

Text\_input\_file >> tmpX >> tmpY;

if (Text\_input\_file.fail()) {

Text\_input\_file.close();

throw myexeption::Bad\_File\_Write\_Exeption();

}

if (Text\_input\_file.bad()) {

Text\_input\_file.close();

throw myexeption::Bad\_File\_Write\_Exeption();

}

x = tmpX;

y = tmpY;

long int p = Text\_input\_file.tellg();

Text\_input\_file.close();

return p;

}

friend ostream& operator<<(ostream& os, const Point& p) {

os << p.x << endl;

os << p.y << endl;

return os;

}

friend istream& operator>>(istream& is, Point& p) {

is >> p.x;

is >> p.y;

return is;

}

void Set() {

x = rand() % 100;

y = rand() % 100;

}

void Get() {

cout << "x: " << setw(5) << setprecision(3) << x << ", y: "

<< setw(5) << setprecision(3) << y << (char)179;

}

double getX() { return x; }

double getY() { return y; }

};

class Ellipse : virtual public Figures {

private:

Point p;

double r1, r2;

public:

Ellipse() : p() {

r1 = 1 + rand() % 30;

r2 = 1 + rand() % (int)r1;

}

Ellipse(int R1, int R2) {

r1 = R1 / 2.0;

r2 = R2 / 2.0;

}

Ellipse(const Ellipse &e) {

r1 = e.r1;

r2 = e.r2;

}

void operator() (int R1, int R2, double x, double y) {

p(x, y);

r1 = R1;

r2 = R2;

}

void Text\_Write(char\*path = "EllipseToRectangle(text).txt") {

ofstream Text\_output\_file(path, ios::app);

if (!Text\_output\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

Text\_output\_file << r1 << " " << r2 << endl;

Text\_output\_file.close();

}

long int Text\_Read(long int pos = 0, char \*path = "EllipseToRectangle(text).txt") {

ifstream Text\_input\_file(path);

Text\_input\_file.seekg(pos);

if (!Text\_input\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

double tmpR1, tmpR2;

Text\_input\_file >> tmpR1 >> tmpR2;

if (Text\_input\_file.fail()) {

Text\_input\_file.close();

throw myexeption::Bad\_File\_Write\_Exeption();

}

if (Text\_input\_file.bad()) {

Text\_input\_file.close();

throw myexeption::Bad\_File\_Write\_Exeption();

}

r1 = tmpR1;

r2 = tmpR2;

long int p = Text\_input\_file.tellg();

Text\_input\_file.close();

return p;

}

friend ostream& operator<<(ostream& os, const Ellipse& e) {

os << e.r1 << endl;

os << e.r2 << endl;

os << e.p;

return os;

}

friend istream& operator>>(istream& is, Ellipse& e) {

is >> e.r1;

is >> e.r2;

is >> e.p;

return is;

}

inline double Square() override { return M\_PI \* r1 \* r2; }

void Set() override {

r1 = 1 + rand() % 30;

r2 = 1 + rand() % (int)r1;

}

void Get() {

cout << "r1: " << setw(5) << setprecision(3) << r1

<< ", r2: " << setw(5) << r2 << (char)179;

p.Get();

}

double getR1() { return r1; }

double getR2() { return r2; }

Point getPoint() { return p; }

};

class Rectangle : virtual public Figures {

private:

Point p;

int a, b;

public:

Rectangle() : p() {

a = 1 + rand() % 30;

b = 1 + rand() % 30;

}

Rectangle(int a1, int b1) {

if (a == b || a < 0 || b < 0)

throw myexeption::Wrong\_Argument\_Exeption("Sides of rectangle shouldn't be equal or less than 0.");

a = a1;

b = b1;

}

Rectangle(const Rectangle &rec) {

a = rec.a;

b = rec.b;

}

void operator() (int A, int B, double x, double y) {

p(x, y);

a = A;

b = B;

}

void Text\_Write(char\*path = "EllipseToRectangle(text).txt") {

ofstream Text\_output\_file(path, ios::app);

if (!Text\_output\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

Text\_output\_file << a << " " << b << endl;

Text\_output\_file.close();

}

long int Text\_Read(long int pos = 0, char \*path = "EllipseToRectangle(text).txt") {

ifstream Text\_input\_file(path);

Text\_input\_file.seekg(pos);

if (!Text\_input\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

int tmpA, tmpB;

Text\_input\_file >> tmpA >> tmpB;

if (Text\_input\_file.fail()) {

Text\_input\_file.close();

throw myexeption::Bad\_File\_Write\_Exeption();

}

if (Text\_input\_file.bad()) {

Text\_input\_file.close();

throw myexeption::Bad\_File\_Write\_Exeption();

}

a = tmpA;

b = tmpB;

long int p = Text\_input\_file.tellg();

Text\_input\_file.close();

return p;

}

friend ostream& operator<<(ostream& os, const Rectangle& r) {

os << r.a << endl;

os << r.b << endl;

return os;

}

friend istream& operator>>(istream& is, Rectangle& r) {

is >> r.a;

is >> r.b;

is >> r.p;

return is;

}

inline double Square() override { return a \* b; }

void Set() override {

a = 1 + rand() % 30;

b = 1 + rand() % 30;

}

void Get() {

cout << (char)179 << "a: " << setw(5) << setprecision(3) << a << ", b:" << setw(5) << b << (char)179;

}

int getA() { return a; }

int getB() { return b; }

Point getPoint() { return p; }

};

class EllipseToRectangle {

private:

Ellipse e; //еліпс

Rectangle rec; //прямокутник

double free\_rec; //площа прямокутника вільна від еліпса

string s\_descr;

char \*descr;

char id;

public:

static int Count;

EllipseToRectangle() : rec() {

e(rec.getA() / 2, rec.getB() / 2, 0, 0);

free\_rec = Free\_rec();

Count++;

if (Count == 91)

Count = 65;

id = (char)abs(Count);

s\_descr = id;

s\_descr += "-Ellipse";

descr = new char[s\_descr.size() + 1];

strcpy(descr, s\_descr.c\_str());

}

EllipseToRectangle(EllipseToRectangle &etr) : rec(etr.rec) {

e(etr.getEllipse().getR1(), etr.getEllipse().getR2(), 0, 0);

free\_rec = etr.free\_rec;

s\_descr = etr.s\_descr;

descr = new char[s\_descr.size() + 1];

for (int i = 0; i < 15; i++)

descr[i] = etr.descr[i];

}

void Text\_Write(char\*path = "EllipseToRectangle(text).txt") {

ofstream Text\_output\_file(path, ios::app);

if (!Text\_output\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

Text\_output\_file << e.getR1() << " " << e.getR2() << " " << e.getPoint().getX() << " "

<< e.getPoint().getY() << " " << rec.getA() << " " << rec.getB() << " " << rec.getPoint().getX() << " " << rec.getPoint().getY() << " " << free\_rec <<

" " << descr << endl;

Text\_output\_file.close();

}

long int Text\_Read(long int pos = 0, char \*path = "EllipseToRectangle(text).txt") {

ifstream Text\_input\_file(path);

Text\_input\_file.seekg(pos);

if (!Text\_input\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

double tmpR1, tmpR2, tmpEX, tmpEY, tmpA, tmpB, tmpRX, tmpRY, tmpFR;

Text\_input\_file >> tmpR1 >> tmpR2 >> tmpEX >> tmpEY >> tmpA >> tmpB >> tmpRX >> tmpRY;

(Text\_input\_file >> tmpFR).get();

char \*tmpDSCR = new char[s\_descr.size() + 1];

Text\_input\_file.getline(tmpDSCR, 10, '\n');

if (Text\_input\_file.fail()) {

delete[]tmpDSCR;

Text\_input\_file.close();

throw myexeption::Bad\_File\_Read\_Exeption();

}

if (Text\_input\_file.bad()) {

delete[]tmpDSCR;

Text\_input\_file.close();

throw myexeption::Bad\_File\_Read\_Exeption();

}

strcpy(descr, tmpDSCR);

e(tmpR1, tmpR2, tmpEX, tmpEY);

rec(tmpA, tmpB, tmpRX, tmpRY);

free\_rec = tmpFR;

delete[]tmpDSCR;

long int p = Text\_input\_file.tellg();

Text\_input\_file.close();

return p;

}

friend ostream& operator<<(ostream& os, const EllipseToRectangle& etr) {

os << etr.rec;

os << etr.free\_rec << endl;

os << etr.s\_descr << endl;

os << etr.e;

return os;

}

friend istream& operator>>(istream& is, EllipseToRectangle& etr) {

is >> etr.rec;

is >> etr.free\_rec;

is >> etr.s\_descr;

is >> etr.e;

return is;

}

double Free\_rec() { return rec.Square() - e.Square(); }

void Get() {

rec.Get();

e.Get();

cout << setw(15) << free\_rec << (char)179;

cout << setw(15) << s\_descr << (char)179;

}

Rectangle getRectangle() { return rec; }

double getFreeRec() { return free\_rec; }

char \*getChar() { return descr; }

Ellipse getEllipse() { return e; }

string getString() { return s\_descr; }

~EllipseToRectangle() {

if (descr)

delete[] descr;

}

};

int EllipseToRectangle::Count = 64;

template < typename T1>

void Stat(EllipseToRectangle \*\*arr, T1 n) {

//(SqrFreeRec 1 \* SqrFreeRec 2 \* SqrFreeRec 3 \* ... \* SqrFreeRec n)^(1/n)

double f;

f = pow(arr[0]->getFreeRec(), (1.0 / n));

for (int i = 1; i < n; i++)

f \*= pow(arr[i]->getFreeRec(), (1.0 / n));

cout << "\nGeometric mean for squares of rectangles free of ellipses: " << f << endl;

}

void Char\_Sort(EllipseToRectangle \*\*etr, int n) {

unsigned t = clock();

for (int i = 0; i < n; i++)

for (int j = i + 1; j < n; j++)

if (strcmp(etr[j]->getChar(), etr[j]->getChar()) < 0)

swap(etr[i], etr[j]);

cout << "Time of sort(char's): " << (clock() - t) / 10000.0 << endl;

}

void String\_Sort(EllipseToRectangle \*\*etr, int n) {

unsigned t = clock();

for (int i = 0; i < n; i++)

for (int j = i + 1; j < n; j++)

if (etr[i]->getString().compare(etr[j]->getString()) > 0)

swap(etr[i], etr[j]);

cout << "Time of sort(string's): " << ((float)(clock() - t)) / 10000.0 << endl;

}

void Number\_Sort(EllipseToRectangle \*\*etr, int n) {

unsigned t = clock();

for (int i = 0; i < n; i++)

for (int j = i + 1; j < n; j++)

if (etr[i]->getFreeRec() < etr[j]->getFreeRec())

swap(etr[i], etr[j]);

cout << "Time of sort(num's): " << ((float)(clock() - t)) / 10000.0 << endl;

}

bool My\_Num\_Comp(EllipseToRectangle \*etr1, EllipseToRectangle \*etr2) {

return (etr1->getFreeRec() > etr2->getFreeRec());

}

bool My\_Str\_Comp(EllipseToRectangle \*etr1, EllipseToRectangle \*etr2) {

return (etr1->getString().compare(etr2->getString()) < 0);

}

void Vec(EllipseToRectangle \*\*etr, int n) {

vector<EllipseToRectangle\*> v;

v.reserve(n);

//cout << "Vector" << endl;

for (int i = 0; i < n; i++) {

v.push\_back(etr[i]);

// cout << \*v[i] << endl;

}

unsigned t1 = clock();

sort(v.begin(), v.end(), My\_Num\_Comp);

cout << "\nTime of STL sort(num's): " << ((float)(clock() - t1)) / 10000.0 << endl;

/\*for (int i = 0; i < n; i++)

cout << \*v[i] << endl;\*/

unsigned t2 = clock();

sort(v.begin(), v.end(), My\_Str\_Comp);

cout << "Time of STL sort(string's): " << ((float)(clock() - t2)) / 10000.0 << endl;

/\*for (int i = 0; i < n; i++)

cout << \*v[i] << endl;\*/

v.clear();

}

class ListOfEllipses {

private:

EllipseToRectangle \*\*etr;

int num;

public:

ListOfEllipses() {}

ListOfEllipses(int n) {

if (n < 0)

throw myexeption::Wrong\_Argument\_Exeption();

EllipseToRectangle::Count = 64;

num = n;

etr = new EllipseToRectangle\*[num];

for (int i = 0; i < num; i++)

etr[i] = new EllipseToRectangle;

Stat(etr, num); //Geometric mean for squares of rectangles free of ellipses

Char\_Sort(etr, num); //sort char's

Number\_Sort(etr, num); //sort numbers

String\_Sort(etr, num); //sort string's

Vec(etr, num);

}

friend ostream& operator<<(ostream& os, ListOfEllipses& l) {

for (int i = 0; i < l.getSize(); i++)

os << \*l[i];

return os;

}

friend istream& operator>>(istream& is, ListOfEllipses& l) {

for (int i = 0; i < l.getSize(); i++) {

is >> \*l[i];

}

return is;

}

void Get() {

cout << (char)218;

for (int i = 0; i < 89; i++)

cout << (char)196;

cout << (char)191 << endl;

cout << (char)179 << setw(17) << "Rectangle" << (char)179 << setw(20) << "Ellipse" << (char)179 << setw(18) << "Center point" << (char)179;

cout << setw(15) << "Free square" << (char)179 << setw(15) << "Description " << (char)179 << endl;

cout << (char)195;

for (int i = 0; i < 89; i++)

cout << (char)196;

cout << (char)180 << endl;

for (int i = 0; i < num; i++) {

etr[i]->Get();

cout << endl;

}

cout << (char)192;

for (int i = 0; i < 89; i++)

cout << (char)196;

cout << (char)217;

}

void Text\_Write(char\*path = "EllipseToRectangle(text).txt") {

ofstream Text\_output\_file(path, ios\_base::app);

if (!Text\_output\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

for (int i = 0; i < num; i++)

etr[i]->Text\_Write(path);

Text\_output\_file.close();

}

long int Text\_Read(long int pos = 0, char \*path = "EllipseToRectangle(text).txt") {

ifstream Text\_input\_file(path);

Text\_input\_file.seekg(pos);

if (!Text\_input\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

if (Text\_input\_file.fail()) {

Text\_input\_file.close();

throw myexeption::Bad\_File\_Read\_Exeption();

}

if (Text\_input\_file.bad()) {

Text\_input\_file.close();

throw myexeption::Bad\_File\_Read\_Exeption();

}

Text\_input\_file.get();

long int p = Text\_input\_file.tellg();

for (int i = 0; i < num; i++)

p = etr[i]->Text\_Read(p, path);

Text\_input\_file.close();

return p;

}

int getSize() { return num; }

EllipseToRectangle \*\*getArr() { return etr; }

EllipseToRectangle\* operator [](int index) {

if (index < 0 || index >= num)

throw myexeption::Out\_Of\_Range\_Exception();

return etr[index];

}

~ListOfEllipses() {

for (int i = 0; i < num; i++)

if (etr[i])

delete[] etr[i];

}

};

void getEl(ListOfEllipses &l) {

int n = 0;

cout << "Enter the number of element information about which you would like to know: ";

cin >> n;

if (n < 0 || n >= l.getSize())

throw myexeption::Out\_Of\_Range\_Exception();

for (int i = 0; i < 89; i++)

cout << (char)196;

cout << \*l[n] << endl;

for (int i = 0; i < 89; i++)

cout << (char)196;

}

void Quest(ListOfEllipses &l) {

bool flg = true;

char answ1;

do {

cout << "\nDo you want to print all elements?" << endl << "Please enter Y(Yes) or N(No) >>>> ";

cin >> answ1;

switch (answ1) {

case 'Y': case 'y':

cout << "\n\nAll elements" << endl;

l.Get();

flg = false;

break;

case 'N': case 'n':

flg = false;

break;

default:

cout << "\nI dont know what is mean '" << answ1 << "'" << endl;

break;

}

} while (flg == true);

flg = true;

char answ2;

do {

cout << "\nDo you want to get info about some element ?" << endl << "Please enter Y(Yes) or N(No) >>>> ";

cin >> answ2;

switch (answ2) {

case 'Y': case 'y':

getEl(l);

break;

case 'N': case 'n':

cout << "\nOK." << endl;

flg = false;

break;

default:

cout << "\nI dont know what is mean '" << answ2 << "'" << endl;

break;

}

} while (flg == true);

}

template<class T>

class InFile {

private:

T \*\*arr;

int size\_of\_array;

public:

InFile(T \*\*arg, int n) {

size\_of\_array = n;

arr = new T\*[size\_of\_array];

for (int i = 0; i < size\_of\_array; i++)

arr[i] = new T(\*arg[i]);

}

void Binary\_Write(char \*path = "EllipsesToRectangles(binary).dat") {

ofstream Binary\_output\_file(path, ios::binary);

if (!Binary\_output\_file.is\_open())

throw myexeption::Bad\_File\_Write\_Exeption();

for (int i = 0; i < size\_of\_array; i++)

Binary\_output\_file.write((char\*)&arr[i], sizeof(&arr[i]));

Binary\_output\_file.close();

}

void Binary\_Read(T \*\*etr, int n, char \*path = "EllipsesToRectangles(binary).dat") {

ifstream Binary\_input\_file(path, ios::binary);

if (!Binary\_input\_file.is\_open())

throw myexeption::Bad\_File\_Read\_Exeption();

else

for (int i = 0; i < n; i++)

Binary\_input\_file.read((char\*)&etr[i], sizeof(T));

cout << "File read out!" << endl;

Binary\_input\_file.close();

}

void Text\_Write(char \*path = "EllipsesToRectangles(text).txt") {

ofstream Text\_output\_file(path, ios::app);

if (!Text\_output\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

Text\_output\_file << typeid(T).hash\_code() << endl;

for (int i = 0; i < size\_of\_array; i++)

arr[i]->Text\_Write(path);

Text\_output\_file.close();

}

void Text\_Read(T \*\*arg, int n, char \*path = "EllipsesToRectangles(text).txt") {

ifstream Text\_input\_file(path);

if (!Text\_input\_file.is\_open())

throw myexeption::Bad\_File\_Exeption();

size\_t hash\_of\_type;

Text\_input\_file >> hash\_of\_type;

long int p = Text\_input\_file.tellg();

for (int i = 0; i < n; i++) {

Text\_input\_file.seekg(p);

if (Text\_input\_file.fail()) {

Text\_input\_file.close();

throw myexeption::Bad\_File\_Read\_Exeption();

}

if (Text\_input\_file.bad()) {

Text\_input\_file.close();

throw myexeption::Bad\_File\_Read\_Exeption();

}

if (hash\_of\_type == typeid(T).hash\_code())

p = arg[i]->Text\_Read(Text\_input\_file.tellg(), path);

}

}

~InFile() {

for (int i = 0; i < size\_of\_array; i++)

if (arr[i])

delete[] arr[i];

}

};

int main(int argc, const char \* argv[]) {

srand(time(NULL));

int count\_of\_ellipses = 0;

try {

cout << "Enter count of ellipses: ";

cin >> count\_of\_ellipses;

if (count\_of\_ellipses < 0)

throw myexeption::MyException("Bad value! Count of ellipses don't be less than 0.");

ListOfEllipses l(count\_of\_ellipses);

InFile<EllipseToRectangle> f(l.getArr(), l.getSize());

f.Text\_Write();

f.Binary\_Write();

ListOfEllipses binary\_test(count\_of\_ellipses);

ListOfEllipses text\_test(count\_of\_ellipses);

f.Binary\_Read(binary\_test.getArr(), binary\_test.getSize());

f.Text\_Read(text\_test.getArr(), text\_test.getSize());

Quest(l);

Quest(binary\_test);

Quest(text\_test);

}

catch (myexeption::MyException &ex) {

cout << endl << ex.what() << endl;

}

catch (...) {

throw myexeption::MyException();

}

\_getch();

return 0;

}

Контрольні запитання:

* 1. Шаблони – це припис для створення класу, в якому один або декілька типів або значень параметризовані.

Приклад:

template<class T>

class Example {

private:

T tmp;

public:

void Show() {

cout << tmp << endl;

}

};

* 1. Якщо клас являється похідним від декількох базових класів, конструктори базових класів викликаються в тому порядку, в якому вони перераховані в оголошенні похідного класу.

7.1 Стандартна бібліотека С містить дві функції, які виконують форматований введення і виведення стандартних типів даних: printf () і scanf (). Термін форматований має на увазі, що ці функції можуть, читати або писати дані в різних форматах, якими можна управляти, Функція printf () використовується для виведення даних на консоль, scanf () - для читання даних з консолі. Як printf (), так і scanf () можуть працювати з будь-якими стандартними типами, включаючи символи, рядки і числа. В С ++ використовується механізм потокового введення / виводу, який зазвичай називаю неформатований. Потік - механізм перетворення значень різного типу в послідовність символів (висновок) і навпаки (введення) #include <iostream>

Часто програмістові потрібно більш детальне управління. Управління здійснюється для кожного потоку окремо, тобто якщо ви визначили якийсь формат для потоку введення, то на потік виведення він не подіє. Більш того, управління впливає тільки на наступне виведене значення. #include <iomanip>

Управління можна організувати використовуючи маніпулятори потоку, які вставляються між об'єктами (записуваними або читаються) і тим самим змінюють стан потоку (формат виведення значень).

Приклад:  
1)Форматований

int main() {

int n;

scanf("%d", &n);

printf("n = %d", &n);

return 0;

}

1)Неформатований

int main(){

int n;

cin >> n;

cout << "n = " << n << endl;

return 0;

}

7.2 Частиною стандартної бібліотеки C ++ є бібліотека iostream - об'єктно-орієнтована ієрархія класів, де використовується і множинне, і віртуальне успадкування. У ній реалізована підтримка для файлового введення / виводу даних вбудованих типів. Крім того, розробники класів можуть розширювати цю бібліотеку для читання і запису нових типів даних.