Пенза 2019

Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

Выполнил:

студенты группы 17ВВ1

Кокин Денис

Беленков Никита

Принял:

Дорошенко И.Н.

**ОТЧЁТ**

по лабораторной работе №8

по курсу «Технологии программирования»

на тему «Сериализация и восстановление»

**Цель работы:** Изучение механизмов сохранения и восстановления объектов данных. Получение навыков разработки самовосстанавливающихся структур данных.

**Лабораторное задание**

Лабораторное задание выполняется на базе лабораторной работы №7.

Требования к классам: В класс Interface добавить две чистые виртуальные функции: Serialize и Deserialize. В классах Derived1, Derived2 и Derived3 реализовать виртуальные функции Serialize и Deserialize. – Serialize – функция сериализация данных. Функция должна сохранять все свои данные в буфер, указанный в параметрах функции. – Deserialize – функция восстановления данных. Функция должна восстанавливать данные объекта из буфера, указанного в параметрах функции. В класс <Структура> добавить два метода: – Serialize – функция сериализация структуры данных. Функция должна сохранить все элементы структуры в файл. В своей работе функция должна вызывать соответствующую функцию Serialize для всех объектов. – Deserialize – функция восстановления структуры данных. Функция должна восстанавливать все элементы структуры из файла. Во время работы функция должна вызывать соответствующую функцию Serialize для всех элементов. В процессе восстановления данных все элементы должны заново размещаться в динамической памяти. Т.е.

все объекты классов Derived1, Derived2 и Derived3 должны заново создаваться, после чего размещенные объекты добавляются в структуру.

Требования к программе: – В основной программе необходимо разместить объект класса <Структура> в динамической памяти. – Выполнить добавление нескольких различных элементов в структуру (Derived1, Derived2 и Derived3). – Вывести на экран содержимое всей структуры. – Сериализовать всю структуру в файл. – Удалить объект класса <Структура> из динамической памяти. При этом все элементы структуры также должны быть удалены. – Разместить новый объект класса <Структура> в динамической памяти. – Восстановить сохраненную структуру данных из файла. – Вывести на экран содержимое всей структуры. Повторная печать должна полностью повторить содержимое первой печати, доказав таким образом что данные восстановлены корректно.

**Листинг программы**

#include <iostream>

#include <wchar.h>

#include <iostream>

#include <conio.h>

#include <fstream>

#ifdef \_MSC\_VER

#define \_CRT\_SECURE\_NO\_WARNINGS

#endif

using namespace std;

int g\_nCount = 0;

inline void\* operator new(size\_t s)

{

g\_nCount++;

return malloc(s);

}

inline void operator delete(void\* p)

{

g\_nCount--;

free(p);

}

char\* substr(char\* arr, int begin, int len)

{

char\* res = new char[len + 1];

memset(res, 0, len + 1);

for (int i = 0; i < len; i++)

res[i] = \*(arr + begin + i);

return res;

}

class Interface {

public:

virtual void Input(void) {};

virtual void Output(void) {};

virtual void Serialize(ofstream&) {};

virtual void Deserialize(ifstream&) {};

virtual ~Interface() {};

};

class Locality : public Interface {

public:

char\* Name;

char\* Region;

Locality() :

Population(0), Budget(0), Code(0) {

init();

};

Locality(int Population, char\* Region) :

Population(Population), Budget(0), Code(0) {

init();

strcpy\_s(this->Region, strlen(Region) + 1, Region);

};

Locality(Locality& Local) :

Population(Local.Population), Budget(Local.Budget), Code(Local.Code) {

init();

};

Locality(char\* Name, char\* Region, int Code = 43, int Population = 100000, int Budget = 50000) :

Population(Population), Budget(Budget), Code(Code) {

init();

strcpy\_s(this->Region, strlen(Region) + 1, Region);

strcpy\_s(this->Name, strlen(Name) + 1, Name);

};

Locality(char\* Region, int Code) :

Population(0), Budget(0), Code(Code) {

init();

strcpy\_s(this->Region, strlen(Region) + 1, Region);

};

Locality(int Population, int Budget) :

Population(Population), Budget(Budget), Code(0) {

init();

};

void init() {

Name = new char[50];

memset(Name, 0, 50);

Region = new char[50];

memset(Region, 0, 50);

}

void Input(void) {

delete Name;

delete Region;

init();

wcout << "Enter name of locality: "; cin >> Name;

wcout << "Enter region of locality: "; cin >> Region;

wcout << "Enter code of locality: "; cin >> Code;

wcout << "Enter population of locality: "; cin >> Population;

wcout << "Enter budget of locality: "; cin >> Budget;

}

void Output(void) {

wcout << "Name: " << Name << endl;

wcout << "Region: " << Region << endl;

wcout << "Code: " << Code << endl;

wcout << "Population: " << Population << endl;

wcout << "Budget: " << Budget << endl;

}

void ClassInfo() {

wcout << "Class size: " << sizeof(Locality) << endl;

wcout << "Length of char's arrays: " << sizeof(Name) << endl;

this->AbleBodied();

this->BudgetToUSD();

wcout << endl;

}

~Locality() {

delete Name;

delete Region;

Population = NULL;

Code = NULL;

Budget = NULL;

};

void Clear() {

this->Locality::~Locality();

}

void Serialize(ofstream &fint) {

int Buf = 1;

fint.write((char\*)&Buf, 4);

fint.write((char\*)Name, 50);

fint.write((char\*)Region, 50);

fint.write((char\*)&Population, sizeof(Population));

fint.write((char\*)&Code, sizeof(Code));

fint.write((char\*)&Budget, sizeof(Budget));

}

void Deserialize(ifstream &fout) {

init();

fout.read((char\*)Name, 50);

fout.read((char\*)Region, 50);

fout.read((char\*)&Population, sizeof(Population));

fout.read((char\*)&Code, sizeof(Code));

fout.read((char\*)&Budget, sizeof(Budget));

}

protected:

int Population;

int Code;

int Budget;

void AbleBodied() {

wcout << "Able-bodied population: " << (this->Population / 2) << endl;

}

private:

void BudgetToUSD() {

wcout << "Budget of locality: " << (this->Budget / 65) << " USD" << endl;

}

};

class City : public Locality {

public: int isRegionCenter;

void Output() {

Locality::Output();

wcout << "Count of district: " << isRegionCenter << endl;

}

void Input() {

Locality::Input();

wcout << "Enter a count of district: "; cin >> isRegionCenter;

}

void Serialize(ofstream& fint) {

int Buf = 2;

fint.write((char\*)&Buf, 4);

fint.write((char\*)Name, 50);

fint.write((char\*)Region, 50);

fint.write((char\*)& Population, sizeof(Population));

fint.write((char\*)& Code, sizeof(Code));

fint.write((char\*)& Budget, sizeof(Budget));

fint.write((char\*)& isRegionCenter, sizeof(isRegionCenter));

}

void Deserialize(ifstream& fout) {

init();

fout.read((char\*)Name, 50);

fout.read((char\*)Region, 50);

fout.read((char\*)& Population, sizeof(Population));

fout.read((char\*)& Code, sizeof(Code));

fout.read((char\*)& Budget, sizeof(Budget));

fout.read((char\*)& isRegionCenter, sizeof(isRegionCenter));

}

};

class Township : public Locality {

public: int SchoolCount;

void Output() {

Locality::Output();

wcout << "Schools: " << SchoolCount << endl;

}

void Input() {

Locality::Input();

wcout << "Enter a count of schools: "; cin >> SchoolCount;

}

void Serialize(ofstream& fint) {

int Buf = 3;

fint.write((char\*)&Buf, 4);

fint.write((char\*)Name, 50);

fint.write((char\*)Region, 50);

fint.write((char\*)& Population, sizeof(Population));

fint.write((char\*)& Code, sizeof(Code));

fint.write((char\*)& Budget, sizeof(Budget));

fint.write((char\*)& SchoolCount, sizeof(SchoolCount));

}

void Deserialize(ifstream& fout) {

init();

fout.read((char\*)Name, 50);

fout.read((char\*)Region, 50);

fout.read((char\*)& Population, sizeof(Population));

fout.read((char\*)& Code, sizeof(Code));

fout.read((char\*)& Budget, sizeof(Budget));

fout.read((char\*)& SchoolCount, sizeof(SchoolCount));

}

};

class Village : public Township {

public: int ShopCount;

void insert(const char\* str) {

char qstr[50] = { 0 };

memset(qstr, 0, 50);

char\* op1 = substr(this->Name, 2, strlen(this->Name) - 2);

strncpy\_s(qstr, op1, strlen(this->Name) - 2);

delete op1;

memmove(qstr + 2 + strlen(str), qstr, strlen(this->Name));

for (int i = 2; i < (strlen(str) + 2); i++)

qstr[i] = str[i - 2];

qstr[0] = this->Name[0];

qstr[1] = this->Name[1];

strcpy\_s(this->Name, 50, qstr);

}

void Output() {

Township::Output();

wcout << "Shops: " << ShopCount << endl;

}

void Input() {

Township::Input();

wcout << "Enter a count of shops: "; cin >> ShopCount;

}

void Serialize(ofstream& fint) {

int Buf = 4;

fint.write((char\*)&Buf, 4);

fint.write((char\*)Name, 50);

fint.write((char\*)Region, 50);

fint.write((char\*)& Population, sizeof(Population));

fint.write((char\*)& Code, sizeof(Code));

fint.write((char\*)& Budget, sizeof(Budget));

fint.write((char\*)& SchoolCount, sizeof(SchoolCount));

fint.write((char\*)& ShopCount, sizeof(ShopCount));

}

void Deserialize(ifstream& fout) {

init();

fout.read((char\*)Name, 50);

fout.read((char\*)Region, 50);

fout.read((char\*)& Population, sizeof(Population));

fout.read((char\*)& Code, sizeof(Code));

fout.read((char\*)& Budget, sizeof(Budget));

fout.read((char\*)& SchoolCount, sizeof(SchoolCount));

fout.read((char\*)& ShopCount, sizeof(ShopCount));

}

bool operator ||(char\* right) {

char\* op1 = substr(this->Name, 2, 5);

char\* op2 = substr(right, 2, 5);

if (strcmp(op1, op2)) {

delete op1;

delete op2;

return false;

}

else {

delete op1;

delete op2;

return true;

}

}

bool operator ||(Village& right) {

return this->ShopCount == right.ShopCount;

}

void operator -(const int right) {

char str[10] = { 0 };

sprintf\_s(str, "%d", right);

insert(str);

}

void operator -(const char\* right) {

insert(right);

}

void operator -(const const Village& right) {

insert(right.Name);

}

};

bool operator || (const Village& left, const int right) {

if (strlen(left.Name) <= right)

return true;

else

return false;

}

class LocalityList {

public:

class LocalityNode {

public:

Interface\* LocalityData;

LocalityNode\* NextNode;

LocalityNode(ifstream &fout) {

int TypeBuf;

fout.read((char\*)&TypeBuf, 4);

switch (TypeBuf) {

case(1): {

LocalityData = new Locality();

break;

}

case(2): {

LocalityData = new City();

break;

}

case(3): {

LocalityData = new Township();

break;

}

case(4): {

LocalityData = new Village();

break;

}

default: {

LocalityData = nullptr;

break;

}

}

NextNode = nullptr;

}

LocalityNode() {

NextNode = nullptr;

cout << "Which kind of Interface should be created?" << endl;

int BufCin;

cin >> BufCin;

switch (BufCin)

{

case(1): {

LocalityData = new Locality();

break;

}

case(2): {

LocalityData = new City();

break;

}

case(3): {

LocalityData = new Township();

break;

}

case(4): {

LocalityData = new Village();

break;

}

default:

cout << "This type not exists" << endl;

break;

}

LocalityData->Input();

}

~LocalityNode() {

delete LocalityData;

}

};

LocalityNode\* FirstNode;

LocalityList() {

FirstNode = nullptr;

}

LocalityNode\* GetFirst() {

return FirstNode;

}

LocalityNode\* GetNext(LocalityNode\* PrevNode) {

return PrevNode->NextNode;

}

void PrtinAll() {

LocalityNode\* CurrentNode = GetFirst();

if (FirstNode == nullptr)

return;

FirstNode->LocalityData->Output();

while (CurrentNode->NextNode != nullptr) {

CurrentNode = GetNext(CurrentNode);

CurrentNode->LocalityData->Output();

}

}

void PushBack(LocalityNode\* LastNode) {

LocalityNode\* CurrentNode = GetFirst();

if (CurrentNode == nullptr) {

FirstNode = LastNode;

return;

}

while (CurrentNode->NextNode != nullptr)

CurrentNode = GetNext(CurrentNode);

CurrentNode->NextNode = LastNode;

}

void RemoveLast() {

if (FirstNode == nullptr)

return;

else {

LocalityNode\* PrevNode = GetFirst();

if (PrevNode->NextNode == nullptr) {

delete PrevNode;

return;

}

else {

LocalityNode\* CurrentNode = PrevNode->NextNode;

while (CurrentNode->NextNode != nullptr) {

PrevNode = CurrentNode;

CurrentNode = GetNext(CurrentNode);

}

delete GetNext(PrevNode);

PrevNode->NextNode = nullptr;

}

}

}

void InsertFirst(LocalityNode\* NewFirstNode) {

if (GetFirst() == nullptr)

FirstNode = NewFirstNode;

else {

NewFirstNode->NextNode = GetFirst();

FirstNode = NewFirstNode;

}

}

~LocalityList() {

if (FirstNode == nullptr)

return;

else {

LocalityNode\* PrevNode = FirstNode;

FirstNode = nullptr;

if (PrevNode->NextNode == nullptr) {

delete PrevNode;

return;

}

else {

LocalityNode\* CurrentNode = PrevNode->NextNode;

while (CurrentNode->NextNode != nullptr) {

delete PrevNode;

PrevNode = CurrentNode;

CurrentNode = GetNext(CurrentNode);

}

delete PrevNode;

delete CurrentNode;

}

}

}

void Serialize() {

ofstream fint;

fint.open("text.txt", ios::binary | ios::out);

LocalityNode\* CurrentNode = GetFirst();

if (FirstNode == nullptr)

return;

while (CurrentNode->NextNode != nullptr) {

CurrentNode->LocalityData->Serialize(fint);

CurrentNode = GetNext(CurrentNode);

}

CurrentNode->LocalityData->Serialize(fint);

fint.close();

}

void Deserialize() {

LocalityNode\* TempNode;

ifstream fout;

fout.open("text.txt");

FirstNode = new LocalityNode(fout);

FirstNode->LocalityData->Deserialize(fout);

if(!fout.eof())

TempNode = FirstNode->NextNode = new LocalityNode(fout);

else{

FirstNode->NextNode = nullptr;

return;

}

while (true) {

TempNode->LocalityData->Deserialize(fout);

TempNode->NextNode = nullptr;

if (!fout.eof()) {

TempNode->NextNode = new LocalityNode(fout);

if (TempNode->NextNode->LocalityData == nullptr) {

RemoveLast();

break;

}

TempNode = TempNode->NextNode;

}

};

}

};

int main() {

cout << g\_nCount << endl;

LocalityList\* list = new LocalityList();

list->PushBack(new LocalityList::LocalityNode());

list->InsertFirst(new LocalityList::LocalityNode());

list->PushBack(new LocalityList::LocalityNode());

list->PushBack(new LocalityList::LocalityNode());

list->RemoveLast();

list->PrtinAll();

list->Serialize();

cout << endl << "Serializing" << endl << endl;

delete list;

list->Deserialize();

list->PrtinAll();

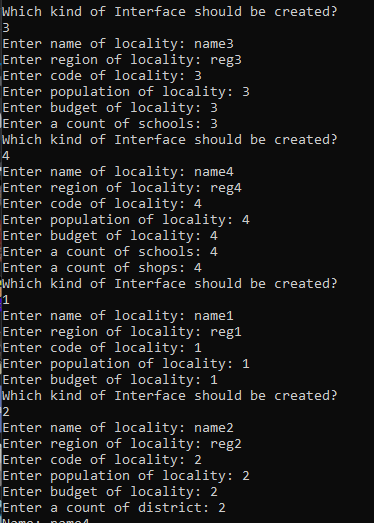
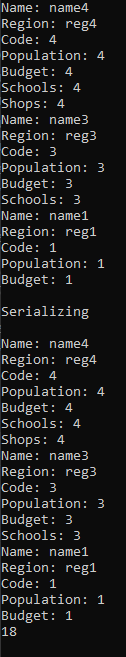
cout << g\_nCount << endl;

\_getch();

return 1;

}

**Результаты работы программы**



**Вывод:** студенты изучили механизмы сохранения и восстановления объектов данных. Получили навыки разработки самовосстанавливающихся структур данных.