**ПРАКТИЧНЕ ЗАВДАННЯ № 13**

**Завдання №1 :**

В Unit 7, Example 7.1 был создан класс coord для хранения целочисленных координат. Создайте родовую версию этого класса, чтобы можно было хранить координаты любого типа. Продемонстрируйте программу решения этой задачи.

**Код програми:**

#include <iostream>

using namespace std;

template <class T> class Coord {

T x, y;

public:

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

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

void show(){ cout << "Point: (" << x <<';' << ' ' << y << ')' << endl; }

};

int main()

{

Coord a(10,6);

Coord b(3.6, 2.1);

Coord c('f', 'g');

a.show();

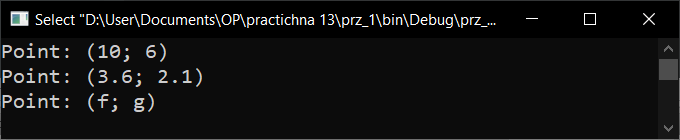
b.show();

c.show();

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №2 :**

Переделайте функцию main() из Example 13.6 так, чтобы для запрещения вывода на экран объектов типа NullShape использовался не оператор typeid, а оператор dynamic\_cast.

**Код програми:**

#include <iostream>

#include <cstdlib>

#include<typeinfo>

using namespace std;

class Shape {

public:

virtual void example() = 0;

};

class Rectangle: public Shape {

public:

void example() { cout << "\*\*\*\*\*\n\* \*\n\* \*\n\*\*\*\*\*\n"; }

};

class Triangle: public Shape {

public:

void example(){ cout << "\*\n\* \*\n\* \*\n\*\*\*\*\*\n"; }

};

class Line: public Shape {

public:

void example(){ cout << "\*\*\*\*\*\n"; }

};

class NullShape: public Shape {

public:

void example(){ }

};

Shape \*generator(){

switch(rand() % 4){

case 0: return new Line;

case 1: return new Rectangle;

case 2: return new Triangle;

case 3: return new NullShape;

}

return NULL;

}

int main() {

Shape \*ptr;

for(int i = 0; i < 10; ++i){

ptr = generator ();

cout << " class " << typeid( \*ptr).name() << endl;

if(!dynamic\_cast<NullShape \*> (ptr))

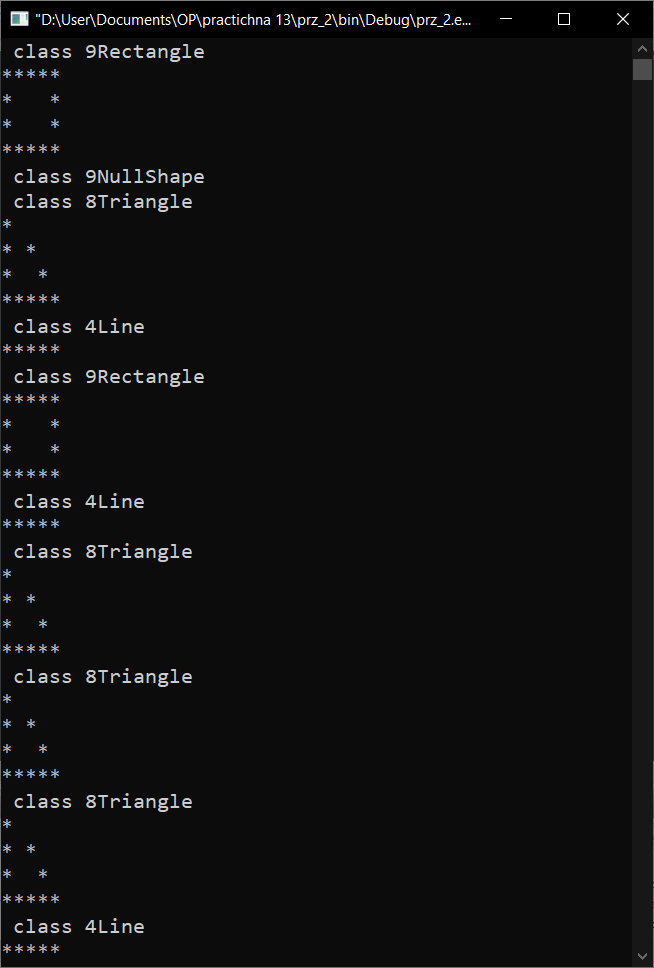
ptr->example();

}

return 0;

}

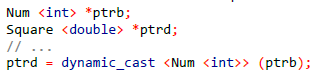
**Результати роботи програми (скриншоти)**



**Завдання №3 :**

Будет ли работоспособен следующий фрагмент программы в иерархии классов с

базовым классом Num из Example 13.9?

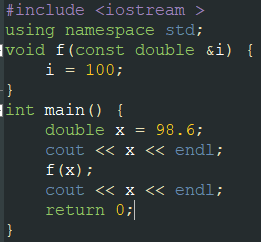
****

**Код програми:**

/\*Код з такою конструкцією працювати не буде, адже вказівники \*ptrb і \*ptrd – є вказівниками різних типів\*/

**Завдання №4 :**

В следующей программе имеется ошибка. Исправьте ее с помощью оператора const\_cast.



Объясните на двух примерах, почему оператор const\_cast следует использовать только в самых крайних случаях.

**A)**

**Код програми:**

#include <iostream >

using namespace std;

void f(const double &i) {

double &a = const\_cast<double &>(i);

a = 100;

}

int main() {

double x = 98.6;

cout << x << endl;

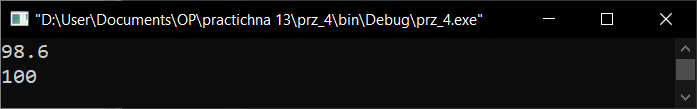
f(x);

cout << x << endl;

return 0;

}

**Результати роботи програми (скриншоти):**



**B)**

**Код програми:**

#include <iostream>

using namespace std;

class Integer {

private:

int number;

public:

void setNumber(int num) {

number = num;

}

void printNumber() const {

cout << number;

const\_cast< Integer \* >( this )->number += 2;

cout << " --> " << number << endl;

}

};

int main() {

Integer X[10];

for(int i = 0; i < 10; ++i)

X[i].setNumber(i);

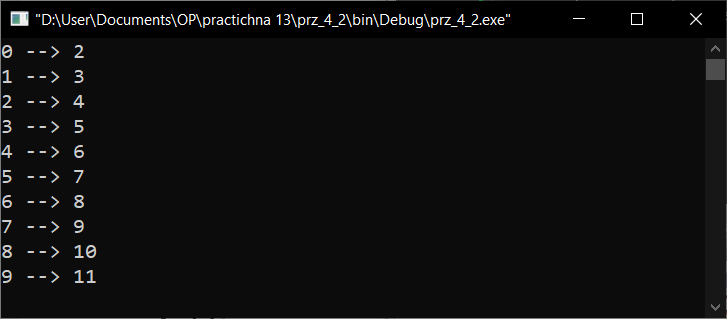
for(int i = 0; i < 10; ++i)

X[i].printNumber();

return 0;

}

**Результати роботи програми (скриншоти):**



**C)**

**Код програми:**

#include <iostream>

#include <cstring>

#define SIZE 3

using namespace std;

class String {

private:

string str;

int mark;

public:

void get() {

getline(cin, str);

}

void get\_v() {

cin >> mark;

}

void show() const {

cout << str << ' ' << mark << endl;

const\_cast<String \*>(this)->str = str + "'s ";

cout << str << "mark is " << mark << endl;

}

};

int main() {

String x[SIZE];

cout << "Enter students name: " << endl;

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

cout << "Student #" << i + 1 << endl;

x[i].get();

}

cout << "Enter students marks: "<< endl;

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

cout << "Student #" << i + 1 << endl;

x[i].get\_v();

}

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

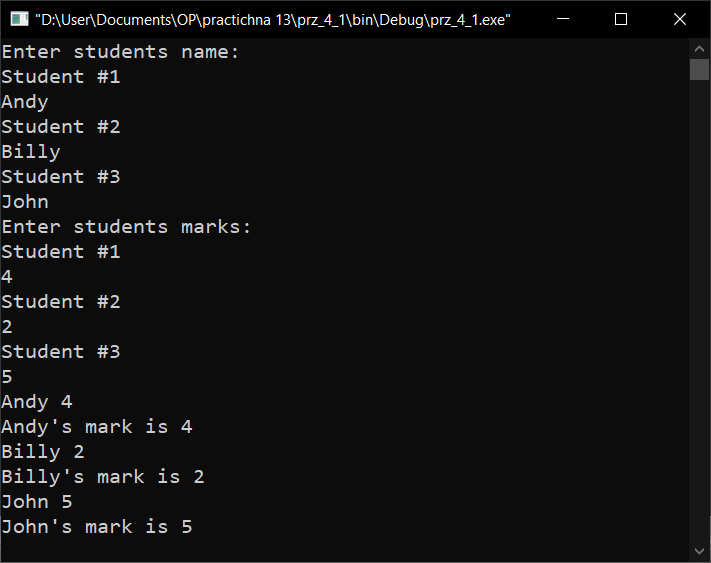
x[i].show();

return 0;

}

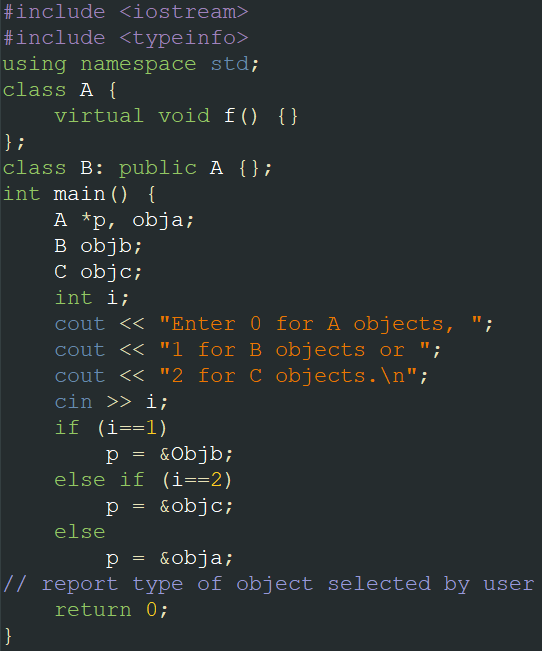
/\*Оператор const\_cast змінює статус специфікаторів const і volatile, але найчастіше використовується для зняття const, тому треба бути обережним при його використані, щоб не виникло необажаних змін в об’єктах.\*/

**Результати роботи програми (скриншоти):**



**Завдання №5 :**

Допишите следующую программу, чтобы на экран выводилась информация о выбранном пользователем типе объекта.



**Код програми:**

#include <iostream>

#include <typeinfo>

using namespace std;

class A {

virtual void f() {}

};

class B: public A {};

class C: public A {};

int main() {

A \*p, obja;

B objb;

C objc;

int i;

cout << "Enter 0 for A objects, ";

cout << "1 for B objects or ";

cout << "2 for C objects.\n";

cin >> i;

if (i == 1)

p = &objb;

else if (i == 2)

p = &objc;

else

p = &obja;

if(typeid( \*p) == typeid(A))

cout << "object type -- A" << endl;

else if(typeid( \*p) == typeid(B))

cout << "object type -- B" << endl;

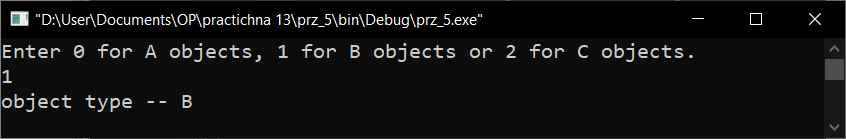
else

cout << "object type -- C" << endl;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №6 :**

a) Переделайте программу Example 13.3a, чтобы возможную ошибку выделения памяти внутри функции generator() отслеживать с помощью механизма обработки исключительных ситуаций. б) Измените функцию generator() из вопроса а), чтобы в ней использовать версию оператора new с ключевым словом nothrow.

**A)**

**Код програми:**

#include <iostream>

#include <typeinfo>

#include <cmath>

#include <cstdlib>

#include <new>

using namespace std;

template <class T> class Num {

public:

T x;

Num(T i) {

x = i;

}

virtual T get\_value() {

return x;

}

};

template <class T> class Square: public Num <T> {

public:

Square(T i): Num <T> (i) {}

T get\_value() {

return this->x \* this->x;

}

};

template <class T> class Sqr\_root: public Num <T> {

public:

Sqr\_root(T i): Num <T> (i) { }

T get\_value() {

return sqrt(double(this->x));

}

};

Num <double> \*generator () {

try {

switch(rand() % 2) {

case 0:

return new Square<double> (rand() % 100);

case 1:

return new Sqr\_root<double> (rand() % 100);

}

} catch (bad\_alloc &ex) {

cerr << "failure allocating memory: " << ex.what() << endl;

return NULL;

}

return NULL;

}

int main() {

Num <double> obj1(10), \*ptr1;

Square <double> obj2(100.0);

Sqr\_root<double> obj3(999.2);

cout << typeid(obj1).name() << endl;

cout << typeid(obj2).name() << endl;

cout << typeid(obj3).name() << endl;

if(typeid(obj2) == typeid(Square<double>))

cout << "Is Square <double> " << endl;

ptr1 = &obj2;

if(typeid( \*ptr1) != typeid(obj1))

cout << "Value is " << ptr1->get\_value() << endl << endl;

cout << "Now, generate some objects " << endl;

for(int i = 0; i < 10; ++i) {

ptr1 = generator();

if(typeid( \*ptr1) == typeid(Square<double>))

cout << "Square object: ";

if (typeid( \*ptr1) == typeid(Sqr\_root<double>))

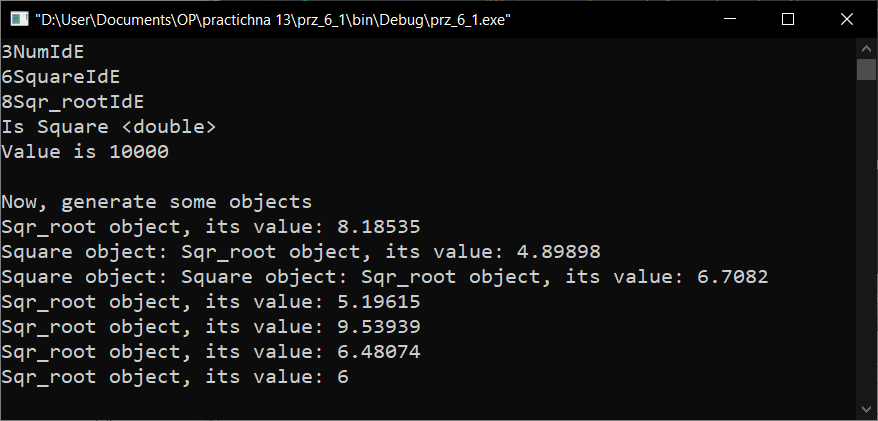
cout << "Sqr\_root object, its value: " << ptr1->get\_value() << endl;

}

return 0;

}

**Результати роботи програми (скриншоти)**



**B)**

**Код програми:**

#include <iostream>

#include <typeinfo>

#include <cmath>

#include <cstdlib>

#include <new>

using namespace std;

template <class T> class Num {

public:

T x;

Num(T i) {

x = i;

}

virtual T get\_value() {

return x;

}

};

template <class T> class Square: public Num <T> {

public:

Square(T i): Num <T> (i) {}

T get\_value() {

return this->x \* this->x;

}

};

template <class T> class Sqr\_root: public Num <T> {

public:

Sqr\_root(T i): Num <T> (i) { }

T get\_value() {

return sqrt(double(this->x));

}

};

Num <double> \*generator () {

switch(rand() % 2) {

case 0:

return new(nothrow) Square<double> (rand() % 100);

case 1:

return new(nothrow) Sqr\_root<double> (rand() % 100);

}

return NULL;

}

int main() {

Num <double> obj1(10), \*ptr1;

Square <double> obj2(100.0);

Sqr\_root<double> obj3(999.2);

cout << typeid(obj1).name() << endl;

cout << typeid(obj2).name() << endl;

cout << typeid(obj3).name() << endl;

if(typeid(obj2) == typeid(Square<double>))

cout << "Is Square <double> " << endl;

ptr1 = &obj2;

if(typeid( \*ptr1) != typeid(obj1))

cout << "Value is " << ptr1->get\_value() << endl << endl;

cout << "Now, generate some objects " << endl;

for(int i = 0; i < 10; ++i) {

ptr1 = generator();

if(typeid( \*ptr1) == typeid(Square<double>))

cout << "Square object: ";

if (typeid( \*ptr1) == typeid(Sqr\_root<double>))

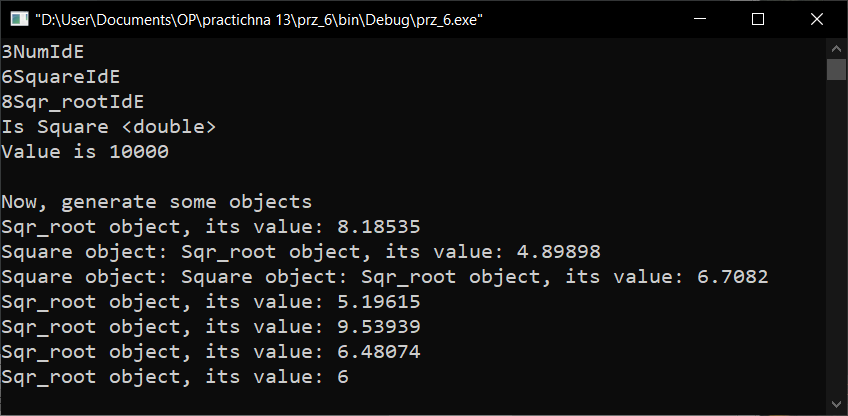
cout << "Sqr\_root object, its value: " << ptr1->get\_value() << endl;

}

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №7 :**

Создайте иерархию классов с абстрактным классом DataStruct на ее вершине (См.  материал Unin 11). В основании иерархии создайте два производных класса. В одном должен быть реализован стек, в другом - очередь. Создайте также функцию DataStructFactory() со следующим прототипом: DataStruct \*DataStructFactory(char what); Функция DataStructFactory() должна создавать стек, если параметр what равен s, и очередь, если параметр what равен q. Возвращаемым значением функции должен быть указатель на созданный объект. Продемонстрируйте работоспособность вашей программы. Совет. Попробуйте поэкспериментировать с RTTI. Хотя польза от динамической идентификации типа в контексте приведенных в Unit 13 простых примеров может показаться не слишком очевидной, тем не менее это мощный инструмент управления объектами во время работы программы.

**Код програми:**

#include <iostream>

#include <cstdlib>

#include <ctime>

#define SIZE 10

using namespace std;

class DataStruct {

public:

DataStruct \*head, \*tail, \*next;

int num;

DataStruct () {

head = tail = next = NULL;

}

virtual void push(int i) = 0;

virtual int pop() = 0;

virtual void create() = 0;

virtual void display() = 0;

};

class Stack: public DataStruct {

public:

void push(int i) {

DataStruct \*temp = new Stack;

if(!temp)

return;

temp->num = i;

if(head)

temp->next = head;

head = temp;

if(!tail)

tail = head;

}

int pop() {

DataStruct \*ptr;

if(!head)

return 0;

int i = head->num;

ptr = head;

head = head->next;

delete ptr;

return i;

}

void create() {

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

push(i);

}

void display() {

cout << "Your Stack" << endl;

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

cout << pop() << ' ';

cout << endl;

}

};

class Queue : public DataStruct {

public:

void push(int i) {

DataStruct \*temp = new Queue;

if(!temp)

return;

temp->num = i;

if(tail)

tail->next = temp;

tail = temp;

temp->next = NULL;

if(!head)

head = tail;

}

int pop() {

DataStruct \*ptr;

if(!head)

return 0;

int i = head->num;

ptr = head;

head = head->next;

delete ptr;

return i;

}

void create() {

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

push(i);

}

void display() {

cout << "Your Queue" << endl;

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

cout << pop() << ' ';

cout << endl;

}

};

DataStruct \*DataStructFactory(char what) {

if(what == 's')

return new Stack;

else if(what == 'q')

return new Queue;

else if(what == 'e') {

cout << "Bye" << endl;

exit(0);

} else {

cout << "incorrect choise" << endl;

cin >> what;

return DataStructFactory(what);

}

}

int main() {

srand(time(NULL));

int a; char b;

cout << "Enter 's' to create Stack\n 'q' - create Queue\n 'e' - exit" << endl;

cin >> b;

DataStruct \*ptr = DataStructFactory(b);

if(!ptr) exit (1);

ptr->create();

ptr->display();

return 0;

}

**Результати роботи програми (скриншоти)**

