Автор: Єлєсін Артем , КІТ-119б

Дата: 01.06.2020

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

```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

```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.