Автор: Білий Вадим, КІТ-119а

Дата: 28.05.2020

Лабораторна робота 15. РОЗУМНІ ВКАЗІВНИКИ

Тема. Розумні вказівники.

Мета – по результатах практичної роботи порівняти розумні вказівники бібліотеки STL.

Загальне завдання

Створити STL-контейнер, що містить у собі об’єкти ієрархії класів, використати розумні вказівники:

- auto\_ptr;

- unique\_ptr;

- shared\_ptr;

- weak\_ptr.

Опис класів

Клас ргз: C\_Rgz

Клас наслідник ргз: C\_RgzM

Опис змінних

string object; - назва об’єкту

int mark; - оцінка

vector<C\_Rgz\*> vect; - вектор

list <C\_Rgz\*> lis; - список

map <int,C\_Rgz\*> mp; - дерево (ключ, данні)

set <C\_Rgz\*> st; - дерево(ключ)

Опис методів

virtual void setObject(const string str); -сетер

virtual string getObject() const; - гетер

virtual string getString() const; - повертає строку з даними

virtual void input(istream& a); - ввід

friend ostream& operator<< (ostream& output, C\_Rgz& obj); - перевантаження <<

virtual bool operator==(C\_Rgz& obj); - перевантаження ==

virtual C\_Rgz& operator= (C\_Rgz& temp); - перевантаження =

friend istream& operator>> (istream& input, C\_Rgz& obj); - перевантаження>>

void setMark(const int a); - сетер

int getMark()const; - гетер

virtual void input(istream& a); -ввід

virtual bool operator==(C\_RgzM& obj); - перевантаження ==

virtual C\_Rgz& operator= (C\_RgzM& temp); - перевантаження =

string getString() const override; - повертає строку з даними

Текст програми

C\_Rgz.cpp

#include "C\_Rgz.h"

void C\_Rgz::setObject(const string str)

{

object = str;

}

string C\_Rgz::getObject() const

{

return object;

}

string C\_Rgz::getString() const

{

return object;

}

void C\_Rgz::input(istream& a)

{

a >> object;

}

bool C\_Rgz::operator>(C\_Rgz& obj)

{

return getString() > obj.getString();

}

C\_Rgz& C\_Rgz::operator+=(C\_Rgz& obj)

{

object += obj.getObject();

return \*this;

}

bool C\_Rgz::operator==(C\_Rgz& obj)

{

return getString() == obj.getString();

}

C\_Rgz& C\_Rgz::operator=(C\_Rgz& temp)

{

object = temp.getObject();

return \*this;

}

C\_Rgz::C\_Rgz():object("Nothing")

{

}

C\_Rgz::C\_Rgz(string str):object(str)

{

}

C\_Rgz::C\_Rgz(C\_Rgz& a):object(a.getObject())

{

}

ostream& operator<<(ostream& output, C\_Rgz& obj)

{

output << obj.getObject();

return output;

}

istream& operator>>(istream& input, C\_Rgz& obj)

{

obj.input(input);

return input;

}

C\_RgzM.cpp

#include "C\_RgzM.h"

void C\_RgzM::setMark(const int a)

{

mark = a;

}

int C\_RgzM::getMark() const

{

return mark;

}

void C\_RgzM::input(istream& a)

{

a >> object >> mark;

}

bool C\_RgzM::operator==(C\_RgzM& obj)

{

return getString()==obj.getString();

}

C\_Rgz& C\_RgzM::operator=(C\_RgzM& temp)

{

object = temp.getObject();

mark = temp.getMark();

return \*this;

}

string C\_RgzM::getString()const

{

stringstream ss;

ss << object << " " << mark;

return ss.str();

}

C\_RgzM::C\_RgzM():mark(0)

{

setObject("Nothing");

}

C\_RgzM::C\_RgzM(string str, int m):mark(m)

{

setObject(str);

}

C\_RgzM::C\_RgzM(C\_RgzM& obj):mark(obj.getMark())

{

setObject(obj.getObject());

}

Source.cpp

#include <iostream>

#include "C\_RgzM.h"

#include <vector>

using std::vector;

using std::auto\_ptr;

using std::endl;

using std::cin;

using std::cout;

using std::unique\_ptr;

using std::shared\_ptr;

using std::weak\_ptr;

using std::make\_shared;

C\_Rgz\* creatEl();

void unique\_ptr\_vect();

void auto\_ptr\_vect();

void shared\_ptr\_vect();

void weak\_ptr\_vect();

int main() {

unique\_ptr\_vect();

auto\_ptr\_vect();

shared\_ptr\_vect();

weak\_ptr\_vect();

if (\_CrtDumpMemoryLeaks())

cout << "\nMemory leack deteckted\n";

else

cout << "\nMemory is not leack deteckted\n";

}

void auto\_ptr\_vect() {

vector <C\_Rgz\*> vect;

auto\_ptr <C\_Rgz> pointer1(creatEl());

auto\_ptr<C\_Rgz> pointer2(creatEl());

auto\_ptr<C\_Rgz> pointer3(creatEl());

auto\_ptr<C\_Rgz> pointer4(creatEl());

vect.push\_back(pointer1.get());

vect.push\_back(pointer2.get());

vect.push\_back(pointer3.get());

vect.push\_back(pointer4.get());

cout << "=======================";

for (auto var: vect) {

cout<<endl<< \*var<<endl;

};

cout << "=======================";

vect.pop\_back();

cout << "=======================";

for (auto var : vect) {

cout << endl << \*var << endl;

};

cout << "=======================";

}

void unique\_ptr\_vect() {

vector <C\_Rgz\*> vect;

unique\_ptr<C\_Rgz> pointer1(creatEl());

unique\_ptr<C\_Rgz> pointer2(creatEl());

unique\_ptr<C\_Rgz> pointer3(creatEl());

unique\_ptr<C\_Rgz> pointer4(creatEl());

vect.push\_back(pointer1.get());

vect.push\_back(pointer2.get());

vect.push\_back(pointer3.get());

vect.push\_back(pointer4.get());

cout << "=======================";

for (auto var : vect) {

cout << endl << \*var << endl;

};

cout << "=======================";

vect.pop\_back();

cout << "=======================";

for (auto var : vect) {

cout << endl << \*var << endl;

};

cout << "=======================";

}

void shared\_ptr\_vect() {

vector<C\_Rgz\*> vect;

shared\_ptr < C\_Rgz> ptr1(creatEl());

shared\_ptr < C\_Rgz> ptr2(creatEl());

shared\_ptr < C\_Rgz> ptr3(creatEl());

shared\_ptr < C\_Rgz> ptr4(creatEl());

vect.push\_back(ptr1.get());

vect.push\_back(ptr2.get());

vect.push\_back(ptr3.get());

vect.push\_back(ptr4.get());

cout << "=======================";

for (auto var : vect) {

cout << endl << \*var << endl;

};

cout << "=======================";

vect.pop\_back();

cout << "=======================";

for (auto var : vect) {

cout << endl << \*var << endl;

};

cout << "=======================";

}

void weak\_ptr\_vect() {

vector<C\_Rgz\*> vect;

shared\_ptr<C\_Rgz>sptr1(creatEl());

shared\_ptr<C\_Rgz>sptr2(creatEl());

shared\_ptr<C\_Rgz>sptr3(creatEl());

shared\_ptr<C\_Rgz>sptr4(creatEl());

weak\_ptr<C\_Rgz> ptr1 = sptr1;

weak\_ptr<C\_Rgz> ptr2 = sptr2;

weak\_ptr<C\_Rgz> ptr3 = sptr3;

weak\_ptr<C\_Rgz> ptr4 = sptr4;

vect.push\_back(ptr1.lock().get());

vect.push\_back(ptr2.lock().get());

vect.push\_back(ptr3.lock().get());

vect.push\_back(ptr4.lock().get());

cout << "=======================";

for (size\_t i = 0; i < 4; i++) {

cout << endl << \*vect[i] << endl;

};

cout << "=======================";

}

C\_Rgz\* creatEl()

{

int choose;

cout << "\n1-Rgz\n2-RgzM\nchoose: ";

cin >> choose;

cout << endl;

C\_Rgz\* a;

if (choose == 1) {

a = new C\_Rgz;

cout << "\nObject:";

cin >> \*a;

}

else

{

a = new C\_RgzM;

cout << "\nObject, mark: ";

cin >> \*a;

}

return a;

};

C\_Rgz.h

#pragma once

#include <iostream>

#include <sstream>

using std::string;

using std::istream;

using std::ostream;

using std::cout;

using std::cin;

using std::stringstream;

class C\_Rgz

{

protected:

string object;

public:

virtual void setObject(const string str);

virtual string getObject() const;

virtual string getString() const;

virtual void input(istream& a);

friend ostream& operator<< (ostream& output, C\_Rgz& obj);

virtual bool operator>(C\_Rgz& obj);

virtual C\_Rgz& operator+=(C\_Rgz& obj);

virtual bool operator==(C\_Rgz& obj);

virtual C\_Rgz& operator= (C\_Rgz& temp);

friend istream& operator>> (istream& input, C\_Rgz& obj);

C\_Rgz();

C\_Rgz(string str);

C\_Rgz(C\_Rgz &a);

virtual ~C\_Rgz() = default;

};

C\_RgzM.h

#pragma once

#include "C\_Rgz.h"

class C\_RgzM :

public C\_Rgz

{

private:

int mark;

public:

void setMark(const int a);

int getMark()const;

virtual void input(istream& a);

virtual bool operator==(C\_RgzM& obj);

virtual C\_Rgz& operator= (C\_RgzM& temp);

string getString() const override;

C\_RgzM();

C\_RgzM(string str, int m);

C\_RgzM(C\_RgzM& obj);

};

Висновок

При виконанні даної лабораторної роботи було набуто практичного досвіду роботи з розумними вказівниками.

Було розроблено програму, що використовує розумні вказівники.

Розумні вказівники, це комфортний спосіб створення динамічних об’єктів, адже не потрібно слідкувати за видаленням пам’яті.

Витоків пам’яті немає, виконується без помилок.