Звіт

Автор: Чугунов В.Ю

КІТ-119а

Лабораторна робота №15

**РОЗУМНІ ПОКАЖЧИКИ**

**Мета:** по результатах практичної роботи порівняти розумні покажчики бібліотеки STL.

1 ЗАВДАННЯ ДО РОБОТИ

**Індивідуальне завдання.** Порівняти розумні покажчики:

* *auto\_ptr;*
* *unique\_ptr;*
* *shared\_ptr;*
* *weak\_ptr;*

2 РОЗРОБКА АЛГОРИТМІВ ФУНКЦІЙ

**Структура Unit**

**Поля структури:**

shared\_ptr<Unit> target – іменований покажчик типу shared

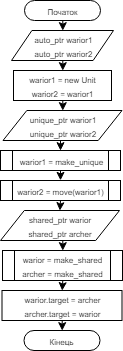
weak\_ptr<Unit> target – іменований покажчик типу weak

Примітка: у роботі використовується тільки один з покажчиків.

**Функції:**

**int main( )**  
 Призначення: Головна функція.  
 Опис роботи:   
 1) створення двох об’єктів класу auto\_ptr. Основна проблема покажчика такого типу полягає в правах володіння. Об’єкт такого класу втрачає права володіння ресурсом при копіюванні (присвоюванні, використанні в конструкторі копій);  
 2) видалення двох об’єктів класу auto\_ptr;  
 3) створення двох об’єктів класу unique\_ptr. Вирішенням проблеми auto\_ptr стала заборона копіювання. А щоб передати права необхідно використати функцію std::move;  
 4) видалення двох об’єктів класу unique\_ptr;  
 5) створення двох об’єктів класу shared\_ptr. Для того, щоб реалізувати права володіння одним об’єктом декількома покажчиками, було введено поле «Кількість сильних посилань». Коли два покажчика типу shared\_ptr вказують на одну і ту ж саму область пам’яті, кількість їх «сильних посилань» дорівнює 2. Але є ситуації, коли виникають циклічні залежності. При використанні shared\_ptr в таких ситуаціях може статися витік пам’яті. Щоб нейтралізувати такі проблеми, використовують weak\_ptr;  
 6) видалення двох об’єктів класу класу shared\_ptr.  
 Схема алгоритму подана на рис. 1.

**Схеми алгоритмів функцій.**

  
Рисунок 1 – Схема алгоритму функції main

3 ТЕКСТ ПРОГРАМИ

**Текст файлу main.cpp**

#include <iostream>

#include <memory>

#include <string>

using std::cout;

using std::cin;

using std::endl;

using std::string;

using std::auto\_ptr;

using std::unique\_ptr;

using std::shared\_ptr;

using std::weak\_ptr;

struct Unit

{

Unit(string name) { cout << "Unit()" << endl; };

~Unit() { cout << "~Unit()" << endl; };

shared\_ptr<Unit> target;

//weak\_ptr<Unit> target;

};

int main()

{

{

auto\_ptr<Unit> warior1(new Unit("warior"));

auto\_ptr<Unit> warior2(warior1);

}

system("cls");

{

unique\_ptr<Unit> warior1 = std::make\_unique<Unit>("warior");

/\*unique\_ptr<int> warior2(warior1);\*/

unique\_ptr<Unit> warior3;

warior3 = std::move(warior1);

}

system("cls");

{

shared\_ptr<Unit> warior1 = std::make\_shared<Unit>("warior");

shared\_ptr<Unit> warior2(warior1);

}

system("cls");

{

shared\_ptr<Unit> warior = std::make\_shared<Unit>("warior");

shared\_ptr<Unit> archer = std::make\_shared<Unit>("archer");

cout << warior.use\_count() << " : " << warior->target.use\_count() << endl;

cout << archer.use\_count() << " : " << archer->target.use\_count() << endl << endl;

warior->target = archer;

cout << warior.use\_count() << " : " << warior->target.use\_count() << endl;

cout << archer.use\_count() << " : " << archer->target.use\_count() << endl << endl;

archer->target = warior;

cout << warior.use\_count() << " : " << warior->target.use\_count() << endl;

cout << archer.use\_count() << " : " << archer->target.use\_count() << endl << endl;

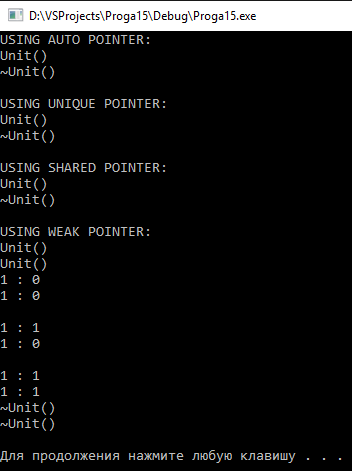
}

system("pause");

return 0;

}

4 РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ

  
Рисунок 2 – Результат роботи програми

ВИСНОВКИ

При виконанні лабораторної роботи набуто практичних навичок щодо розробки програм із використанням розумних покажчиків. Досліджено механізми їх роботи. Виявлено недоліки і переваги кожного з них.