Автор: Білий Вадим, КІТ-119а

Дата: 28.05.2020

Лабораторна робота 16. РОБОТА З ДИНАМІЧНОЮ ПАМ’ЯТТЮ

Тема. Системна робота з динамічною пам’яттю.

Мета – дослідити особливості мови С++ при роботі з динамічною пам’яттю.

Загальне завдання

Маючи класи з прикладної області РЗ (тільки базовий клас та клас / класи-спадкоємці), перевантажити оператори new / new [] та delete / delete []. Продемонструвати їх роботу і роботу операторів розміщення new / delete при розробці власного менеджера пам’яті (сховища).

Детальна інформація про власне сховище: є статично виділений масив заданого обсягу. Організувати виділення і звільнення пам’яті елементів ієрархії класів тільки у рамках цього сховища.

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())
{
}

void* C_Rgz::operator new(size_t size)
{
	return ::operator new(size);
}

void* C_Rgz::operator new[](size_t size)
{
	return ::operator new[](size);
}

void C_Rgz::operator delete(void* ptr)
{
	::operator delete(ptr);
}

void C_Rgz::operator delete[](void* ptr)
{
	::operator delete(ptr);
}

ostream& operator<<(ostream& output, C_Rgz& obj)
{
	output << obj.getString();
	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());
}

void* C_RgzM::operator new(size_t size)
{
	return ::operator new(size);
}

void* C_RgzM::operator new[](size_t size)
{
	return ::operator new[](size);
}

void C_RgzM::operator delete(void* ptr)
{
	::operator delete(ptr);
}

void C_RgzM::operator delete[](void* ptr)
{
	::operator delete(ptr);
}
```

Source.cpp

```
#include <iostream>
#include "C_RgzM.h"
#include <vector>
using std::cout;
using std::endl;
int main() {
	const int N = 3;
	C_Rgz* array[N];
	*array = new C_Rgz("Math");
	*(array + 1) = new C_Rgz("Physics");
	*(array + 2) = new C_RgzM("Chemistry",100);
	for (size_t i = 0; i < N; i++)
		cout << *array[i] << endl;
	cout << endl;
	C_Rgz* arrRgz = new C_Rgz[N]();
	arrRgz[0].setObject("1");
	arrRgz[1].setObject("2");
	arrRgz[2].setObject("3");
	for (size_t i = 0; i < N; i++)
		cout << arrRgz[i] << endl;
	cout << endl;
	for (size_t i = 0; i < N; i++)
		delete*(array + i);
	delete[] arrRgz;
	if (_CrtDumpMemoryLeaks())
		cout << "ERROR! Memory leak!" << endl;
	else
		cout << endl << "There is no memory leak" << endl;

}
```

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;

        void* operator new(size_t size);
        void* operator new[](size_t size);
        void operator delete(void* ptr);
        void operator delete[](void* ptr);
};
```

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);

        void* operator new(size_t size);
        void* operator new[](size_t size);
        void operator delete(void* ptr);
        void operator delete[](void* ptr);
};
```

Висновок

При виконанні даної лабораторної роботи було набуто практичного досвіду роботи з виключеннями.

Було створено програму, що використовує перевантаження операторів

new/new[] та delete/delete[].

Перевантаження операторів виділення пам’яті дозволяє одразу ініціалізувати об’єкт.

Витоків пам’яті немає, виконується без помилок.