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.