Автор: Білий Вадим, КІТ-119а

Дата: 25.05.2020

Лабораторна робота 11. ШАБЛОННІ КЛАСИ

Тема. Шаблонні функції.

Мета – поширити знання у шаблонізації (узагальненні) на основі вивчення шаблонних класів та створення власних шаблонних типів.

Загальне завдання

Модернізувати клас, що був розроблений у попередній роботи таким шляхом:

- зробити його шаблонним;

- додати поле– шаблонний масив;

- видалити з аргументів існуючих методів масив, а замість цього використовувати масив-поле класу.

Необхідно продемонструвати роботу програми як з використанням стандартних типів даних, так і типів, які створені користувачем.

Додаткове завдання на оцінку «відмінно»:

- продемонструвати роботу шаблонного класу, в масиві якого знаходиться ієрархія класів (тобто не тільки базовий клас, а ще й класспадкоємець).

Опис класів

Клас с основним завданням: CList

Клас ргз: C\_Rgz

Клас наслідник ргз: C\_RgzM

Опис змінних

T\*\* mass; - масив

string object; - назва об’єкту

int mark; - оцінка

Опис методів

T\*\* getMass(); - повертає масив

void showMass(int size); - показує данні

int index(T el, int size); - індекс

T\* sortMass(int size); - сортування

T minEl(int size); - мінімальний елемент

CList(T\*\* mass); -конструктор с параметрами

~CList() - деструктор

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();

}

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

{

return getString()>obj.getString();

}

int C\_Rgz::getMark()

{

return 0;

}

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.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()

{

return mark;

}

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

{

a >> object >> mark;

}

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

{

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

}

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

{

object = temp.getObject();

mark = temp.getMark();

return \*this;

}

/\*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;

string a;

a = ss.str();

a = " " + ss.str();

return a;

}

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());

}

/\*ostream& operator<<(ostream& output, C\_RgzM& obj)

{

output << obj.getString();

return output;

}\*/

CList.cpp

#include "CList.h"

template<class T>

void CList<T>::showMass(int size)

{

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

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

}

cout << endl;

}

template<class T>

int CList<T>::index(T el, int size)

{

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

if(\*mass[i]==el)

return i;

}

return -1;

}

template<class T>

T\* CList<T>::sortMass(int size)

{

bool prz = 0;

T\* temp = NULL;

do {

prz = 0;

for (size\_t i = 0; i < size - 1; i++) {

if (\*mass[i] > \* mass[i + 1]) {

temp = \*(mass + i);

\*(mass + i) = \*(mass + i + 1);

\*(mass + i + 1) = temp;

prz = 1;

}

}

} while (prz);

return temp;

}

template<class T>

T CList<T>::minEl(int size)

{

T min = \*mass[0];

for (size\_t i = 1; i < size; i++)

{

if (\*mass[i] < min)

{

min = \*mass[i];

}

}

return min;

}

template<class T>

T\*\* CList<T>::getMass()

{

return mass;

}

template<class T>

inline CList<T>::CList(T\*\* mass):mass(mass)

{

}

Source.cpp

#include "CList.cpp"

#include "C\_Rgz.h"

#include "C\_RgzM.h"

struct MYTYPE {

string ch;

//MYTYPE& operator=(MYTYPE& temp) {

//ch = temp.ch;

//return \*this;

//};

};

ostream& operator<<(ostream& output, MYTYPE obj)

{

output << obj.ch;

return output;

};

istream& operator>>(istream& input, MYTYPE& obj)

{

input >> obj.ch;

return input;

};

bool operator==(MYTYPE& a, MYTYPE& b) {

return a.ch == b.ch;

};

bool operator>(MYTYPE& a, MYTYPE& b) {

return a.ch > b.ch;

};

int main() {

{

C\_Rgz obj1("Math");

C\_RgzM obj2("Art",10);

C\_Rgz obj3("Algoritmh");

C\_Rgz\*\* a= new C\_Rgz\*[6];

for (size\_t i = 0; i < 3; i++) {

a[i] = new C\_Rgz;

\*a[i] = obj1;

}

for (size\_t i = 3; i < 6; i++) {

a[i] = new C\_RgzM;

\*a[i] = obj2;

}

\*a[2] = obj3;

CList <C\_Rgz> b(a);

cout << b.index(obj3,6)<<endl;

cout << endl;

b.showMass(6);

cout << endl;

b.sortMass(6);

b.showMass(6);

cout << endl;

cout << b.minEl(6).getString();

for (size\_t i = 0; i < 6; i++)

{

delete a[i];

}

delete a;

}

if (\_CrtDumpMemoryLeaks())

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

else

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

}

Test.cpp

#include "CList.cpp"

int main() {

{

int\*\* test= new int\*[8];

for (size\_t i = 0; i < 8; i++)

{

test[i] = new int;

}

\*test[0] = 0;

\*test[1] = 1;

\*test[2] = 0;

\*test[3] = 6;

\*test[4] = 5;

\*test[5] = 7;

\*test[6] = 844;

\*test[7] = 9;

CList<int> list(test);

int rez1[8] = { 0,0,1,5,6,7,9,844 };

int rez2 = 0;

list.sortMass(8);

for (int i = 0; i < 8; i++) {

if (\*list.getMass()[i] == rez1[i])

cout << "test 1." << i << ": true"<<endl;

else

cout << "test 1." << i << ": false" << endl;

}

if(list.minEl(8)==rez2)

cout << "test 2: true" << endl;

else

cout << "test 2: false" << endl;

if(list.index(844,8)==7)

cout << "test 3: true" << endl;

else

cout << "test 3: false" << endl;

}

if (\_CrtDumpMemoryLeaks())

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

else

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

}

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;

using std::getline;

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 bool operator>(C\_Rgz& obj);

virtual int getMark();

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()override;

virtual void input(istream& a);

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

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

string getString() const override;

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

C\_RgzM();

C\_RgzM(string str, int m);

C\_RgzM(C\_RgzM& obj);

};

CList.h

#pragma once

#include <iostream>

using std::cout;

using std::cin;

using std::endl;

using std::ostream;

using std::istream;

using std::string;

using std::ostream;

template<class T>

class CList

{

private:

T\*\* mass;

public:

T\*\* getMass();

void showMass(int size);

int index(T el, int size);

T\* sortMass(int size);

T minEl(int size);

CList(T\*\* mass);

~CList()= default;

};

Висновок

При виконанні даної лабораторної роботи було набуто практичного досвіду роботи з шаблонними класами.

Було розроблено програму, що працює з шаблонним класом.

Шаблонний клас, це клас який працює з заздалегідь невідомими типами змінних.

Програма протестована, витоків пам’яті немає, виконується без помилок.