**ПРАКТИЧНЕ ЗАВДАННЯ № 6**

**Завдання №1 :**

Для класса coord (См. Unit7 "Overloading An Operator In Which The Order Of The Operands Is Important") перегрузите операторы \* и /. Продемонстрируйте их работу.

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y;

public:

Coord() { x = 0; y = 0; }

Coord(int i, int j) { x = i; y = j; }

void get\_xy(int &i, int &j) { i = x ; j = y; }

Coord operator/ (Coord obj2);

Coord operator\* (Coord obj2);

};

Coord Coord::operator/(Coord obj2){

Coord temp;

temp.x = x / obj2.x;

temp.y = y / obj2.y;

return temp;

}

Coord Coord::operator\*(Coord obj2){

Coord temp;

temp.x = x \* obj2.x;

temp.y = y \* obj2.y;

return temp;

}

int main()

{

Coord object1(20, 15), object2(2, 3), object3;

int x, y;

object3 = object1 / object2;

object3.get\_xy(x, y);

cout << "X = " << x << " Y = "<< y << endl;

object3 = object1 \* object2;

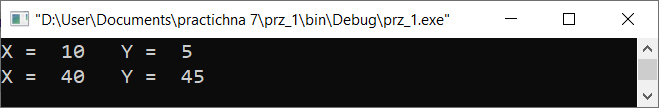
object3.get\_xy(x, y);

cout << "X = " << x << " Y = "<< y << endl;

return 0;

}

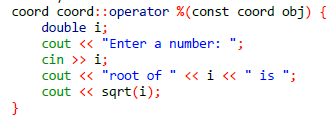
**Результати роботи програми (скриншоти)**



**Завдання №2 :**

В приведенном ниже примере некорректно перегружен оператор *%.*

Почему?



**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y;

public:

Coord() { x = 0; y = 0; }

Coord(int i, int j) { x = i; y = j; }

void get\_xy(int &i, int &j) { i = x ; j = y; }

Coord operator% (Coord obj2);

};

Coord Coord::operator% (Coord obj2){

Coord temp;

temp.x = x % obj2.x;

temp.y = y % obj2.y;

return temp;

}

int main()

{

Coord object1(20, 15), object2(6, 9), object3;

int x, y;

object3 = object1 % object2;

object3.get\_xy(x, y);

cout << "X = " << x << " Y = "<< y << endl;

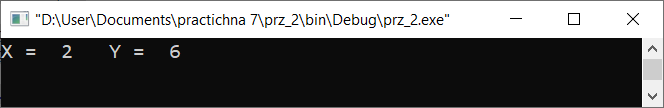
return 0;

}

/\* Реалізація перезавантаження оператора через метод, поданий вище, неможливий адже в функцію передається об’єкт *const coord obj*, що не використовується ніяк, крім того функція нічого не повертає, тобто її тип void, а не coord, як має бути при перезавантажені операторів.

Результат роботи цієї перезавантаженої функції не збігається з результато традиційного оператора. Оператор % перезавантажується, як і будь-який інший арифметичний оператор\*/

**Результати роботи програми (скриншоти)**



**Завдання №3 :**

Поэкспериментируйте, меняя тип возвращаемого значения оператор- функций на что-нибудь отличное от coord. Обратите внимание на генерируемые компилятором сообщения об ошибках. Объясните причины ошибок.

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y;

public:

Coord() { x = 0; y = 0; }

Coord(int i, int j) { x = i; y = j; }

void get\_xy(int &i, int &j) { i = x ; j = y; }

int operator\* (Coord obj2);

};

int Coord::operator\*(Coord obj2){

Coord temp;

//2) int temp;

temp.x = x \* obj2.x;

temp.y = y \* obj2.y;

return temp;

}

int main()

{

Coord object1(20, 15), object2(2, 3), object3;

int x, y;

object3 = object1 \* object2;

object3.get\_xy(x, y);

cout << "X = " << x << " Y = "<< y << endl;

return 0;

}

/\*1) connot conver 'Coord' to 'int' in return;

int Coord::operator\*(Coord obj2)

функція int operator\* (Coord obj2) не типу Coord, тому не може повернути

temp - об'єкт типу Coord;

2) int Coord::operator\*(Coord obj2){

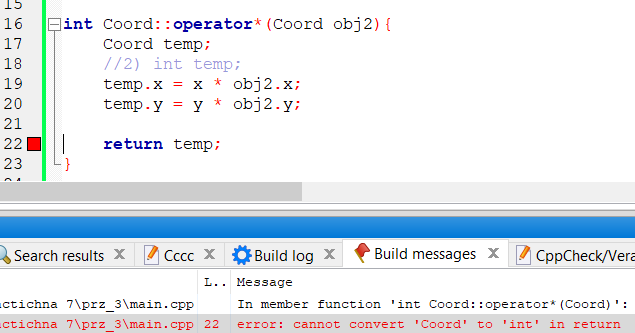
int temp;

request for member 'x' in 'temp' which is of non-class type 'int'

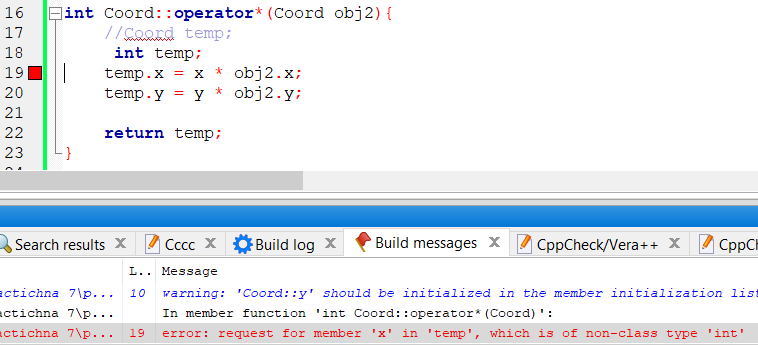
temp не є об'єктом типу класу, тому звернення до членів класу x, y через нього неможливе\*/

**Результати роботи програми (скриншоти)**

1)



2)



**Завдання №4 :**

Относительно класса coord (См. Unit7 "Overloading The Relational And Logical Operators") перегрузите операторы отношения < и >.

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y;

public:

Coord() { x = 0; y = 0; }

Coord(int i, int j) { x = i; y = j; }

void get\_xy(int &i, int &j) { i = x ; j = y; }

int operator< ( Coord obj2);

int operator> ( Coord obj2);

};

int Coord::operator< ( Coord obj2){

return x < obj2.x && y < obj2.y;

}

int Coord::operator> ( Coord obj2){

return x > obj2.x && y > obj2.y;

}

int main()

{

Coord object1(20, 15), object2(2, 3);

if(object1 > object2)

cout << "object1 > object2 " << endl;

else

cout << "object1 < object2 " << endl;

if(object1 < object2)

cout << "object1 < object2 " << endl;

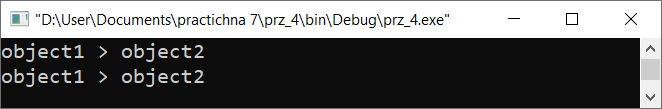
else

cout << "object1 > object2 " << endl;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №5 :**

Перегрузите оператор минус - относительно класса coord. Создайте его префиксную и постфиксную формы.

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y;

public:

Coord() { x = 0; y = 0; }

Coord(int i, int j) { x = i; y = j; }

void get\_xy(int &i, int &j) { i = x ; j = y; }

Coord operator-- ();

Coord operator-- (int notused);

Coord operator++ ();

Coord operator++ (int notused);

};

Coord Coord::operator--(){

x--;

y--;

return \*this;

}

Coord Coord::operator--(int notused) {

x--;

y--;

return \*this;

}

Coord Coord::operator++(){

x++;

y++;

return \*this;

}

Coord Coord::operator++(int notused) {

x++;

y++;

return \*this;

}

int main()

{

Coord object1(20, 15), object2(2, 3);

int x, y;

object1--;

object1.get\_xy(x, y);

cout << " object1-- X = " << x << " Y = "<< y << endl;

--object1;

object1.get\_xy(x, y);

cout << " --object1 X = " << x << " Y = "<< y << endl;

object2++;

object2.get\_xy(x, y);

cout << " object2++ X = " << x << " Y = "<< y << endl;

++object2;

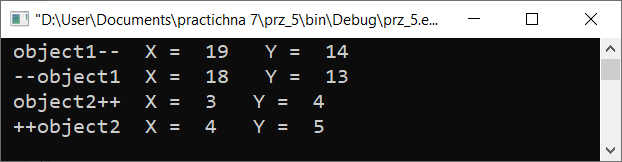
object2.get\_xy(x, y);

cout << " ++object2 X = " << x << " Y = "<< y << endl;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №6 :**

Перегрузите оператор + относительно класса coord так, чтобы он был как бинарным, так и унарным оператором (См. Unit7 "Overloading A Unary Operator"). При использовании в качестве унарного оператор + должен делать положительным значение любой отрицательной координаты.

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y;

public:

Coord() { x = 0; y = 0; }

Coord(int i, int j) { x = i; y = j; }

void get\_xy(int &i, int &j) { i = x ; j = y; }

Coord operator+ ();

Coord operator+ (const Coord obj2);

};

Coord Coord::operator+(const Coord obj2){

Coord temp;

temp.x = x + obj2.x;

temp.y = y + obj2.y;

return temp;

}

Coord Coord::operator+ () {

x = (x < 0 ) ? -x : x;

y = (y < 0 ) ? -y : y;

return \*this;

}

int main()

{

Coord object1(1, 1), object2(-2, 3), object3;

int x, y;

object1 = object1 + object2;

object1.get\_xy(x, y);

cout << "x = " << x << " y = " << y << endl;

object2 = +object2;

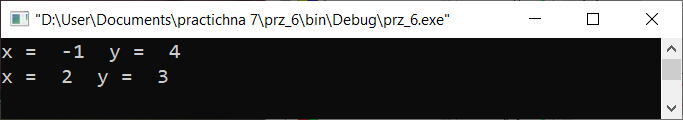
object2.get\_xy(x, y);

cout << "x = " << x << " y = "<< y << endl;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №7 :**

Перегрузите операторы - и / для класса coord посредством дружественных функций.

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y;

public:

Coord() { x = 0; y = 0; }

Coord(int i, int j) { x = i; y = j; }

void get\_xy(int &i, int &j) { i = x ; j = y; }

Coord operator- ();

friend Coord operator- (Coord obj1, Coord obj2);

friend Coord operator/ (Coord obj1, Coord obj2);

friend Coord operator- (int i, Coord obj2);

};

Coord operator- (Coord obj1, Coord obj2){

Coord temp;

temp.x = obj1.x - obj2.x;

temp.y = obj1.y - obj2.y;

return temp;

}

Coord operator/ (Coord obj1, Coord obj2){

Coord temp;

temp.x = obj1.x / obj2.x;

temp.y = obj1.y / obj2.y;

return temp;

}

Coord operator- (int i, const Coord obj2){

Coord temp;

temp.x = obj2.x - i;

temp.y = obj2.y - i;

return temp;

}

int main() {

Coord object1(3, 5), object2(1, 2), object3;

int x, y;

object3 = object1 - object2;

object3.get\_xy(x, y);

cout << "x " << x << " y " << y << endl;

object3 = 5 - object3 ;

object3.get\_xy(x, y);

cout << "x " << x << " y " << y << endl;

object3 = object1 / object2;

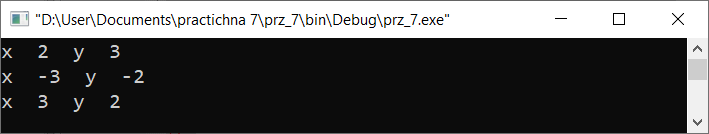
object3.get\_xy(x, y);

cout << "x " << x << " y " << y << endl;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №8 :**

Перепишите класс coord так, чтобы можно было использовать объекты типа coord для умножения каждой из координат на целое. Должны быть корректными обе следующие инструкции: obj \* int и int \* obj. Объясните, почему решение требует использования дружественных оператор-функций. Покажите, как с помощью дружественной оператор-функции перегрузить оператор - относительно класса coord. Определите как префиксную, так и постфиксную формы.

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y;

public:

Coord() { x = 0; y = 0; }

Coord(int i, int j) { x = i; y = j; }

void get\_xy(int &i, int &j) { i = x ; j = y; }

Coord operator- ();

friend Coord operator\* (const Coord obj1, int i);

friend Coord operator\* (int i, const Coord obj2);

friend Coord operator-- (Coord &obj);

friend Coord operator-- (Coord &obj, int unused);

};

Coord operator\* (Coord obj1, int i){

Coord temp;

temp.x = obj1.x \* i;

temp.y = obj1.y \* i;

return temp;

}

Coord operator\* (int i, Coord obj2){

Coord temp;

temp.x = obj2.x \* i;

temp.y = obj2.y \* i;

return temp;

}

Coord operator-- (Coord &obj){

--obj.x;

--obj.y;

return obj;

}

Coord operator-- (Coord &obj, int unused){

--obj.x;

--obj.y;

return obj;

}

int main() {

Coord object1(3, 5), object2;

int x, y;

object1--;

object1.get\_xy(x, y);

cout << "x " << x << " y " << y << endl;

object2 = object1 \* 4;

object2.get\_xy(x, y);

cout << "x " << x << " y " << y << endl;

object2 = 2 \* object1 ;

object2.get\_xy(x, y);

cout << "x " << x << " y " << y << endl;

--object2;

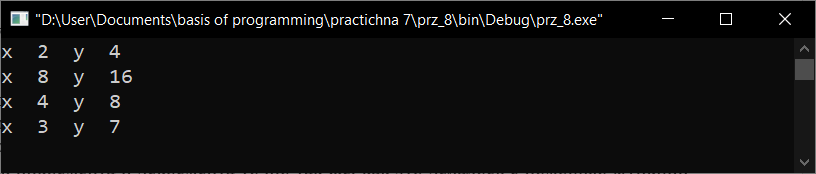
object2.get\_xy(x, y);

cout << "x " << x << " y " << y << endl;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №9 :**

Из материала Unit7 вы должны были уяснить, что с помощью дружественной оператор-функции можно определить разницу между префиксной и постфиксной формами операторов инкремента и декремента точно так же, как это делалось с помощью функций-членов (просто добавляете целый параметр при задании постфиксной версии). Например, здесь приводятся префиксная и постфиксная версии оператора инкремента относительно класса coord :

coord operator ++(coord &obj); // prefix

coord operator ++(coord &obj, int notused); // postfix

Если оператор ++ находится перед операндом, то вызывается функция coord operator +

+(coord &obj). Если оператор ++ находится после операнда, вызывается функция coord operator ++(coord &obj, int notused). В этом случае переменной notused будет передано значение 0. Придумайте свой собственный пример (и класс тоже), демонстрирующий все вышесказанное.

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x;

public:

Coord() { x = 0; }

Coord(int i) { x = i; }

void get\_x(int &i) { i = x ; }

int return\_x(){ return x;}

friend Coord operator ++(Coord &obj); // prefix

friend Coord operator ++(Coord &obj, int notused); // postfix

};

Coord operator++(Coord &obj){

obj.x++;

return obj;

}

Coord operator++(Coord &obj, int notused) {

obj.x++;

return obj;

}

int main()

{

int a, b, c;

cin >> a >> b >> c;

Coord object1(a), object2(b);

int x;

object1.get\_x(x);

cout << " object1 X = " << x << endl;

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

object1++;

cout << object1.return\_x() << ' ';

}

cout << endl;

object1.get\_x(x);

cout << " object1++ X = " << x << endl;

object2.get\_x(x);

cout << " object2 X = " << x << endl;

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

++object2;

cout << object2.return\_x() << ' ';

}

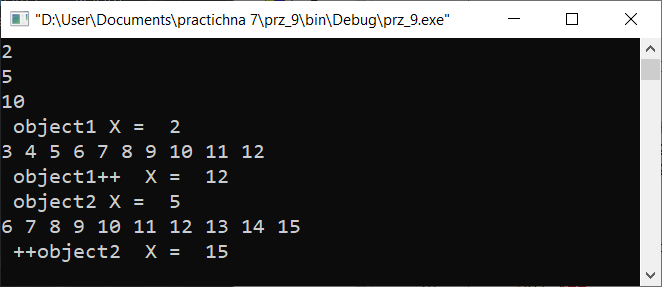
object2.get\_x(x);

cout << "\n ++object2 X = " << x << endl;

return 0;

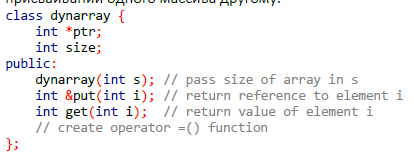
}

**Результати роботи програми (скриншоти)**



**Завдання №10 :**

Пусть дано следующее объявление класса, добавьте все необходимое для создания типа динамический массив. То есть выделите память для массива и сохраните указатель на эту память по адресу рtr. Размер массива в байтах сохраните в переменной size. Создайте функцию put(), возвращающую ссылку на заданный элемент массива и функцию get(), возвращающую значение заданного элемента. Обеспечьте контроль границ массива. Кроме этого перегрузите оператор присваивания так, чтобы выделенная каждому массиву такого типа память не была случайно повреждена при присваивании одного массива другому.



**Код програми:**

#include <iostream>

using namespace std;

class Array {

private:

int \*p;

int size;

public:

Array(int capability) {

p = new int[capability];

if(!p)

exit(1);

size = capability;

}

int &put(int i) {

if( i < 0 || i > size)

exit(1);

return p[i];

}

int get(int i) {

if( i < 0 || i > size)

exit(1);

return p[i];

}

Array &operator= (Array &object) {

if(size != object.size){

cout << "sizes differ";

exit(1);

}

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

p[i] = object.get(i);

return \*this;

}

~Array() { delete[] p;};

};

int main(){

int a, b;

cin >> a >> b;

Array obj1(a), obj2(b);

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

obj1.put(i) = i;

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

cout << obj1.get(i) << ' ';

cout << endl;

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

obj2.put(i) = i + 5;

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

cout << obj2.get(i) << ' ';

cout << endl;

obj1 = obj2;

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

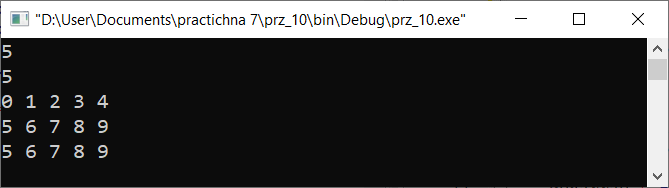
cout << obj1.get(i) << ' ';

cout << endl;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №11 :**

Сравните реализацию strtype, использующую конструктор копий (См. Unit6 "Copy Constructor To Allow Objects To Be Passed To Functions", и реализацию strtype, использующую ссылку в качестве параметра и ссылку в качестве возвращаемого значения функции (См. Unit7 "A Closer Look At The Assignment Operator"). Объясните чем одно из решений лучше другого.

\*-(Для продвинутых) создайте пример, в котором продемонстрируйте ситуацию, в которой конструктор копий окажется более предпочтительным решением.

Подсказка: Как вы узнали из Unit6, создание конструктора копий - это другой путь решения проблем, описанных в Unit7. Конструктор копий может оказаться не столь эффективным решением, как ссылка в качестве параметра + ссылка в качестве возвращаемого значения функции. Это происходит потому, что использование ссылки исключает затраты ресурсов, связанные с копированием объекта в каждом из двух указанных случаев. В C++ часто имеется несколько способов достижения одной и той же цели. Понимание их преимуществ и недостатков есть часть процесса вашего становления как профессионального программиста C++.

**Код програми:**

#include <iostream>

#include <cstring>

using namespace std;

class String {

private:

char \*ptr;

int len;

public:

String(const char \*s);

String(const String &s);

String();

~String(){ delete []ptr; }

char \*get() { return ptr; }

String &operator = (const String &obj);

const char &operator[](int i) const;

void change();

void print(){ cout << ptr << endl; }

};

String::String(){

len = 0;

ptr = new char[1];

ptr[0] = '\0';

}

String::String(const char \*s){

int len = strlen(s) + 1;

ptr = new char[len];

if(!ptr)

exit(1);

this->len = len;

strcpy(ptr, s);

}

String::String(const String &s){

len = s.len;

ptr = new char[len];

if(!ptr) exit(1);

strcpy(ptr, s.ptr);

}

String &String::operator = (const String & obj){

if (this == &obj)

return \*this;

delete [] ptr;

ptr = new char[obj.len + 1];

if(!ptr) exit(1);

len = obj.len;

strcpy(ptr, obj.ptr);

return \*this;

}

const char & String::operator[](int i) const {

return ptr[i];

}

void String::change(){

for(int i = 0; i < len - 1; ++i)

cin >> ptr[i];

}

int main(){

String a("Amsterdam"), c; //Reykjavik

a.print();

a.change();

c = a;

c.print();

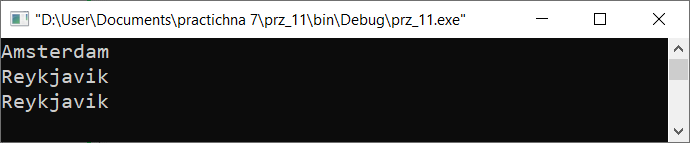
//String d(a); d.print();

return 0;

}

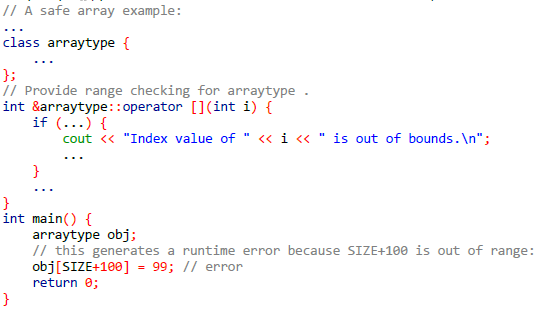
/\*При виклику конструктора копій створюється порозрядна копія елемента, тимчасовий елемент, що призводить до додаткових затрат пам’яті, використання посилання в якості параметра функції дає можливість уникнути цього, адже ми передаємо посилання на цей об’єкт та проводимо дії безпосередньо з ним\*/

**Результати роботи програми (скриншоти):**



**Завдання №12 :**

В Unit5 "Bounded Array" рассматривался простейший способ реализации безопасного массива, в котором для доступа к элементам массива использовались функции get() и put(). Перегрузка оператора [] позволяет создать такой массив гораздо проще. Для создания безопасного массива реализуйте в функции operator[]() контроль границ. Кроме этого, функция operator[]() должна возвращать ссылку на индексируемый элемент. В представленном ниже фрагменте сделана попытка добавить контроль границ массива, что позволяет при нарушении границ генерировать соответствующую ошибку. Подсказка: Вспомните, что безопасный массив - это массив, который инкапсулирован в классе, и при этом класс обеспечивает контроль границ массива. Благодаря перегрузке оператора [] , работать с безопасным массивом можно так же, как с обычным.



**Код програми:**

#include <iostream>

using namespace std;

const int SIZE = 10;

class Array {

private:

int arr[SIZE];

public:

Array(){

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

arr[i] = rand()/(SIZE \* 10 + i) % 16 ;

//arr[i] = i;

}

~Array(){ delete[] arr; }

int &operator[](int i){

if (i < 0 || i > SIZE - 1){

cout << "error" << endl;

exit(1);

}

return arr[i];

}

void print(){

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

cout << arr[i] << ' ';

cout << endl;

}

};

int main() {

Array obj1;

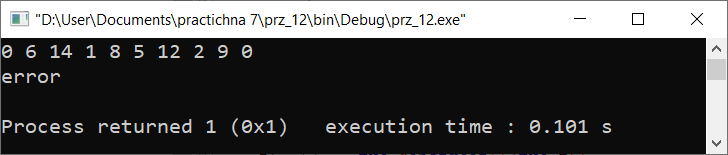
obj1.print();

obj1[SIZE + 5] = 10;

return 0;

}

**Результати роботи програми (скриншоти):**



**Завдання №12 :**

Переделайте Example 7.10 (См. Unit7 "A Closer Look At The Assignment Operator") так, чтобы относительно класса strtype перегрузить оператор [] . Этот оператор должен возвращать символ по заданному индексу. Кроме этого, необходима возможность задавать оператор [] в левой части инструкции присваивания. Покажите, что ваша программа работает.

**Код програми:**

#include <iostream>

#include <cstring>

#include <cstdlib>

using namespace std;

class String {

private:

char \*ptr;

int lenght;

public:

String(char const \*str);

~String() { delete []ptr; }

char \*get() { return ptr; }

String &operator= (const String &object);

char &operator[](int i);

};

String::String (const char \*str){

int len = strlen(str) + 1;

ptr = new char[len];

if(!ptr)

exit(1);

lenght = len;

strcpy(ptr, str);

}

String &String::operator= (const String &object){

if(lenght < object.lenght){

delete[] ptr;

ptr = new char[object.lenght];

if(!ptr)

exit(1);

}

lenght = object.lenght;

strcpy(ptr, object.ptr);

return \*this;

}

char &String::operator[](int i){

if (lenght < 0 || i > lenght - 1){

cout << "error" << endl;

exit(1);

}

return ptr[i];

}

int main()

{

String a("Hello"), b("Goodbye");

cout << a.get() << "\n" << b.get() << "\n";

a = b;

cout << a.get() << "\n" << b.get() << "\n";

b[1] = ' ';

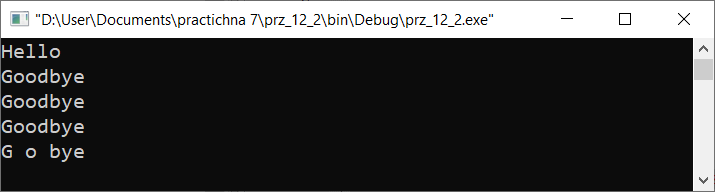
b[3] = ' ';

cout << b.get() << "\n";

return 0;

}

**Результати роботи програми (скриншоти):**



**Завдання №13 :**

Измените ваше решение Задачи 7.12 так, чтобы использовать оператор [] для индексирования динамического массива. То есть замените функции get() и put() оператором [] .

**Код програми:**

#include <iostream>

using namespace std;

class String {

private:

int \*ptr;

int size;

public:

String(int num);

~String() { delete []ptr; }

String &operator= (const String &object);

int &operator[](int i);

void print();

};

String::String (int num){

ptr = new int[num];

if(!ptr)

exit(1);

size = num;

}

String &String::operator= (const String &object){

if(size != object.size){

cout << "sizes deffer" << endl;

exit(1);

}

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

ptr[i] = object.ptr[i];

return \*this;

}

int &String::operator[](int i){

if (i < 0 || i > size)

exit(1);

return ptr[i];

}

int main()

{

int c, d;

cin >> c >> d;

String obj1(c), obj2(d);

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

obj1[i] = i;

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

cout << obj1[i] << ' ';

cout << endl;

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

obj2[i] = i % 8 + rand() % 16;

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

cout << obj2[i] << ' ';

cout << endl;

obj1 = obj2;

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

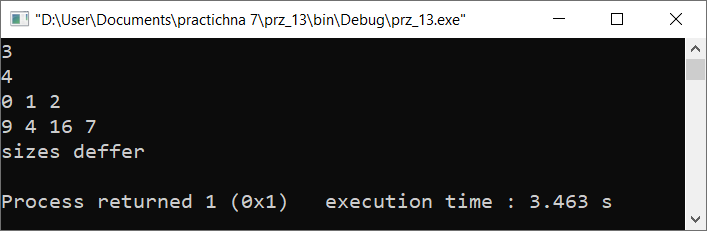
cout << obj1[i] << ' ';

cout << endl;

return 0;

}

**Результати роботи програми (скриншоти):**



**Завдання №14 :**

Перегрузите операторы сдвига ≫ и ≪ относительно класса coord так, чтобы стали возможными следующие типы операций:

obj << integer

obj >> integer

Удостоверьтесь, что ваши операторы действительно сдвигают значения х и у на заданное количество разрядов.

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y;

public:

Coord() { x = 0; y = 0; }

Coord(int i, int j) { x = i; y = j; }

void get\_xy(int &i, int &j) { i = x ; j = y; }

Coord operator<< (int i);

Coord operator>> (int i);

};

Coord Coord::operator<< (int i){

Coord temp;

temp.x = x << i;

temp.y = y << i;

return temp;

}

Coord Coord::operator>> (int i){

Coord temp;

temp.x = x >> i;

temp.y = y >> i;

return temp;

}

int main()

{

Coord object1(2, 3);

int x, y;

object1.get\_xy(x, y);

cout << x << ' ' << y << endl;

object1 = object1 << 4;

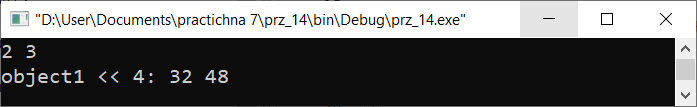
object1.get\_xy(x, y);

cout << "object1 << 4: " << x << ' ' << y << endl;

return 0;

}

**Результати роботи програми (скриншоти):**



**Завдання №15 :**

У вас есть класс:

class three\_d {

int x, y, z;

public:

three\_d(int i, int j, int k) { x=i; y=j; z=k; }

three\_d(){ x=0; y=0; z=0; }

void get(int &i, int &j, int &k) { i=x; j=y; k=z; }

};

Перегрузите для этого класса операторы +, -, ++ и --. Для операторов инкремента и декремента перегрузите только префиксную или только постфиксную форму.

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y, z;

public:

Coord() { x = 0; y = 0; z = 0; }

Coord(int i, int j, int k) { x = i; y = j; z = k;}

void get\_xy(int &i, int &j, int &k) { i = x ; j = y; k = z; }

Coord operator- (Coord obj2);

Coord operator+ (Coord obj2);

Coord operator-- ();

Coord operator++ ();

};

Coord Coord::operator+(Coord obj2){

Coord temp;

temp.x = x + obj2.x;

temp.y = y + obj2.y;

temp.z = z + obj2.z;

return temp;

}

Coord Coord::operator-(Coord obj2){

Coord temp;

temp.x = x - obj2.x;

temp.y = y - obj2.y;

temp.z = z - obj2.z;

return temp;

}

Coord Coord::operator--(){

x--;

y--;

z--;

return \*this;

}

Coord Coord::operator++(){

x++;

y++;

z++;

return \*this;

}

int main()

{

Coord object1(10, 8, 5), object2(2, 3, 4);

int x, y, z;

object1.get\_xy(x, y, z);

cout << "object1: X = " << x << " Y = "<< y << " Z = "<< z <<endl;

object2.get\_xy(x, y, z);

cout << "object2: X = " << x << " Y = "<< y << " Z = "<< z <<endl;

object1 = object2 - object1;

object1.get\_xy(x, y, z);

cout << "object1: X = " << x << " Y = "<< y << " Z = "<< z <<endl;

object1 = object1 + object2;

object1.get\_xy(x, y, z);

cout << "object1: X = " << x << " Y = "<< y << " Z = "<< z <<endl;

--object1;

object1.get\_xy(x, y, z);

cout << "--object1 X = " << x << " Y = "<< y << " Z = "<< z <<endl;

++object2;

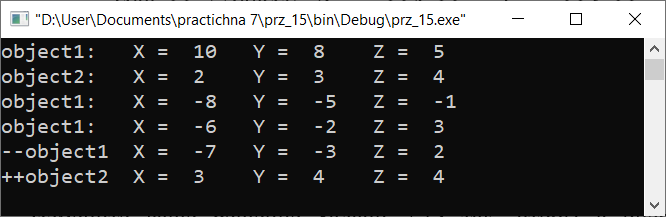
object2.get\_xy(x, y, z);

cout << "++object2 X = " << x << " Y = "<< y << " Z = "<< y <<endl;

return 0;

}

**Результати роботи програми (скриншоти):**



**Завдання №16 :**

Измените ваше решение Задачи 7.15 так, чтобы в оператор-функциях вместо параметров-значений использовать параметры-ссылки.

Подсказка: Для операторов инкремента и декремента потребуются дружественные функции.

Перегрузите операторы ==, !=, || и + относительно класса three\_d Задачи 7.15 так, чтобы

иметь возможность выполнять следующие типы операций:

obj +int;

int + obj;

**Код програми:**

#include <iostream>

using namespace std;

class Coord {

private:

int x, y, z;

public:

Coord() { x = 0; y = 0; z = 0; }

Coord(int i, int j, int k){ x = i; y = j; z = k; }

void get(int &i, int &j, int &k) { i = x; j = y; k = z; }

Coord operator- (Coord obj);

Coord operator+ (Coord obj);

friend Coord operator++ (Coord &obj);

friend Coord operator++ (Coord &obj, int notused);

friend Coord operator-- (Coord &obj);

friend Coord operator-- (Coord &obj, int notused);

friend Coord operator+(Coord &obj, int i);

friend Coord operator+(int i, Coord &obj);

int operator== (Coord obj);

int operator!= (Coord obj);

int operator|| (Coord obj);

};

Coord Coord::operator+(Coord obj){

Coord temp;

temp.x = x + obj.x;

temp.y = y + obj.y;

temp.z = z + obj.z;

return temp;

}

Coord Coord::operator-(Coord obj){

Coord temp;

temp.x = x - obj.x;

temp.y = y - obj.y;

temp.z = z - obj.z;

return temp;

}

Coord operator+(Coord &obj, int i){

Coord temp;

temp.x = obj.x + i;

temp.y = obj.y + i;

temp.z = obj.z + i;

return temp;

}

Coord operator+(int i, Coord &obj){

Coord temp;

temp.x = obj.x + i;

temp.y = obj.y + i;

temp.z = obj.z + i;

return temp;

}

Coord operator++(Coord &obj){

obj.x++;

obj.y++;

obj.z++;

return obj;

}

Coord operator--(Coord &obj, int notused){

obj.x--;

obj.y--;

obj.z--;

return obj;

}

Coord operator--(Coord &obj){

obj.x--;

obj.y--;

obj.z--;

return obj;

}

Coord operator++(Coord &obj, int notused){

obj.x++;

obj.y++;

obj.z++;

return obj;

}

int Coord::operator==(Coord obj){

return x == obj.x && y == obj.y && z == obj.z;

}

int Coord::operator!=(Coord obj){

return x != obj.x && y != obj.y && z != obj.z;

}

int Coord::operator||(Coord obj){

return x || obj.x && y || obj.y && z || obj.z;

}

int main(){

Coord obj1(3, 4, 5), obj2(3, 6, 2), obj3(1, 0, 1);

int x, y, z;

obj1 = obj1 + obj3;

obj1.get(x, y, z);

cout << "obj1 + obj3 x " << x << " y " << y << " z " << z << endl;

obj2 = obj2 - obj3;

obj2.get(x, y, z);

cout << "obj2 - obj3 x " << x << " y " << y << " z " << z << endl;

obj1 = 4 + obj1;

obj1.get(x, y, z);

cout << "4 + obj1 x " << x << " y " << y << " z " << z << endl;

obj1 = obj1 + 2;

obj1.get(x, y, z);

cout << "obj1 + 2 x " << x << " y " << y << " z " << z << endl;

obj2 = obj2 - obj1;

obj2.get(x, y, z);

cout << "obj2 - obj1 x " << x << " y " << y << " z " << z << "\n\n";

obj3++;

obj3.get(x, y, z);

cout << "obj3++ x " << x << " y " << y << " z " << z << endl;

++obj3;

obj3.get(x, y, z);

cout << "++obj3 x " << x << " y " << y << " z " << z << endl;

obj2--;

obj2.get(x, y, z);

cout << "obj2++ x " << x << " y " << y << " z " << z << endl;

--obj2;

obj2.get(x, y, z);

cout << "++obj2 x " << x << " y " << y << " z " << z << "\n\n";

cout << ((obj1 == obj3) ? "obj1 == obj3" : "obj1 != obj3") << endl;

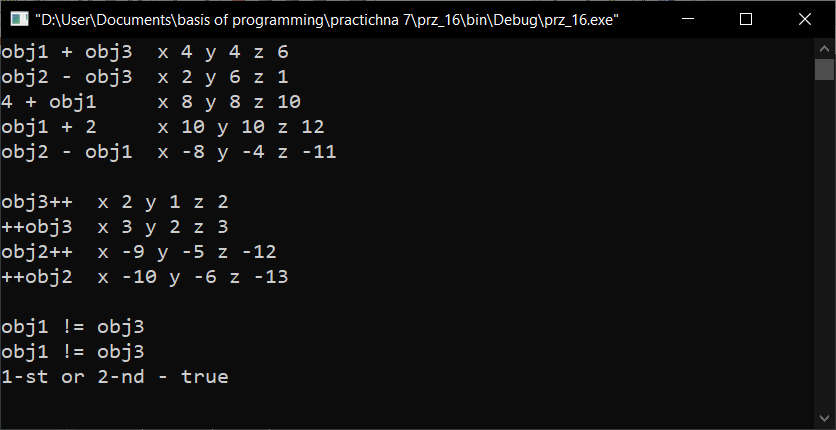
cout << ((obj1 != obj3) ? "obj1 != obj3" : "obj1 == obj3") << endl;

cout << ((obj1 || obj3) ? "1-st or 2-nd - true" : "1-st or 2-nd - false") << endl;

return 0;

}

**Результати роботи програми (скриншоти):**



**Завдання №17 :**

Создайте свой вариант класса strtype, который допускает следующие типы операций:

* Конкатенацию строк с помощью оператора +
* Присваивание строк с помощью оператора =
* Сравнение строк с помощью операторов < > и ==

Разрешено пользоваться строками фиксированной длины. На первый взгляд это может показаться непростой задачей, но, немного подумав и поэкспериментировав, вы обязательно справитесь. ВВ. в вас верит! :)

**Код програми:**

#include <iostream>

#include <cstring>

using namespace std;

class String {

private:

char str[100];

public:

String(const char \*ptr = "\0") { strcpy(str, ptr); }

//char \*get() { return str; }

String operator+ (String obj);

String operator= (String obj);

int operator> (String obj);

int operator< (String obj);

int operator== (String obj);

void print() { cout << str << endl; }

};

String String::operator+(String obj){

String temp;

strcpy(temp.str, str);

strcat(temp.str, obj.str);

return temp;

}

String String::operator= (const String obj){

strcpy(str, obj.str);

return \*this;

}

//

//int String::operator > (const String obj){

// return (strlen(str) > strlen(obj.str)) ? 1 : 0;

//}

//

//int String::operator < (const String obj){

// return (strlen(str) < strlen(obj.str)) ? 1 : 0;

//}

int String::operator > (const String obj){

return strcmp(str, obj.str) > 0;

}

int String::operator < (const String obj){

return strcmp(str, obj.str) < 0;

}

int String::operator== (const String obj){

return (strcmp(str, obj.str)) ? 0 : 1;

}

int main (){

String obj1(" Hello"), obj2(" Good"), obj3;

obj1.print();

obj2.print();

obj3 = obj1 + obj2;

obj3.print();

cout << ((obj1 > obj2) ? "obj1 > obj2" : "obj1 < obj2") << endl;

cout << ((obj1 < obj2) ? "obj1 < obj2" : "obj1 > obj2") << endl;

cout << ((obj1 == obj2) ? "obj1 = obj2" : "obj1 != obj2") << endl;

return 0;

}

**Результати роботи програми (скриншоти):**

