Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Инженерно-экономический факультет

Кафедра экономической информатики

ОТЧЕТ

по лабораторной работе

**« ПРИНЦИП ПОЛИМОРФИЗМА. ВИРТУАЛЬНЫЕ ФУНКЦИИ. АБСТРАКТНЫЕ КЛАССЫ в языке C++»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 272303 |  | Н. Д. Вдовенко |
| Проверила |  | К.А.Борщевич |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск 2023

# Общая постановка задачи

**Цель работы:** изучить реализацию принципа динамического и статического полиморфизма, особенности использования перегрузки операторов, методов, переопределения функций.  
  
2. Разработать набор классов «Город» (наименование, количество жителей), «Магазин» (наименование, тип), «Товар» (наименование, сорт, количество, цена), «Банк» (номер счета, сумма денег на счете), «Покупатель» (фамилия, сумма денег, сумма покупки). Определить и реализовать связи между этими классами. Класс «Покупатель» содержит методы выполнения различных операций покупателя с товаром, приобретаемым в магазине иоплачиваемом через банк. Использовать конструктор с параметрами, конструктор без параметров, конструктор копирования. В класс добавить необходимый набор полей и методов (минимум два поля и два метода) на свое усмотрение. Предусмотреть метод для записи полученных данных в файл.

**Ответы на контрольные вопросы**

1. Виртуальная функция - это функция, объявленная в базовом классе с ключевым словом "virtual" и переопределенная в производных классах. Она позволяет вызывать соответствующую версию функции, определенную в объекте производного класса, даже если тип указателя или ссылки - базовый класс.  
  
2. Чисто виртуальная функция - это виртуальная функция, объявленная в базовом классе с помощью "= 0" в ее объявлении. Она не имеет определения в базовом классе и должна быть переопределена в производных классах. Класс, содержащий хотя бы одну чисто виртуальную функцию, становится абстрактным классом.  
  
3. Виртуальные функции отличаются от перегруженных функций тем, что выбор версии функции для вызова происходит на основе типа указателя или ссылки на объект, а не на основе типа самого объекта.  
  
4. Абстрактный класс - это класс, содержащий чисто виртуальные функции. Он не может быть использован для создания объектов напрямую, а служит только в качестве базового класса для производных классов.  
  
5. Класс делается абстрактным, чтобы определить интерфейс, который должны реализовывать его производные классы. Такой подход позволяет создать общий контракт для всех классов в иерархии и обеспечить единообразное поведение.  
  
6. Раннее связывание (статическое связывание) происходит во время компиляции и определяет, какая версия функции будет вызываться, исходя из типа указателя или ссылки на объект. Позднее связывание (динамическое связывание) происходит во время выполнения и позволяет вызывать соответствующую версию функции, определенную в объекте производного класса.  
  
7. Конструктор не может быть виртуальным, поскольку конструкторы вызываются во время создания объекта, когда тип объекта уже известен.  
  
8. Виртуальный деструктор используется, когда класс содержит виртуальные функции и предназначен для наследования. Он гарантирует, что вызывается правильный деструктор для объектов производных классов при удалении через указатель на базовый класс.

9. Чтобы защитить объект класса от копирования, можно объявить приватный конструктор копирования и оператор присваивания в классе.  
  
10. Ключевое слово "virtual" используется для объявления виртуальных функций в базовом классе. Оно позволяет производным классам переопределять эту функцию и обеспечивает позднее связывание вызовов функций.  
Ключевое слово "override" используется в производных классах для явного указания переопределения виртуальной функции из базового класса. Это помогает при компиляции обнаружить ошибки, связанные с неправильным переопределением функции.

# Листинг

#pragma once

#include <iostream>

#include <fstream>

using namespace std;

float checkFloat(float a);

int checkInt(int a);

class City {

string name;

int population;

public:

City(const string& name, int population) : name(name), population(population) {}

string getName() const {

return name;

}

int getPopulation() const {

return population;

}

};

class Shop {

string name;

string type;

public:

Shop(const string& name, const string& type) : name(name), type(type) {}

string getName() const {

return name;

}

string getType() const {

return type;

}

};

class Product {

string name;

string type;

int quantity;

double price;

public:

Product(const string& name, const string& type, int quantity, double price)

: name(name), type(type), quantity(quantity), price(price) {}

string getName() const {

return name;

}

string getType() const {

return type;

}

int getQuantity() const {

return quantity;

}

double getPrice() const {

return price;

}

};

class Bank {

string accountNumber;

double balance;

public:

Bank(const string& accountNumber, double balance) : accountNumber(accountNumber), balance(balance) {}

string getAccountNumber() const {

return accountNumber;

}

double getBalance() const {

return balance;

}

void setBalance(double balance) {

this->balance = balance;

}

};

class Buyer {

string lastName;

double money;

double purchaseAmount;

public:

Buyer(const string& lastName, double money, double purchaseAmount)

: lastName(lastName), money(money), purchaseAmount(purchaseAmount) {}

string getLastName() const {

return lastName;

}

double getMoney() const {

return money;

}

double getPurchaseAmount() const {

return purchaseAmount;

}

void makePurchase(const Product& product) {

if (product.getQuantity() > 0 && money >= product.getPrice()) {

purchaseAmount += product.getPrice();

money -= product.getPrice();

cout << "Purchase of product " << product.getName() << " was successful." << endl;

}

else {

cout << "Failed to make a purchase of product " << product.getName() << "." << endl;

}

}

void payThroughBank(Bank& bank) {

if (purchaseAmount <= money) {

money -= purchaseAmount;

bank.setBalance(bank.getBalance() + purchaseAmount);

purchaseAmount = 0;

cout << "Payment through the bank was successful." << endl;

}

else {

cout << "Failed to make a payment through the bank." << endl;

}

}

};  
  
  
#include"header.h"

float checkFloat(float a) {

while (true) {

cin >> a;

if (cin.fail()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "The input can only contain numbers. Try again." << endl;

continue;

}

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

if (cin.gcount() > 1) {

cout << "The input can only contain numbers. Try again." << endl;

continue;

}

if (a < 0) {

cout << "The input cannot contain negative numbers. Try again." << endl;

continue;

}

break;

}

return a;

}

int checkInt(int a) {

while (true) {

cin >> a;

if (cin.fail()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "The input can only contain numbers. Try again." << endl;

continue;

}

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

if (cin.gcount() > 1) {

cout << "The input can only contain numbers. Try again." << endl;

continue;

}

if (a < 0) {

cout << "The input cannot contain negative numbers. Try again." << endl;

continue;

}

break;

}

return a;

}

#include "header.h"

int main() {

const int maxObjects = 100;

int objectCount = 0;

City\*\* cities = new City \* [maxObjects];

Shop\*\* shops = new Shop \* [maxObjects];

Product\*\* products = new Product \* [maxObjects];

Bank\*\* banks = new Bank \* [maxObjects];

Buyer\*\* buyers = new Buyer \* [maxObjects];

ofstream outputFile("data.txt");

int choice = 0;

do {

cout << "Menu:" << endl;

cout << "1. Add a city" << endl;

cout << "2. Add a shop" << endl;

cout << "3. Add a product" << endl;

cout << "4. Add a bank" << endl;

cout << "5. Add a buyer" << endl;

cout << "6. Exit" << endl;

cout << "Choose an action: ";

choice = checkInt(choice);

switch (choice) {

case 1: {

if (objectCount < maxObjects) {

string name;

int population = 0;

cout << "Enter the city name: ";

cin >> name;

cout << "Enter the population: ";

population = checkInt(population);

cities[objectCount] = new City(name, population);

objectCount++;

cout << "City successfully added." << endl;

outputFile << cities[objectCount - 1]->getName() << "," << cities[objectCount - 1]->getPopulation() << endl;

}

else {

cout << "Maximum number of objects reached." << endl;

}

break;

}

case 2: {

if (objectCount < maxObjects) {

string name, type;

cout << "Enter the shop name: ";

cin >> name;

cout << "Enter the shop type: ";

cin >> type;

shops[objectCount] = new Shop(name, type);

objectCount++;

cout << "Shop successfully added." << endl;

outputFile << shops[objectCount - 1]->getName() << "," << shops[objectCount - 1]->getType() << endl;

}

else {

cout << "Maximum number of objects reached." << endl;

}

break;

}

case 3: {

if (objectCount < maxObjects) {

string name, type;

int quantity = 0;

double price = 0;

cout << "Enter the product name: ";

cin >> name;

cout << "Enter the product type: ";

cin >> type;

cout << "Enter the product quantity: ";

quantity = checkInt(quantity);

cout << "Enter the product price: ";

price = checkFloat(price);

products[objectCount] = new Product(name, type, quantity, price);

objectCount++;

cout << "Product successfully added." << endl;

outputFile << products[objectCount - 1]->getName() << "," << products[objectCount - 1]->getType() << ","

<< products[objectCount - 1]->getQuantity() << "," << products[objectCount - 1]->getPrice() << endl;

}

else {

cout << "Maximum number of objects reached." << endl;

}

break;

}

case 4: {

if (objectCount < maxObjects) {

string accountNumber;

double balance = 0;

cout << "Enter the account number: ";

cin >> accountNumber;

cout << "Enter the account balance: ";

balance = checkFloat(balance);

banks[objectCount] = new Bank(accountNumber, balance);

objectCount++;

cout << "Bank successfully added." << endl;

outputFile << banks[objectCount - 1]->getAccountNumber() << "," << banks[objectCount - 1]->getBalance() << endl;

}

else {

cout << "Maximum number of objects reached." << endl;

}

break;

}

case 5: {

if (objectCount < maxObjects) {

string lastName;

double money = 0, purchaseAmount = 0;

cout << "Enter the buyer's last name: ";

cin >> lastName;

cout << "Enter the buyer's money: ";

money = checkFloat(money);

cout << "Enter the buyer's purchase amount: ";

purchaseAmount = checkFloat(purchaseAmount);

buyers[objectCount] = new Buyer(lastName, money, purchaseAmount);

objectCount++;

cout << "Buyer successfully added." << endl;

outputFile << buyers[objectCount - 1]->getLastName() << "," << buyers[objectCount - 1]->getMoney() << ","

<< buyers[objectCount - 1]->getPurchaseAmount() << endl;

}

else {

cout << "Maximum number of objects reached." << endl;

}

break;

}

case 6: {

break;

}

default: {

cout << "Invalid choice. Please try again." << endl;

break;

}

}

} while (choice != 6);

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

delete cities[i];

delete shops[i];

delete products[i];

delete banks[i];

delete buyers[i];

}

delete[] cities;

delete[] shops;

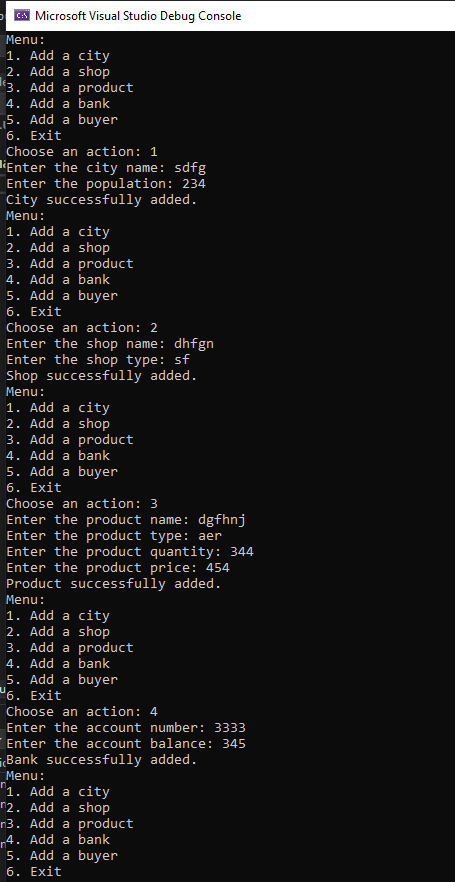
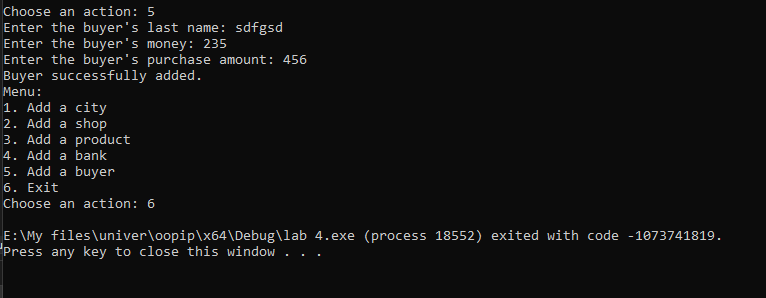
delete[] products;

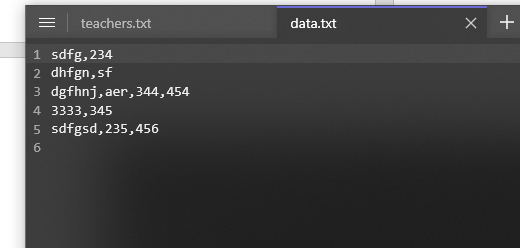
delete[] banks;

delete[] buyers;

return 0;

}



# Выводы

В ходе работы было изучено реализация принципа динамического и статического полиморфизма, особенности использования перегрузки операторов, методов и переопределения функций.