**1 ЛАБОРАТОРНАЯ РАБОТА №1**

**«ИССЛЕДОВАНИЕ ОСОБЕННОСТЕЙ ОБЪЕКТНОГО ПОДХОДА ПРИ ПРОГРАММИРОВАНИИ СЛОЖНЫХ СТРУКТУР ДАННЫХ С ДИНАМИЧЕСКИМИ ПОЛЯМИ»**

**1.1 Цель работы**

Исследование основных средств определения класса, создания объектов класса, приобретение навыков разработки и отладки программ, использующих динамическую память. Исследование особенностей использования конструкторов копирования.

**1.2 Вариант задания – 8**

Требуется для динамической структуры — циклической очереди, хранящей информацию о штрафах: номер автомобиля (строка символов 8-9) и величина штрафа (цифра), описать класс, содержащий указатель на динамический тип как поле данных. Для этого класса описать конструкторы (не менее трех, в том числе и конструктор копирования), деструктор, функцию печати данных. Создать экземпляр полученного класса и проиллюстрировать его корректную работу: распечатать данные, изменить данные и распечатать вновь. Создать второй экземпляр класса как копию первого и проиллюстрировать корректную работу конструктора копирования: распечатать и изменить данные объекта-копии, распечатать данные обоих объектов, сравнить результат. Предусмотреть обработку ошибок при манипуляции с данными. Также предусмотреть функции добавления элементов в очередь и удаления из нее, а также функцию вычисления величины суммы штрафов со всех авто.

**1.3 Ход работы**

1.3.1 Программа с помощью «Меню» и switch выполняет различные действия с двумя объектами класса Panalties. В классе описаны 4 метода, 3 конструктора и один деструктор. С помощью конструкторов можно создать пустую очередь, очередь с одним элементом или скопировать одну очередь в другую. С помощью методов можно добавить элемент в какую либо очередь, удалить элемент, вывести данные содержащиеся в очереди и подсчитать количество штрафов в очереди.

1.3.2 Написана программа на С++ согласно вышеописанного алгоритма.

#include <iostream>  
#include <windows.h>  
#include <iomanip>  
#include <cstring>  
  
using namespace std;  
  
//одна запись  
struct Node{  
 char numAuto[9]; //номер авто  
 int pricePenalty; //стоимость штрафа  
 struct Node \*next; //указатель на след. элемент  
};  
  
//штрафы  
class Penalties {  
private:  
 Node \*begin; //голова очереди  
 Node \*end; //хвост очереди  
  
public:  
 Penalties(); //1 конструктор - создать очередь с первым элементом имеющим стандартные значения  
 Penalties(char num[9], int price); //2 конструктор - создание очереди с одним элементом  
 Penalties(const Penalties &queue); //3 конструктор - копирования  
 ~Penalties (); //деструктор  
  
 void addElement(char num[9], int price); // Добавление элемента в список  
 void DeleteElement(); // Удалить элемент из очереди  
 long sumPenalties(); // Сумма штрафов  
 void show(); // Отображение очереди  
};  
  
  
//1 конструктор - создать очередь с первым элементом имеющим стандартные значения  
Penalties::Penalties() {  
 Node \*newE = new Node; //создать новый элемент и выделить под него память  
 newE->pricePenalty = 0;  
 char num[9] = "--------";  
 strcpy(newE->numAuto, num);  
  
 begin = newE;  
 end = newE;  
 end->next = begin;  
  
 this->DeleteElement();  
 cout << endl << "(1 конструктор выполнен)" << endl << endl;  
}  
  
  
//2 конструктор - создание очереди с одним элементом  
Penalties::Penalties(char num[9], int price){  
 Node \*newE = new Node; //создать новый элемент и выделить под него память  
 newE->pricePenalty = price;  
 strcpy(newE->numAuto, num);  
  
 begin = newE;  
 end = newE;  
 end->next = begin;  
  
 cout << endl << "(2 конструктор выполнен)" << endl << endl;  
}  
  
  
//3 конструктор - копирования  
Penalties::Penalties(const Penalties &queue) {  
 //если нет данных в исходном объекте  
 if (!queue.begin) {  
 Node \*newE = new Node; //создать новый элемент и выделить под него память  
 newE->pricePenalty = 0;  
 char num[9] = "--------";  
 strcpy(newE->numAuto, num);  
  
 this->begin = newE;  
 this->end = newE;  
 this->end->next = this->begin;  
 }  
 else {  
 //добавить первый элемент  
 Node \*newE = new Node; //создать новый элемент и выделить под него память  
 newE->pricePenalty = queue.begin->pricePenalty;  
 strcpy(newE->numAuto, queue.begin->numAuto);  
  
 this->begin = newE;  
 this->end = newE;  
 this->end->next = this->begin;  
  
 //если в циклической очереди больше одного элемента  
 if (queue.begin->next != queue.begin) {  
 Node \*temp = queue.begin->next;  
 Node \*prev = this->begin;  
 while (temp != queue.begin) {  
 Node \*newE = new Node; //создать новый элемент и выделить под него память  
 newE->pricePenalty = temp->pricePenalty;  
 strcpy(newE->numAuto, temp->numAuto);  
  
 prev->next = newE;  
 this->end = newE;  
 newE->next = this->begin;  
 temp = temp->next;  
 prev = end;  
 }  
 }  
 }  
 cout << endl << "(конструктор копирования выполнен)" << endl << endl;  
}  
  
  
//деструктор  
Penalties::~Penalties()  
{  
 Node \*temp = begin;  
 while(begin != end) // До тех пор, пока головной элемент не равен хвостовому  
 {  
 temp = begin;  
 begin = begin->next;  
 delete temp;  
 }  
 delete end;  
  
 cout << endl << "деструктор выполнен" << endl << endl;  
}  
  
  
// Добавление элемента в список  
void Penalties::addElement(char num[9], int price) {  
 Node \*newE = new Node; //создать новый элемент и выделить под него память  
 newE->pricePenalty = price;  
 strcpy(newE->numAuto, num);  
  
 if (begin == NULL) {  
 begin = newE;  
 end = newE;  
 end->next = begin;  
 }  
 else {  
 newE->next = begin;  
 begin = newE;  
 end->next = begin;  
 }  
}  
  
  
// Удалить элемент из очереди  
void Penalties::DeleteElement() {  
 if (!begin) {  
 cout << "Удалять нечего" << endl;  
 return;  
 }  
 else if (begin->next == begin) { //если в списке один элемент  
 delete begin;  
 begin = end = NULL;  
 }  
 else{  
 Node \*temp = begin;  
 while (temp->next != end)  
 temp = temp->next;  
 delete end;  
 end = temp;  
 end->next = begin;  
 }  
}  
  
  
// Сумма штрафов  
long Penalties::sumPenalties() {  
 long sum = 0;  
 if (!begin) {  
 return 0;  
 }  
 Node \*temp = begin;  
 while(temp != end) {  
 sum += temp->pricePenalty;  
 temp = temp->next;  
 }  
 sum += temp->pricePenalty;  
 return sum;  
}  
  
  
// Отображение очереди  
void Penalties::show()  
{  
 if(!begin) {  
 cout << "Очередь отсутствует" << endl << endl;  
 return;  
 }  
 Node \*temp = begin;  
  
 cout << "-------------------------------" << endl;  
 cout << "| № Авто | Стоимость штрафа |" << endl;  
 cout << "-------------------------------" << endl;  
 while(temp != end)  
 {  
 cout << "| " << setw(8) << left << temp->numAuto << " | " << setw(16) << left << temp->pricePenalty << " |" << endl;  
 cout << "-------------------------------" << endl;  
 temp = temp->next;  
 }  
 cout << "| " << setw(8) << left << temp->numAuto << " | " << setw(16) << left << temp->pricePenalty << " |" << endl;  
 cout << "-------------------------------" << endl << endl;  
}  
  
  
//основная программа  
int main() {  
 SetConsoleCP(1251);  
 SetConsoleOutputCP(1251);  
  
 int menu;  
 char numAuto[9];  
 int penalty\_price;  
 Penalties \*queue\_1 = new Penalties(), \*queue\_2 = new Penalties();  
  
  
 while (1) {  
 cout << endl;  
 cout << "1 - Добавить элемент в очередь" << endl;  
 cout << "2 - Удалить элемент из очереди" << endl;  
 cout << "3 - Демонстрация очереди" << endl;  
 cout << "4 - Сумма штрафов в очереди" << endl;  
 cout << "5 - Копировать копию первой очереди (очередь\_2)" << endl;  
 cout << "6 - Выход из программы" << endl;  
 cout << " Введите пункт меню ->";  
 cin >> menu; cout << endl;  
  
 switch (menu) {  
  
 case 1 : {  
 cout << "C какой очередью работаем? (1-первой / 2-второй) ->"; cin >> menu;  
 cout << endl;  
 if (menu == 1) {  
 cout << "Введите номер авто (8 символов) ->";  
 cin >> numAuto;  
 cout << "Введите сумму штрафа ->";  
 cin >> penalty\_price;  
 cout << endl;  
 queue\_1->addElement(numAuto, penalty\_price);  
 }  
 else {  
 cout << "Введите номер авто (8 символов) ->";  
 cin >> numAuto;  
 cout << "Введите сумму штрафа ->";  
 cin >> penalty\_price;  
 cout << endl;  
 queue\_2->addElement(numAuto, penalty\_price);  
 }  
 break;  
 }  
  
 case 2 : {  
 cout << "C какой очередью работаем? (1-первой / 2-второй) ->"; cin >> menu;  
 cout << endl;  
 if (menu == 1) {  
 queue\_1->DeleteElement();  
 }  
 else {  
 queue\_2->DeleteElement();  
 }  
 break;  
 }  
  
 case 3 : {  
 cout << "C какой очередью работаем? (1-первой / 2-второй) ->"; cin >> menu;  
 cout << endl;  
 if (menu == 1) {  
 queue\_1->show();  
 }  
 else {  
 queue\_2->show();  
 }  
 break;  
 }  
  
 case 4 : {  
 cout << "C какой очередью работаем? (1-первой / 2-второй) ->"; cin >> menu;  
 cout << endl;  
 if (menu == 1) {  
 cout << "Cумма штрафов в первой очереди = " <<  
 queue\_1->sumPenalties() << endl;  
 }  
 else {  
 cout << "Cумма штрафов в первой очереди = " <<  
 queue\_2->sumPenalties() << endl;  
 }  
 break;  
 }  
  
 case 5 : {  
 cout << "Кого куда копируем?" << endl << "1 - первую очередь копируем во вторую" << endl << "2 - вторую очередь копируем в первую" << endl;  
 cin >> menu; cout << endl;  
 if (menu == 1) {  
 delete queue\_2;  
 queue\_2 = new Penalties(\*queue\_1);  
 }  
 else {  
 delete queue\_1;  
 queue\_1 = new Penalties(\*queue\_2);  
 }  
 break;  
 }  
  
 case 6 : {  
 cout << endl << "Выход из программы" << endl << endl;  
 return 0;  
 }  
 }  
 }  
}

1.3.3 Выполнена отладка программы.

Результаты тестирования отображены на рисунках 1.1- 1.5.

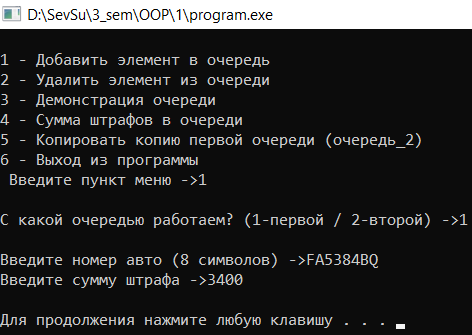


Рисунок 1.1 – Добавление элемента в первую очередь

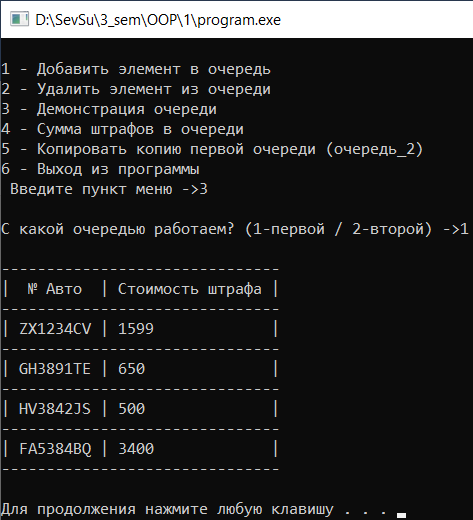


Рисунок 1.2 – Демонстрация первой очереди

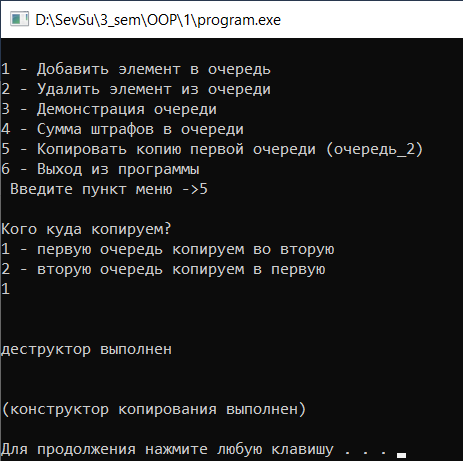


Рисунок 1.3 – Копирование первой очереди во вторую

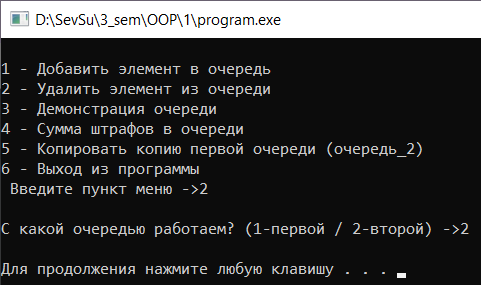


Рисунок 1.4 – Удаление элемента из второй очереди

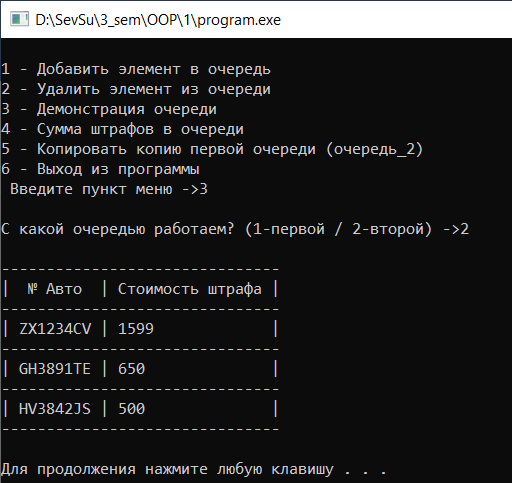


Рисунок 1.5 – Демонстрация второй очереди

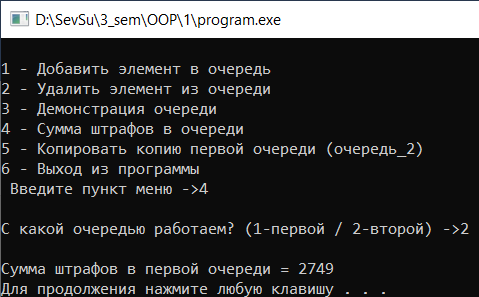


Рисунок 1.6 – Сумма штрафов во второй очереди

Результаты тестирования полностью соответствуют ожиданиям.

**Выводы**

В ходе выполнения данной лабораторной работы были получены навыки разработки программ, использующих классы и объекты. Были изучены способы создания объектов, описания классов, обращения к объектам. Закреплены навыки разработки и отладки программ, использующих динамическую память. Полученные во время разработки навыки помогут разрабатывать более сложные программы с использованием классов и объектов, более эффективные по времени выполнения алгоритмы.