ЛАБОРАТОРНАЯ РАБОТА №8

“Программирование с использованием перегрузки функций и операторов”

**Цель работы:** сформировать знания и умения по использованию принципа полиморфизма в ЯП Си++, приобрести практические навыки создания программ на языке Си++ по перегрузке функций, операций, использованию дружественных функций.

Задание 2:

Пример 1:

#include <iostream>

using namespace std;

class complex{

int Re,Im;

public: void input();//ввод комплексного числа

void output();//вывод комплексного числа

complex operator+(complex ob2);//переопределяем оператор +};

void complex::input(){//ввод

cout<<"Input Re"; cin>>Re;

cout<<"Input Im"; cin>>Im;

}

void complex::output(){//вывод

cout<<Re<<"+i"<<Im;

}

complex complex::operator +(complex ob2){

complex temp; // переменная для результата сложения чисел

temp.Re=Re+ob2.Re;// складываем действительные части

temp.Im=Im+ob2.Im;// складываем мнимые части

return temp;// возвращаем результат

}

int main(){

complex A,B,C;//объявляем три переменные типа complex

A.input();//ввод числа A

B.input();//ввод числа B

C=A+B;//сложение двух комплексных чисел

C.output();//вывод результата

return 0;}

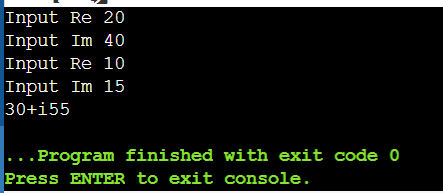


Рисунок 1 – выполнение кода примера 1

Пример 2:

#include<iostream>

using namespace std;

class blockFence;//предварительное объявление

class ground {//класс участок

int length, width; //размеры участка

public: ground(int l, int w) { length = l; width = w; };//конструктор

//объявляем дружественную функцию

friend int countBlockFence(ground g, blockFence b);};

class blockFence {//заборный блок

int width; //ширина

public: blockFence(int w) { width = w; };//конструктор

// объявляем дружественную функцию

friend int countBlockFence(ground g, blockFence b);

};

int countBlockFence(ground g, blockFence b) {/\*возвращает количество заборных блоков для

ограждения участка с трех сторон\*/

return (g.length + 2 \* g.width) / b.width;

}

int main() {

ground g(100, 30); //участок размером 100 на 30 метров

blockFence b1(2); //заборный блок размером 10 метров

cout << "Вам потребуется " << countBlockFence(g, b1) << " блоков";

return 0;

}

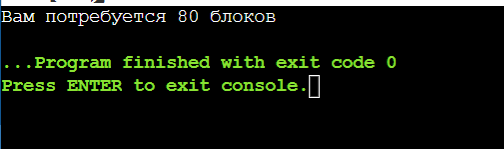


Рисунок 2 – выполнение кода примера 2

Пример 3:

#include<iostream>

using namespace std;

class complex {

public:

int Re, Im;

public:

complex operator+(complex ob2);//переопределяем оператор +

friend ostream & operator<<(ostream& stream, complex ob);//переопределяем оператор вывода

friend istream & operator>>(istream& stream, complex& ob);//переопределяем оператор ввода

};

complex complex::operator +(complex ob2) {

complex temp; // переменная для результата сложения чисел

temp.Re = Re + ob2.Re;// складываем действительные части

temp.Im = Im + ob2.Im;// складываем мнимые части

return temp;// возвращаем результат

}

ostream & operator<<(ostream & stream, complex ob) {

stream << ob.Re << "+i" << ob.Im;

return stream;

};

istream& operator>>(istream& stream, complex& ob) {

cout << endl << "Input Re"; cin >> ob.Re;

cout << "Input Im"; cin >> ob.Im;

return stream;}

int main() {

complex A, B, C;

cin >> A >> B;//ввод чисел

C = A + B;//сложение

cout << A << "+" << B << "=" << C;//вывод результата

return 0;}

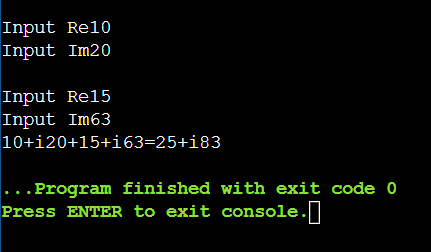


Рисунок 3 – выполнение кода примера 3

**Задание 3:**

Переопределить операцию \* для класса fraction. Обратите внимаение, что придется два раза перегружать оператор \*: один раз для умножения на целое число, а другой раз для умножения дроби на дробь.

#include<iostream>

using namespace std;

class fraction {

public:

int m; //числитель

int n; //знаменатель

public:

fraction() // для с

{

m = 0;

n = 0;

};

fraction(int a, int b);

void output();// вывод дроби

void mult(int a);// умножение дроби на целое число

fraction operator \*(const fraction& other);

};

//Программируем методы класса fraction

fraction::fraction(int a, int b) {

m = a;

n = b;

}

void fraction::output() {// вывод дроби

cout << m << "/" << n<<endl;

}

void fraction::mult(int a) {// умножение дроби на число

m = m \* a;

}

fraction fraction :: operator \*(const fraction & other) {

fraction temp;//переменная для сохранения результата

temp.m = this->m \* other.m;

temp.n = this->n \* other.n;

return temp;

}

//Программируем алгоритм решения основной задачи

int main() {

// ввод дроби

cout << "Vvedite drob': " << endl;

fraction A(1,2); //дробь

fraction B(3,4);

cout << "drob 1: ";

A.output();

cout << endl;

cout << "drob 2: ";

B.output();

cout << endl;

fraction c = A \* B;

cout << "drob\*drob: ";

c.output();

int k; // число на которое нужно умножить дробь

/\*Чтобы ввести дробь А, нужно у объекта A вызвать метод input. \*/

//умножение дроби на целое число

cout << "Vvedite chislo na kotoroe nuzno ymnozit' drob 1'";

cin >> k;

/\*Чтобы умножить дробь А на число k нужно у объекта A вызвать \*/

A.mult(k);//вызываем метод mult для дроби A

// вывод дроби

A.output();//вызываем метод output для дроби A

return 0;

}

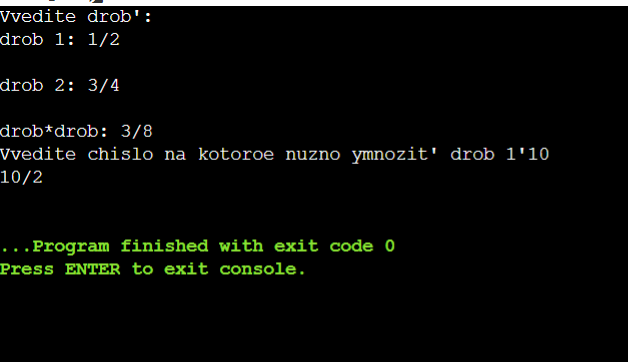


Рисунок 4 – выполнение кода

**Задание 4:**

1.Перегрузить операции ввода вывода для классов ground и blockFence, 2.а также операцию вывода для класса dinamicArray.

**1.**

#include<iostream>

using namespace std;

class complex {

public:

int Re, Im;

};

class blockFence;//предварительное объявление

class ground {//класс участок

public:

int length, width; //размеры участка

public:

ground()

{ length = 0; width = 0; };//конструктор

friend ostream& operator<<(ostream& stream, ground& ob);//переопределяем оператор вывода

friend istream& operator>>(istream& stream, ground& ob);//переопределяем оператор ввода

//объявляем дружественную функцию

friend int countBlockFence(ground g, blockFence b);

};

class blockFence {//заборный блок

public:int width; //ширина

public: blockFence()

{ width = 0; };//конструктор

// объявляем дружественную функцию

friend int countBlockFence(ground g, blockFence b);

friend ostream& operator<<(ostream& stream, blockFence& ob);//переопределяем оператор вывода

// friend istream& operator>>(istream& stream, blockFence& ob);//переопределяем оператор ввода

};

ostream& operator<<(ostream& stream, blockFence& ob) {

stream << " width b1: " << ob.width<<endl;

return stream;

};

istream& operator>>(istream& stream, blockFence& ob) {

cout << "Input width"; cin >> ob.width;

return stream;

}

ostream& operator<<(ostream& stream, ground& ob) {

stream << "length: " << ob.length <<" width: " << ob.width<<endl;

return stream;

};

istream& operator>>(istream& stream, ground& ob) {

cout << endl << "Input length: "; cin >> ob.length;

cout << "Input width: "; cin >> ob.width;

return stream;

}

int countBlockFence(ground g, blockFence b) {/\*возвращает количество заборных блоков для

ограждения участка с трех сторон\*/

return (g.length + 2 \* g.width) / b.width;

}

int main() {

ground g; //участок размером 100 на 30 метров

cin >> g;

cout << g;

blockFence b1; //заборный блок размером 10 метров

cin >> b1;

cout << b1;

cout << "Potrebuetsya " << countBlockFence(g, b1) << " blokov";

return 0;}

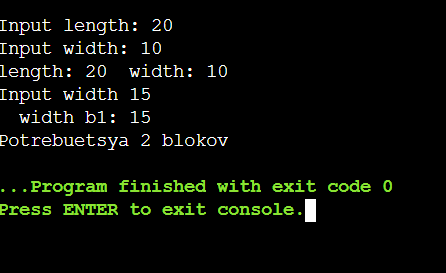
****

Рисунок 5 – выполнение кода

**2.**

#include<iostream>

using namespace std;

class dinamicArray {

int\* p = new int;// выделили память для переменной целого типа

int lenght;// размер

int\* a;//указатель на массив целых чисел в динамической памяти

public:

dinamicArray(int n);//конструктор

int sum();// сумма элементов массива

void output();//вывод массива

~dinamicArray() { delete a; };//деструктор

friend ostream& operator<<(ostream& stream, dinamicArray& ob);};

dinamicArray::dinamicArray(int n) {//конструктор

lenght = n;//устанавливаем длину

a = new int[n];// выделяем память

for (int i = 0; i < lenght; i++)a[i] = i;//заполняем массив}

int dinamicArray::sum() {//сумма элементов массива

int s = 0; // вспомогательная переменная для накопления суммы

for (int i = 0; i < lenght; i++)s += a[i];

return s;}

ostream& operator<<(ostream& stream, dinamicArray& ob) {

for (int i = 0; i < ob.lenght; i++) { stream << "a["<<i<<"]" << ob.a[i] << endl; }

return stream;};

void dinamicArray::output() {//вывод элементов массива

cout << endl;

for (int i = 0; i < lenght; i++)cout << a[i] << " ";

delete p; //освободили память, на которую указывает указатель p.}

int main() {

dinamicArray A(5);//создаем массив из пяти элементов

cout << endl << "Array A";

cout << A;

cout << "Sum of elements " << A.sum();//вывод суммы элементов

return 0;}

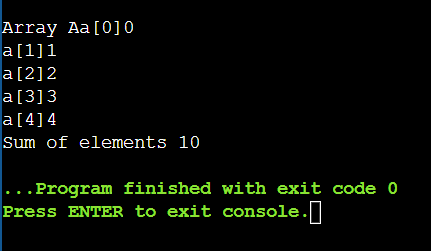


Рисунок 6 – выполнение кода

**Задание 5:**

В задаче об ограждении участка добавьте заборному блоку свойство цена и создайте функцию, которая из массива заборных блоков будет выбирать наиболее дешевый вариант

#include<iostream>

using namespace std;

class blockFence;

class ground {

public:

int length, width;

public:

ground()

{

length = 0; width = 0;};};

class blockFence {

public:int width;

int price;

public:

blockFence()

{

width = 0;

};

friend void choise();};

void choise()

{

blockFence A[3];

cout << "ЗАПОЛНИТЕ ИНФОРМАЦИЮ О БЛОКАХ" << endl << endl;

for (int i = 0; i < 3; i++)

{

cout << i << " блок: " << endl;

cout << "Введите ширину: ";

cin >> A[i].width;

cout << "Введите цену: ";

cin >> A[i].price;

cout << endl;

}

for (int i = 0; i < 3; i++)

{

if (A[i].price < A[i + 1].price)

{

int temp;

int temp2;

temp2 = A[i + 1].width;

A[i + 1].width = A[i].width;

A[i].width = temp2;

temp = A[i + 1].price;

A[i + 1].price = A[i].price;

A[i].price = temp;

}

}

cout << "Самая низкая цена: " << A[2].price << ", ширина которого " << A[2].width << ".";

}

int main()

{

setlocale(LC\_ALL, "rus");

int c = 0;

choise();

return 0;

}

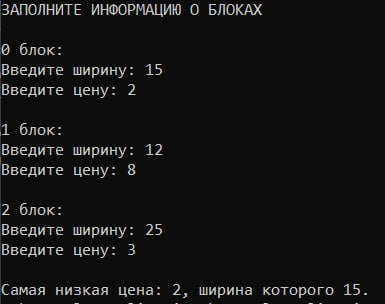


Рисунок 7 – выполнение кода

**Контрольные вопросы**

**1. Что подразумевает под собой механизм перегрузки функций?**

В Си++ реализован механизм перегрузки. Внутри класса допускается существование нескольких функций с одинаковым именем, но различающимися типами результатов и наборами параметров. Например, для класса fraction можно создать метод с одинаковый названием mult, который будет выполнять умножение, но в зависимости от того на что надо умножать дробь, будут реализованы разные алгоритмы.

**2. Как перегрузить бинарный и унарный оператор?**

Когда оператор функция – член класса перегружает унарный оператор у нее не будет параметров, а когда бинарный оператор, у функции будет один параметр. Этот параметр получит тот объект, который будет расположен справа от знака оператора. Объект слева генерирует вызов оператор-функции и передается неявно с помощью указателя this.

**3. В чем особенность дружественных функций?**

Дружественные функции – функциями, которые не являются членами класса, но имеют доступ к его закрытым членам.

1. Дружественная функция может быть определена внутри самого класса, однако это самостоятельная функция, которая имеет право доступа к частным (private) и защищённым ( protected ) членам класса.

2. Отличие дружественных функций от обычных функций членов класса состоит в том, что для них не используется указатель THIS .

**4. В чем особенность переопределения операторов ввода/вывода?**

Пользовательская функция ввода не может быть членом класса, для которого она создана. Если оператор функция является членом класса, то левый операнд является объектом, генерирующим вызов оператор функции. Т.е. левый операнд должен быть объектом этого класса. Однако в случае операторов ввода/вывода левый операнд – это поток, а правый – объект, который нужно вывести. Как и в предыдущем примере, оператор функцию можно сделать дружественной.