**Контрольная работа**

по предмету

«Программирование»

Исполнитель:

студент гр. ПК-11-1з

Шевцов Леонид Анатольевич

# Оглавление

Оглавление 2

Введение 3

Суть работы 3

Выводы 4

Список использованной литературы 5

Исходный код 5

main.cpp 5

shop.h 6

shop.cpp 7

menu.h 9

menu.cpp 10

# Введение

Эта работа ставит задачей изучить основы объектно-ориентированного программирования на языке C++, включая такие понятия, как:

* Определение классов
* Задание конструктора и деструктора
* Приватные свойства объектов
* Публичные и приватные методы
* Дружественные методы

А также

* использование библиотеки iostream для потокового ввода и вывода,
* перегрузку оператора для ввода и вывода объектов пользовательских классов,
* построение интерфейса меню,
* чтение и запись объекта в файл.

# Суть работы

Для выполнения задания была написана программа на языке C++.

Программа была разбита на три модуля:

* модуль, реализующий заданный условием основной класс программы;
* модуль, реализующий интерфейс меню;
* основной модуль, демонстрирующий работу класса и меню.

Для упрощения создания интерфейса меню была создана подпрограмма, организующая выбор пункта из заранее заданного набора; это позволило упростить основной код программы и отвязать реализацию задания от (достаточно сложного технически) пользовательского интерфейса. Такую подпрограмму можно использовать и в других программах, требующих реализации меню.

При огранизации записи в файл и чтения из файла был выбор между перегрузкой оператора и написанием отдельной функции. Я выбрал написание функцию, т.к. перегрузку оператора опробовал на выводе объекта на экран, а в случае ввода-вывода.

Кроме того, при организации ввода оказалось, что использования оператора ввода (>>) очень неудобно по причине отсутствия обработки ошибок. Поэтому, после проверок работы программы на намеренно ошибочных входных данных, код

int integer\_variable  
cin >> integer\_variable;

пришлось поменять на

int integer\_variable;  
char buffer[100], \*ptr;  
cin.getline(buffer, 100);  
integer\_variable = strtol(buffer, &ptr, 10);

Иначе при попытке ввести нечисловое значение программа запирается.

Другой источник проблем – ввод строки; поток прекращает чтение, как только встречает пробел.

Для хранения объектов в файле был выбран самый простой формат, в котором в каждой строчке хранится по по одному полю, и объекты разделяются числом полей. Отсутствие счетчика в начале файла позволило легко дописывать объекты в конец файла, что было важно в выбранной реализации (если бы был счетчик, его бы нужно было менять при дописывании.)

Зато это несколько усложнило чтение объектов из файла; вместо чтения счетчика и выделения места под все объекты пришлось выделять место постепенно, функцией realloc. У такого подхода есть свои преимущества: он может работать с потоком, в котором число объектов не может быть известно в принципе. Например, написанную процедуру можно использовать для ввода неограниченного числа объектов с клавиатуры, не спрашивая заранее их количество.

Еще одна проблема, которую пришлось решить – после чтения последнего объекта курсор не переводился в самый конец файла. Чтоб избежать создания лишнего объекта, пришлось принудительно пытаться считать следующий символ из файла, и, если конец файла не был достигнут, возвращать символ назад методом fstream.putback().

# Выводы

Язык C++ позволяет нам реализовывать объектно-ориентированные программы, описывать классы объектов, и вводить и выводить объекты на экран или в файл.

# Список использованной литературы

1. StackOverflow <http://stackoverflow.com>
2. MSDN C++ Language Reference http://msdn.microsoft.com/en-us/library/3bstk3k5.aspx

# Исходный код

## main.cpp

#include <stdio.h>  
#include <stdlib.h>  
#include <iostream>  
#include <fstream>  
  
#include "menu.h"  
#include "shop.h"  
  
Shop \*theShop;  
  
void mainPrompt() {  
 cout << endl << "=== Current shop state" << endl;  
 cout << \*theShop << endl;  
}  
  
  
void changeName() {  
 char buffer[100];  
 cout << "New shop name: ";  
 cin.getline(buffer, 100);  
 theShop->setName(buffer);  
}  
  
void changeManager() {  
 char buffer[100];  
 cout << "New manager name: ";  
 cin.getline(buffer, 100);  
 theShop->setManager(buffer);  
}  
  
void changeNumber() {  
 char buffer[100], \*ptr;  
 cout << "New shop number: ";  
 cin.getline(buffer, 100);  
 theShop->setNumber(strtol(buffer, &ptr, 10));  
}  
  
void changeSalesArea() {  
 char buffer[100], \*ptr;  
 cout << "New sales area: ";  
 cin.getline(buffer, 100);  
 theShop->setSalesArea(strtof(buffer, &ptr));  
}  
  
void changeWarehouseArea() {  
 char buffer[100], \*ptr;  
 cout << "New warehouse area: ";  
 cin.getline(buffer, 100);  
 theShop->setWarehouseArea(strtof(buffer, &ptr));  
}  
  
void appendToFile() {  
 fstream shopsFile;  
  
 shopsFile.open("shops.txt", fstream::out | fstream::app);  
 theShop->writeToFile(shopsFile);  
 shopsFile.close();  
  
 cout << "File updated!" << endl;  
}  
  
  
void printFile() {  
 Shop\*\* shopList = NULL;  
 int shopCount=0, i;  
  
 shopList = readShopsFile(shopCount);  
  
 if (shopCount > 0) {  
 sortShopList(shopList, shopCount);  
 printShopList(shopList, shopCount);  
  
 for (i=0; i<shopCount; ++i) {  
 delete \*(shopList+i);  
 }  
 free(shopList);  
 } else {  
 cout << "Shops.txt is empty!" << endl;  
 }  
}  
  
void wipeFile() {  
 fstream shopsFile;  
 shopsFile.open("shops.txt", fstream::out);  
 shopsFile.close();  
}  
  
const MenuItem mainMenu[] = {  
 {"Change shop name", &changeName},  
 {"Change manager name", &changeManager},  
 {"Change number", &changeNumber},  
 {"Change sales area", &changeSalesArea},  
 {"Change warehouse area", &changeWarehouseArea},  
 {"Append this shop to file (shops.txt)", &appendToFile},  
 {"Print list of shops in file ordered by name (shops.txt)", &printFile},  
 {"Wipe file (shops.txt)", &wipeFile}  
 };  
  
int main(int argc, char\*\* argv) {  
 theShop = new Shop("John's shop", "John Doe", 123, 100, 300);  
  
 runMenu(mainPrompt, mainMenu, 8);  
  
 delete theShop;  
 return 0;  
}

## shop.h

#ifndef \_\_SHOP\_H\_\_  
#define \_\_SHOP\_H\_\_  
  
#include <iostream>  
using namespace std;  
  
// The Shop class  
class Shop {  
public:  
 //per-element constructor  
 Shop(const char\* name, const char\* manager, int number,  
 float salesArea, float warehouseArea);  
 //copy constructor  
 Shop(const Shop &shop);  
 //destructor  
 ~Shop();  
  
 // Getter methods  
 const char\* getName() const;  
 const char \*getManager() const;  
 int getNumber() const;  
 float getSalesArea() const;  
 float getWarehouseArea() const;  
 // Setter methods  
 void setName(const char\* name);  
 void setManager(const char\* manager);  
 void setNumber(int number);  
 void setSalesArea(float salesArea);  
 void setWarehouseArea(float warehouseArea);  
 // Output  
 friend ostream& operator<<(ostream& os, const Shop& shop);  
 // File interface  
 ostream& writeToFile(ostream& os);  
 static Shop\* readFromFile(istream& is);  
  
private:  
 char\* name;  
 char\* manager;  
 int number;  
 float salesArea;  
 float warehouseArea;  
};  
  
// Reads a list of shops from file.  
// Returns number of shops into shopCount.  
Shop\*\* readShopsFile(int &shopCount);  
// Sorts shop list by name  
void sortShopList(Shop\*\* shopList, int shopCount);  
// Prints shop list to cout  
void printShopList(Shop\*\* shopList, int shopCount);  
  
#endif

## shop.cpp

#include <cstring>  
#include <iomanip>  
#include <fstream>  
#include "shop.h"  
  
Shop::Shop(const char\* \_name, const char\* \_manager,  
 int \_number, float \_salesArea, float \_warehouseArea):  
 number(\_number),  
 salesArea(\_salesArea),  
 warehouseArea(\_warehouseArea)  
{  
 name = strdup(\_name);  
 manager = strdup(\_manager);  
}  
  
Shop::Shop(const Shop &shop):  
 number(shop.number),  
 salesArea(shop.salesArea),  
 warehouseArea(shop.warehouseArea)  
{  
 name = strdup(shop.name);  
 manager = strdup(shop.manager);  
}  
  
Shop::~Shop() {  
 free(name);  
 free(manager);  
}  
  
const char\* Shop::getName() const {  
 return name;  
}  
  
const char \*Shop::getManager() const {  
 return manager;  
}  
  
int Shop::getNumber() const {  
 return number;  
}  
  
float Shop::getSalesArea() const {  
 return salesArea;  
}  
  
float Shop::getWarehouseArea() const {  
 return warehouseArea;  
}  
  
  
void Shop::setName(const char\* \_name) {  
 delete name;  
 name = strdup(\_name);  
}  
  
void Shop::setManager(const char\* \_manager) {  
 delete manager;  
 manager = strdup(\_manager);  
}  
  
void Shop::setNumber(int \_number) {  
 number = \_number;  
}  
  
void Shop::setSalesArea(float \_salesArea) {  
 salesArea = \_salesArea;  
}  
  
void Shop::setWarehouseArea(float \_warehouseArea) {  
 warehouseArea = \_warehouseArea;  
}  
  
ostream& operator<<(ostream& os, const Shop& shop) {  
 os << setw(20) << "Shop name: " << shop.getName() << endl;  
 os << setw(20) << "Number: " << shop.getNumber() << endl;  
 os << setw(20) << "Manager name: " << shop.getManager() << endl;  
 os << setw(20) << "Sales area: " << shop.getSalesArea() << endl;  
 os << setw(20) << "Warehouse area: " << shop.getWarehouseArea() << endl;  
 return os;  
}  
  
ostream& Shop::writeToFile(ostream& os) {  
 os << getName() << endl;  
 os << getNumber() << endl;  
 os << getManager() << endl;  
 os << getSalesArea() << endl;  
 os << getWarehouseArea() << endl;  
 return os;  
};  
  
Shop\* Shop::readFromFile(istream& is) {  
 char name[100];  
 char manager[100];  
 char buffer[100], \*ptr;  
 int number;  
 float salesArea;  
 float warehouseArea;  
  
 is.getline(name, 100);  
 is.getline(buffer, 100);  
 number = strtol(buffer, &ptr, 10);  
 is.getline(manager, 100);  
 is.getline(buffer, 100);  
 salesArea = strtof(buffer, &ptr);  
 is.getline(buffer, 100);  
 warehouseArea = strtof(buffer, &ptr);  
  
 return new Shop(name, manager, number, salesArea, warehouseArea);  
}  
  
Shop\*\* readShopsFile(int &shopCount) {  
 Shop\*\* shopList = NULL;  
 shopCount=0;  
  
 fstream shopsFile;  
 shopsFile.open("shops.txt", fstream::in);  
  
 while (1) {  
 // fix error with last shop not moving to EOF, by reading one char ahead  
 char lastchar;  
 shopsFile.read(&lastchar, 1);  
 if (shopsFile.eof()) {  
 break;  
 } else {  
 shopsFile.putback(lastchar);  
 }  
  
 if (!shopList) {  
 ++shopCount;  
 shopList = (Shop\*\*) malloc(sizeof(Shop\*));  
 } else {  
 // Grow array by another element. Not most efficient with memory,  
 // but avoids reading file twice  
 // (which also requires allocating memory).  
 shopList = (Shop\*\*) realloc(shopList, sizeof(Shop\*)\*(++shopCount));  
 }  
  
 \*(shopList+shopCount-1) = Shop::readFromFile(shopsFile);  
 }  
  
 shopsFile.close();  
  
 return shopList;  
}  
  
int shopComparator(const void\* shop1, const void\* shop2) {  
 return strcmp((\*(Shop\*\*)shop1)->getName(), (\*(Shop\*\*)shop2)->getName());  
}  
  
void sortShopList(Shop\*\* shopList, int shopCount) {  
 qsort(shopList, shopCount, sizeof(Shop\*), &shopComparator);  
}  
  
void printShopList(Shop\*\* shopList, int shopCount) {  
 for (int i=0; i<shopCount; ++i) {  
 cout << "=== Shop #"<<i+1<<endl;  
 cout << \*\*(shopList+i);  
 }  
 cout << "====== End of shops file " << endl << endl;  
}

## menu.h

#ifndef \_\_MENU\_H\_\_  
#define \_\_MENU\_H\_\_  
  
// A MenuItem consists of a label, and a handler method.  
struct MenuItem {  
 const char\* label;  
 void (\*handler)();  
};  
  
// Menu-driven interface  
// \* prompt (optional) - this method will be called every time  
// before the menu is rendered  
// \* commands - a list of commands  
// \* commandCount - number of commands in the list  
void runMenu(void (\*prompt)(), const MenuItem\* commands, int commandCount);  
  
#endif

## menu.cpp

#include <iostream>  
#include <iomanip>  
  
#include "menu.h"  
  
using namespace std;  
  
bool readInt(int &i) {  
 char buffer[100];  
 char \*ptr;  
 cin.getline(buffer, 100);  
 i = strtol(buffer, &ptr, 10);  
 return buffer[0] != '\n' && (\*ptr == '\n' || \*ptr == '\0');  
}  
  
void runMenu(void (\*prompt)(), const MenuItem\* commands, int commandCount) {  
 int choice;  
 while(1) {  
 if (prompt) (\*prompt)();  
 cout << "=== MENU" << endl;  
 for (int i=0;i<commandCount; ++i) {  
 cout << setw(2) << i+1 << ": " << commands[i].label << endl;  
 }  
 cout << " 0: Exit / Return" << endl << " > ";  
 if (!readInt(choice) || (choice>commandCount) || (choice<0)) {  
 cout << "Error!" << endl;  
 continue;  
 } else if (choice==0) {  
 break;  
 } else {  
 cout << endl << endl;  
 (\*commands[choice-1].handler)();  
 }  
 };  
}