**ПРАКТИЧНЕ ЗАВДАННЯ № 9**

**Завдання №1 :**

 Переделайте представленную ниже программу так, чтобы в ней отсутствовала инструкция using namespace std.

**Код програми:**

#include <iostream>

#include <fstream>

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

if (argc !=3) {

std::cout << " Usage : CONVERT <input> <output>\n";

return 1;

}

std::ifstream fin(argv[1]);

std::ofstream fout(argv[2]);

if (!fout) {

std::cout << "Cannot open output file.\n";

return 1;

}

if (!fin) {

std::cout << "Cannot open input file.\n";

return 1;

}

char ch;

fin.unsetf(std::ios::skipws); // do not skip spaces

while (!fin.eof()) {

fin >> ch;

if (ch == ' ') ch = '|';

if (!fin .eof()) fout << ch;

}

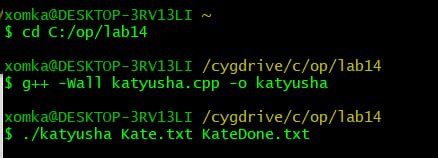
fin.close();

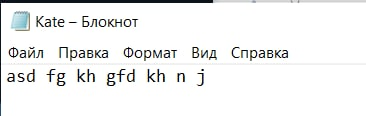
fout.close();

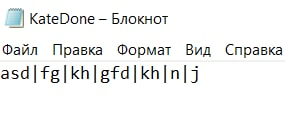
return 0;

}

**Результати роботи програми (скриншоти)**







**Завдання №2 :**

Используя класс strtype из Example 14.5, создайте функцию преобразования для превращения строки в целое. Функция преобразования должна возвращать длину строки, хранящейся в str. Покажите, что ваша функция преобразования работает.

**Код програми:**

#include <iostream>

#include <cstring>

using namespace std;

class Strtype {

char str[100];

int len;

public:

Strtype(const char \*s) {

strcpy(str, s);

len = strlen(s);

}

operator char \*() {

return str;

}

operator int (){

return len;

}

};

int main() {

Strtype s("This is a test");

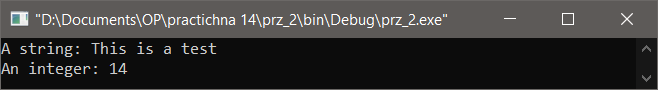
cout << "A string: " << (char \*)s << endl;

cout << "An integer: " << (int)s << endl;

return 0;

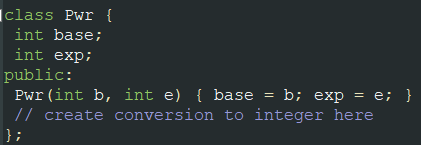
}

**Результати роботи програми (скриншоти)**



**Завдання №3 :**

У вас есть класс:



Создайте функцию преобразования для превращения объекта типа Pwr в целое. Функция должна возвращать результат возведения в степень base^exp.

**Код програми:**

#include <iostream>

#include <cmath>

using namespace std;

class Pwr {

int base;

int exp;

public:

Pwr() { base = 0; exp = 0; }

Pwr(int b, int e) {

base = b;

exp = e;

}

void show() { cout << "base: " << base << ", exponent: " << exp << endl; }

operator int() { return pow(base, exp); }

};

int main() {

int a, b;

cout << "Enter data" << endl;

cin >> a >> b;

Pwr obj(a, b);

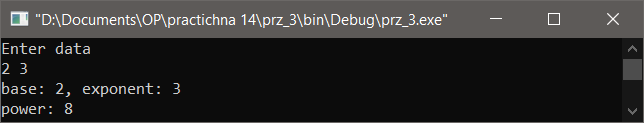
obj.show();

cout << "power: " << (int)obj << endl;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №4 :**

Переделайте Example 14.8 так, чтобы на экране отображался тот объект, который осуществляет вывод символов, и тот объект или те объекты, для которых из-за занятости буфера вывод запрещен.

**Код програми:**

#include <iostream>

#include <cstring>

using namespace std;

class Output {

static char outbuf[255];

static int inuse;

static int oindex;

char str[80];

int i;

int who;

public:

Output(int w, const char \*s) {

strcpy(str, s);

i = 0;

who = w;

}

int putbuf() {

if(!str[i]) {

inuse = 0;

return 0;

}

if(!inuse) {

cout << "Who " << who << endl;

inuse = who;

}

if(inuse != who)

return -1;

if(str[i]) {

outbuf[oindex] = str[i];

++i;

++oindex;

outbuf[oindex] = '\0';

return 1;

}

return 0;

}

void show() {

cout << outbuf << endl;

}

};

char Output::outbuf[255];

int Output::inuse = 0;

int Output::oindex = 0;

int main() {

Output obj1(1, "This is test"), obj2(2, " of statics");

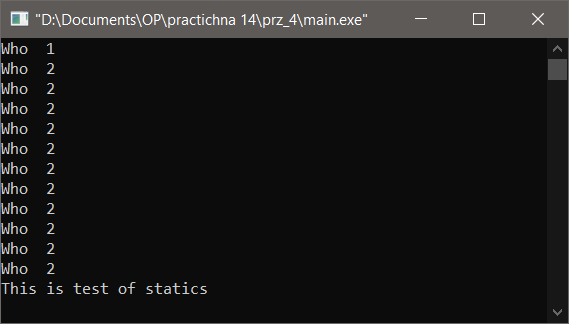
while(obj1.putbuf() | obj2.putbuf());

obj1.show();

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №5 :**

Одним из интересных применений статических переменных-членов является хранение информации о количестве объектов класса, существующих в каждый конкретный момент времени. Для этого необходимо увеличивать на единицу статическую переменнуючлен каждый раз, когда вызывается конструктор класса, и уменьшать на единицу, когда вызывается деструктор. Реализуйте эту схему и продемонстрируйте ее работу.

**Код програми:**

#include <iostream>

using namespace std;

class Objects {

int index;

static int count;

public:

Objects() {

index = ++count;

cout << "Created object #" << index << endl;

}

~Objects() {

index = --count;

cout << "Deleting, objects now: " << index << endl;

}

int get() const { return index; }

};

int Objects::count = 0;

int main() {

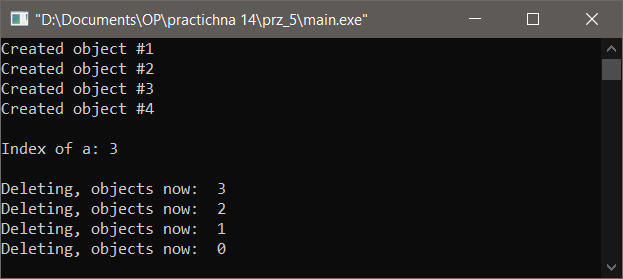
Objects a, b, c, d;

cout << "\nIndex of a: " << c.get() << "\n\n";

return 0;

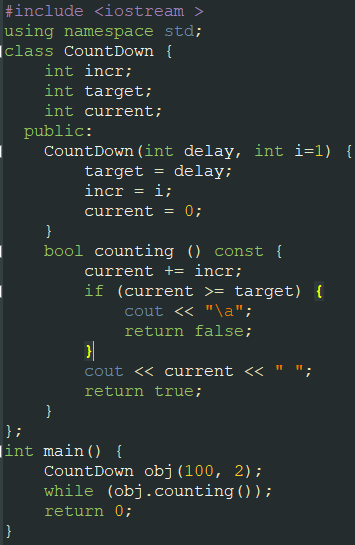
}

**Результати роботи програми (скриншоти)**



**Завдання №6 :**

В следующей программе сделана попытка создать простой таймер для измерения временных интервалов. По истечении каждого такого интервала таймер должен подавать звуковой сигнал. В том виде, в котором эта программа здесь представлена, она компилироваться не будет. Найдите и исправьте ошибку.



**Код програми:**

#include <iostream>

#include "windows.h"

using namespace std;

class CountDown {

int incr;

int target;

int current;

public:

CountDown(int delay, int i = 1) {

target = delay;

incr = i;

current = 0;

}

bool counting () {

current += incr;

if (current >= target) {

Sleep(1000);

cout << "\a";

return false;

}

cout << current << " ";

return true;

}

};

int main() {

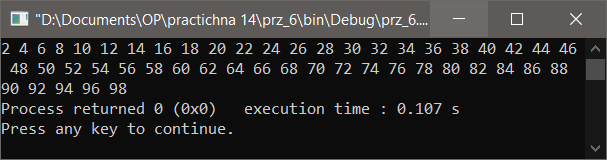
CountDown obj(100, 2);

while (obj.counting());

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №7 :**

В примере из Unit 14, в котором:

explicit myclass(int x) { a = x; }

explicit myclass(const char \*str) { a = atoi(str); }

если спецификатор explicit указать только для конструктора myclass(int) , можно ли будет выполнить неявное преобразование также и для конструктора myclass(const char \*)? Попробуйте и посмотрите, что получится. Попробуйте убрать const из myclass(const char \*) . Посмотрите и проанализируйте то, что вам сообщит компилятор.

**Код програми:**

#include <iostream>

#include <cstdlib>

using namespace std;

class myclass {

int a;

public:

//explicit myclass(int x) { a = x; };

myclass(const char \*str) { a = atoi(str); }

int get() { return a; }

};

int main() {

//myclass obj1 = 4;

myclass obj2 = "123";

//cout << "obj1: " << obj1.get() << endl;

cout << "obj2: " << obj2.get() << endl;

return 0;

}

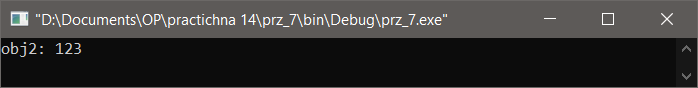
/\*

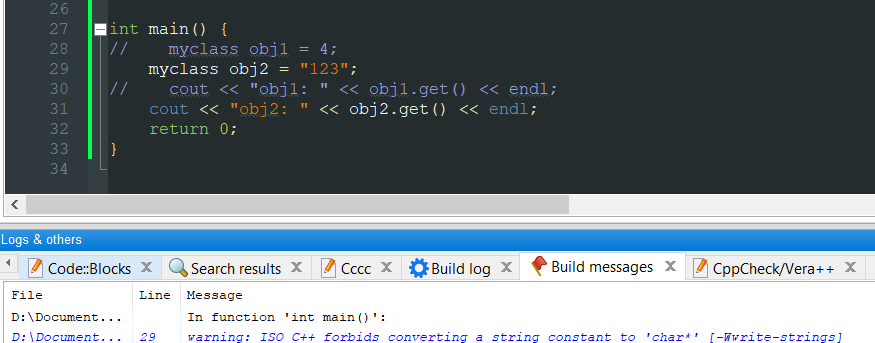
Так, неявне конвертування буде вібуватись, адже конструктор myclass(const char \*str) { a = atoi(str); } оголошено як неявний.

Видає попередження про заборону пертворення: forbids converting const string to char\*, тим самим виконується неявне перетворення, що означає, що літерал може бути зміненим, а це в свою чергу може призвести до undefined behaviour

\*/

**Результати роботи програми (скриншоти)**





**Завдання №8 :**

Попытайтесь на конкретных примерах оправдать введение ключевого слова explicit. Другими словами, объясните, почему неявное преобразование конструкторов в некоторых случаях может оказаться нежелательным.

**Код програми:**

#include <iostream>

#include <cstdlib>

#include <cstring>

using namespace std;

class myclass {

int a;

public:

explicit myclass(int x) {

a = x;

}

void show() {

cout << a << endl;

}

};

int main() {

myclass obj1(17);

myclass obj2(43.98);

myclass obj3('a');

// myclass obj4 = 'c'; //помилка

obj1.show();

obj2.show();

obj3.show();

//obj4.show();

return 0;

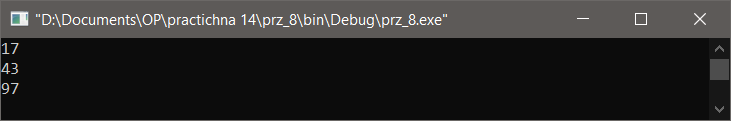
}

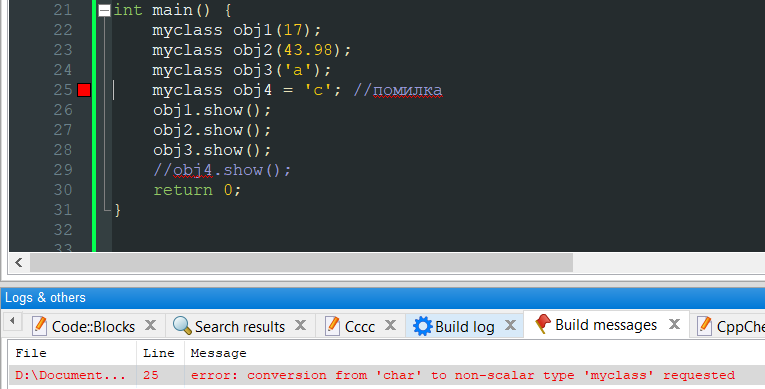
/\*

Для конструкторів з одним аргументом, або вдля таких, що містять перший заданий, а інші за зомовчваням існує дві фоорми задання obj(element) & obj = element, при другій формі відбувається автоматичне, неявне перетворення, яке буває небажаним і може призвести до помилок в програмі, для їх уникнення необхідно зробити конструктор явним, дописавши перед ниим слово explicit, таким чином неявны конвертацыъ не будуть виконуватись ы допустимою формою буде лише obj(element)

\*/

**Результати роботи програми (скриншоти)**



****

**Завдання №9 :**

Используя массивы в качестве объектов ввода/вывода, напишите программу для копирования содержимого одного массива в другой, хотя мы с вами уже знаем, что это не самый эффективный способ решения подобной задачи.

**Код програми:**

#include <iostream>

#include <cstring>

#include <strstream>

using namespace std;

int main() {

const char \*array = "Hello! How are things?";

char \*dest;

ostrstream in(dest, strlen(array) + 1);

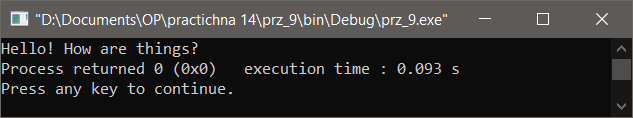
in << array;

cout << array;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №10 :**

Используя массивы в качестве объектов ввода/вывода, напишите программу для преобразования строки, содержащей значение с плавающей точкой , в число с плавающей точкой (double).

**Код програми:**

#include <iostream>

#include <cstring>

#include <strstream>

using namespace std;

int main() {

double value;

const char\* str = "234.234 Hello world!";

istrstream in(str, strlen(str));

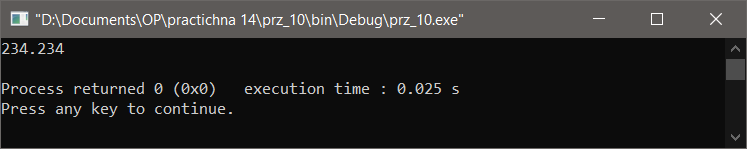
in >> value;

cout << value << endl;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №11 :**

Поскольку для конструктора с одним аргументом преобразование типа этого аргумента в тип класса, в котором определен конструктор, происходит автоматически, исчезает ли в этой ситуации необходимость в использовании перегруженного оператора присваивания? Ответ продемонстрируйте на примере.

**Код програми:**

#include <iostream>

#include <string>

using namespace std;

class A {

int value;

public:

A(int a) { value = a; }

// A &operator =(int a) {

// value = a;

// return \*this;

// }

operator int() { return value; }

A(const string str) { value = stoi(str); }

};

int main() {

const string str = "12348";

A a = 10;

cout << "first: " << a;

a = str;

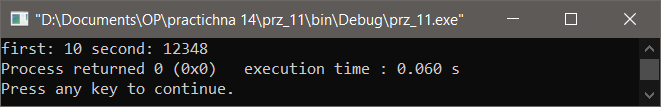
cout << " second: " << a;

return 0;

}

/\*Так, відпадає потреба у використанні перезавантаженого оператора присвоєння, адже дії, виконуванні ним відбуваються автоматично: компілятор автоматично створює “відкритий” оператор присвоєнгня\*/

**Результати роботи програми (скриншоти):**



**Завдання №12 :**

Можно ли в постоянной функции-члене использовать оператор const\_cast, чтобы разрешить этой функции-члену модифицировать вызвавший ее объект? Ответ продемонстрируйте на примере.

**Код програми:**

#include <iostream>

using namespace std;

class Area {

double value;

public:

Area(double a) : value(a){ }

void show(){ cout << "Your area: " << value \* value << endl; }

void change(double a) const {

const\_cast<Area \*>(this)->value = a;

}

};

int main() {

Area obj = 1.2;

obj.show();

obj.change(23.5);

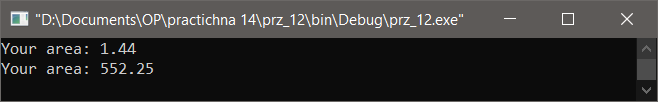
obj.show();

return 0;

}

/\*Так, можна. Таким чином надається можливість модифікування переданих даних у константній функцї\*/

**Результати роботи програми (скриншоти):**



**Завдання №13 :**

Вопрос для размышления: поскольку библиотека исходного C++ содержится в глобальном пространстве имен и для старых программ на C++ это уже свершившийся факт, какая польза от размещения указанной библиотеки в пространстве имен std "задним числом"?

**Код програми:**

**/\***

Сукупності стандартних бібліотек, або користувацьких, можна розміщувати в просторі імен std для використання з різними кодами, або частинами коду, підключаючи леше потрібні бібліотеки та користувацькі функції, змінні, що дає можливість уникнути конфліктів з назвами власних ідентифікаторів та заданих в бібліотеках в цих частинах коду, використовувати функції з однією назвою, але ріним призначенням, також часткова оптимізація коду.

**\*/**

**Завдання №14 :**

Вопрос для размышления: вернитесь к примерам первых 13-ти лекций. Подумайте о том, в каких из них функции-члены можно было бы сделать постоянными или статическими. Может быть это примеры, в которых определение пространств имен наиболее предпочтительно? Пары примеров будет достаточно.

**А)**

**Код програми:**

#include <iostream>

using namespace std;

template <class Type1, class Type2> class Myclass {

Type1 i; Type2 j;

public:

Myclass(const Type1 a, const Type2 b){ i = a; j = b; }

***void show() const*** *{ cout << i << ' ' << j << "\n"; }*

};

int main()

{

Myclass<int, double> obj1(10, 0.23);

Myclass<char, const char \*> obj2('X', "this is a test");

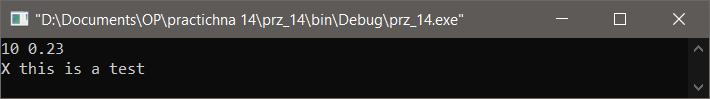
obj1.show();

obj2.show();

return 0;

}

**Результати роботи програми (скриншоти)**



**В)**

**Код програми:**

#include <iostream>

#define \_\_SIZE\_STACK 10

using namespace std;

class Stack {

char stack[\_\_SIZE\_STACK];

int index;

char character;

public:

Stack(const char ch) {

index = 0;

character = ch;

}

void push(const char ch);

void create(){

for(int i = 0; i < \_\_SIZE\_STACK; ++i)

push(rand() % 26 + 'a');

}

const char pop();

void show();

};

void Stack::push(const char ch) {

if(index == \_\_SIZE\_STACK)

return;

stack[index++] = ch;

}

***const char Stack::pop()*** *{*

*if(index == 0)*

*return 0;*

*return stack[--index];*

*}*

void Stack::show() {

for(int i = 0; i < \_\_SIZE\_STACK; ++i)

cout << pop() << ' ';

cout << endl;

}

int main() {

Stack s1('A'), s2('B');

s1.create();

s2.create();

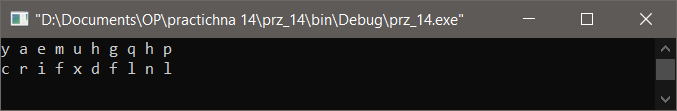
s1.show();

s2.show();

return 0;

}

**Результати роботи програми (скриншоти)**



**С)**

**Код програми:**

#include <iostream>

using namespace std;

class Objects {

int a;

static int count;

public:

Objects(int n) : a(n) { cout << "Created object: " << a << " #" << ++count << endl; }

~Objects() { --count; }

***static int get()*** *{ return count; }*

};

int Objects::count = 0;

int main() {

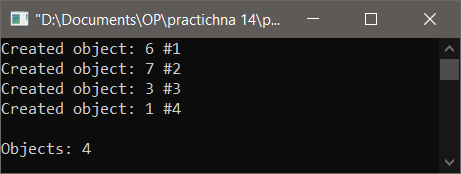
Objects a(6), b(7), c(3), d(1);

cout << "\nObjects: " << Objects::get() << "\n\n";

return 0;

}

**Результати роботи програми (скриншоти)**



/\*Найчастіше використання const функціями-членами є ті функції, які демонструють результат, виводять значення або повертають його, адже при цьому вони не змінюють його

Статичні функції використовуються, коли нам необхідна функція незалежна від жодного об’єкту (тобто викливається так: ClassName::function() ) і яка буде мати доступ до статичних членів, методів класу. Найчастіше використовується для визначення, демонстрації створення та видалення об’Єєктів класу\*/