МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт Информационных Технологий

Кафедра МПО ЭВМ

Дисциплина «Объектно-ориентированное программирование»

Лабораторная работа №2

«Разработка абстрактных типов данных»

Выполнил:

студент группы 1ПИб-02-3оп-22

Маркелов Сергей Александрович

Проверил:

Табунов Павел Александрович

Череповец, 2023 год

**Цель работы:**

* изучить способы определения классов, правила доступа к элементам;
* приобрести практические навыки работы с объектами класса;
* изучить принципы и механизмы создания абстрактных типов данных;

**Требования к разработке:**

1. Каждый класс должен быть оформлен в отдельных файлах: заголовочный (.h) и файл с кодом (.cpp).
2. Запрещается использовать обработку исключительных ситуаций и генерировать исключения.
3. Придерживаться принципа DRY (Don’t repeat yourself).
4. Обязательно наличие комментариев.

**Задание на лабораторную работу:**

Разработать АТД – класс, обеспечивающий хранение объектов класса из лабораторной работы №1, согласно варианту задания: очередь

Интерфейс класса может содержать конструкторы и должен содержать деструктор

Интерфейс класса должен содержать функции:

* Добавления
* Удаления
* Вывода содержимого
* Обработки (сортировка по полю класса, поиск по условию)

У класса-контейнера не должно быть прямого доступа к полям класса предметной области из лабораторной работы №1.

**Текст программы**

Заголовочный файл **Queue.h**:

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

using namespace std;

class Queue {

private:

struct Node { //узел очереди

string data;

Node\* next;

};

Node\* head;

Node\* tail;

public:

Queue() { //конструктор

head = NULL;

tail = NULL;

}

bool empty() { //проверка очереди на пустоту

return head == NULL;

}

void add(string value) { //добавление элемента в очередь

if (empty()) { //если она пустая

head = new Node;

head->data = value;

head->next = NULL;

tail = head;

}

else { //если она не пустая

tail->next = new Node;

tail = tail->next;

tail->data = value;

tail->next = NULL;

}

}

string del() { //взятие и удаление элемента из очереди

if (empty()) {

cout << "Queue is empty" << endl;

}

else {

string d = head->data;

Node\* tmp = head;

head = head->next;

delete(tmp);

return d;

}

}

void nullQueue() { //обнуление очереди

Node\* tmp;

while (!empty()) {

tmp = head;

head = head->next;

delete(tmp);

}

}

void print\_OldNew() {

while (!empty())

cout << del() << endl;

}

void print\_NewOld(int count) {

string\* networks = new string[count];

int i = 0;

while (!empty()) {

networks[i] = del();

i++;

}

i--;

for (i; i >= 0; i--)

cout << networks[i] << endl;

}

void print\_AZ(int count) {

string\* networks = new string[count];

int i = 0;

while (!empty()) {

networks[i] = del();

i++;

}

string k;

for (i = 0; i < count; i++)

for (int j = i; j < count; j++) {

if (networks[i] > networks[j]) {

k = networks[i];

networks[i] = networks[j];

networks[j] = k;

}

}

for (i = 0; i < count; i++)

cout << networks[i] << endl;

}

void print\_ZA(int count) {

string\* networks = new string[count];

int i = 0;

while (!empty()) {

networks[i] = del();

i++;

}

string k;

for (i = 0; i < count; i++)

for (int j = i; j < count; j++) {

if (networks[i] < networks[j]) {

k = networks[i];

networks[i] = networks[j];

networks[j] = k;

}

}

for (i = 0; i < count; i++)

cout << networks[i] << endl;

}

void search(string name, int count) {

string\* networks = new string[count];

int i = 0;

while (!empty()) {

networks[i] = del();

i++;

}

int k2 = 0;

for (i = 0; i < count; i++) {

int k1 = 0;

string s;

for (int l = 0; l < name.length(); l++) {

s = networks[i];

if (s[l] == name[l]) k1++;

}

if (k1 == name.length()) {

if (k2 == 0) cout << endl << "Найденные сети:" << endl;

cout << networks[i] << endl;

k2++;

}

}

if (k2 == 0) cout << "Ничего не найдено!" << endl;

}

~Queue() { //деструктор

}

};

Заголовочный файл **Header.h**:

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

using namespace std;

enum security { WEP, WPA, WPA2, WPA3, NoProtection };

class WiFi {

private:

string login;

string password;

static string work;

security protection;

int speed, radius;

double frequency;

public:

WiFi(string valueLogin, string valuePassword, security valueProtection, int valueSpeed, int valueRadius, double valueFrequency) {

this->login = valueLogin; //конструктор

this->password = valuePassword;

this->protection = valueProtection;

this->speed = valueSpeed;

this->radius = valueRadius;

this->frequency = valueFrequency;

this->work = "Да";

}

WiFi() {

this->login = "Неопределенная сеть"; //конструктор

this->password = "0";

this->protection = NoProtection;

this->speed = 0;

this->radius = 0;

this->frequency = 0;

this->work = "Да";

}

WiFi(double valueFrequency, string valueLogin) {

this->login = valueLogin; //конструктор

this->password = "0";

this->protection = NoProtection;

this->speed = 0;

this->radius = 0;

this->frequency = valueFrequency;

this->work = "Да";

}

WiFi(int valueSpeed, string valueLogin) {

this->login = valueLogin; //конструктор

this->password = "0";

this->protection = NoProtection;

this->speed = valueSpeed;

this->radius = 0;

this->frequency = 0;

this->work = "Да";

}

WiFi(const WiFi& obj) {

this->login = obj.login; //конструктор

this->password = obj.password;

this->protection = obj.protection;

this->speed = obj.speed;

this->radius = obj.radius;

this->frequency = obj.frequency;

this->work = "Да";

}

string getLogin() { //геттер для логина

return login;

}

string getPassword() { //геттер для пароля

return password;

}

string getProtection() { //геттер для протокола безопасности

switch (protection) {

case WEP: return "WEP";

case WPA: return "WPA";

case WPA2: return "WPA2";

case WPA3: return "WPA3";

case NoProtection: return "Без защиты";

default: return "N/A";

}

}

void networkInfo() { //вывод информации о сети

cout << "Вы успешно подключились к WiFi-сети " << login << endl << endl;

cout << "Информация о сети" << endl;

cout << "Скорость: " << speed << " Мбит/с" << endl;

cout << "Частота: " << frequency << " ГГц" << endl;

cout << "Радиус покрытия: " << radius << " м" << endl;

cout << "Протокол безопасности: " << getProtection() << endl;

cout << "Cеть работает: " << work << endl;

cout << endl << "=======================" << endl << endl;

}

~WiFi() { //деструктор

}

};

Файл с кодом лб1.cpp:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

#include "Header.h"

#include "Queue.h"

using namespace std;

string WiFi::work;

void NetworkError(int err) { //функция вывода ошибок

switch (err) {

case 1: cout << "Такой WiFi-сети нет!";

break;

case 2: cout << "Пароль неверный!" << endl;

break;

case 3: cout << "Такого варианта нет!" << endl;

break;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int number, count = 0;

string password, name;

Queue q;

while (true) {

WiFi Network\_1("CHSU\_FREE", "0", security::NoProtection, 100, 70, 5); //описание доступных WiFi-сетей

WiFi Network\_2("CHSU\_WORK", "1234567890", security::WPA3, 120, 50, 5);

WiFi Network\_3("IPhone", "ghfdgldklkl", security::WPA2, 80, 40, 2.4);

WiFi Network\_4("Xiaomi", "h5hjg54g5", security::WPA2, 70, 40, 2.4);

WiFi Network\_5;

WiFi Network\_6(Network\_1);

WiFi Network\_7(2.4, "Rostelecom");

WiFi Network\_8(100, "Beeline");

cout << "Доступные сети: " << endl; //вывод доступных сетей на экран

cout << "1. " << Network\_1.getLogin() << endl;

cout << "2. " << Network\_2.getLogin() << endl;

cout << "3. " << Network\_3.getLogin() << endl;

cout << "4. " << Network\_4.getLogin() << endl;

cout << "5. " << Network\_5.getLogin() << endl;

cout << "6. " << Network\_6.getLogin() << endl;

cout << "7. " << Network\_7.getLogin() << endl;

cout << "8. " << Network\_8.getLogin() << endl << endl;

cout << "0. " << "Вывести историю подключения к сетям" << endl;

cout << endl << "Введите номер сети, к которой хотите подключиться: "; //выбор сети

cin >> number;

if (number < 0 || number > 8) { //если введен номер несуществуещей сети, выводится сообщение об ошибке

NetworkError(1);

return -1;

}

else if (number == 0 && !q.empty()) {

int num;

cout << endl << "Порядок вывода:" << endl; //вывод доступных сетей на экран

cout << "1. " << "В хронологическом (от старых подключений к новым)" << endl;

cout << "2. " << "В хронологическом (от новых подключений к старым)" << endl;

cout << "3. " << "В алфавитном (A - Z)" << endl;

cout << "4. " << "В обратном алфавитному (Z - A)" << endl;

cout << "5. " << "Поиск по названию" << endl;

cout << endl << "Выберите порядок: ";

cin >> num;

if (num < 1 || num > 5) {

NetworkError(3);

return -1;

}

if (num >=1 && num <= 4) cout << endl << "История подключений: " << endl;

switch (num) {

case 1: q.print\_OldNew();

break;

case 2: q.print\_NewOld(count);

break;

case 3: q.print\_AZ(count);

break;

case 4: q.print\_ZA(count);

break;

case 5: {

cout << endl << "Введите название сети: ";

string name;

cin >> name;

cin.get();

q.search(name, count);

}

break;

}

return 0;

}

else if (number == 0 && q.empty()) {

cout << "Подключений не было" << endl;

return 0;

}

cout << "Введите пароль: "; //ввод пароля

cin >> password;

cin.get();

switch (number) {

case 1: {

q.add(Network\_1.getLogin());

count++;

if (password == Network\_1.getPassword()) Network\_1.networkInfo(); //если пароль верный, то выводится информация о сети

else {

NetworkError(2); //если пароль неверный, выводится сообщение об ошибке

return -1;

}

}

break;

case 2: {

q.add(Network\_2.getLogin());

count++;

if (password == Network\_2.getPassword()) Network\_2.networkInfo();

else {

NetworkError(2);

return -1;

}

}

break;

case 3: {

q.add(Network\_3.getLogin());

count++;

if (password == Network\_3.getPassword()) Network\_3.networkInfo();

else {

NetworkError(2);

return -1;

}

}

break;

case 4: {

q.add(Network\_4.getLogin());

count++;

if (password == Network\_4.getPassword()) Network\_4.networkInfo();

else {

NetworkError(2);

return -1;

}

}

break;

case 5: {

q.add(Network\_5.getLogin());

count++;

if (password == Network\_5.getPassword()) Network\_5.networkInfo();

else {

NetworkError(2);

return -1;

}

}

break;

case 6: {

q.add(Network\_6.getLogin());

count++;

if (password == Network\_6.getPassword()) Network\_6.networkInfo();

else {

NetworkError(2);

return -1;

}

}

break;

case 7: {

q.add(Network\_7.getLogin());

count++;

if (password == Network\_7.getPassword()) Network\_7.networkInfo();

else {

NetworkError(2);

return -1;

}

}

break;

case 8: {

q.add(Network\_8.getLogin());

count++;

if (password == Network\_8.getPassword()) Network\_8.networkInfo();

else {

NetworkError(2);

return -1;

}

}

break;

}

}

system("pause");

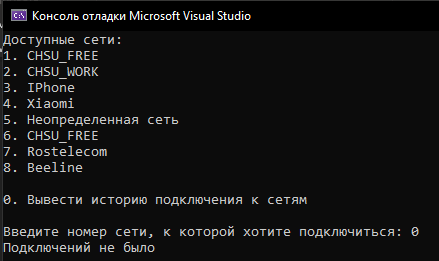
return 0;

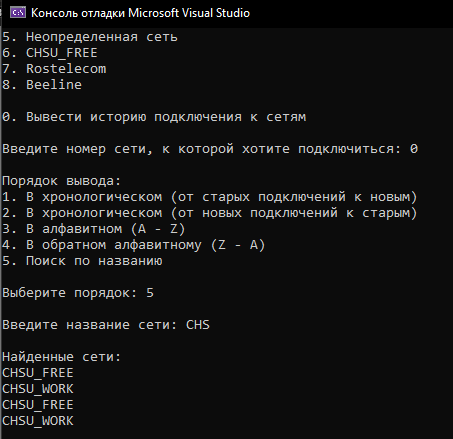
}

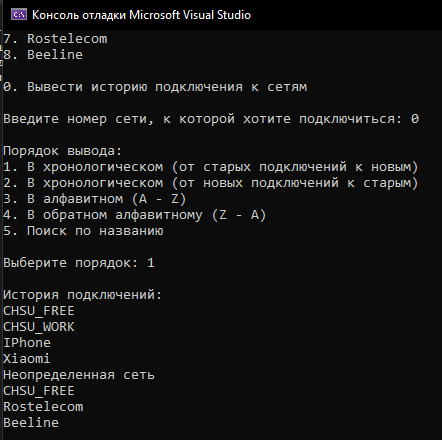
**Описание программы**

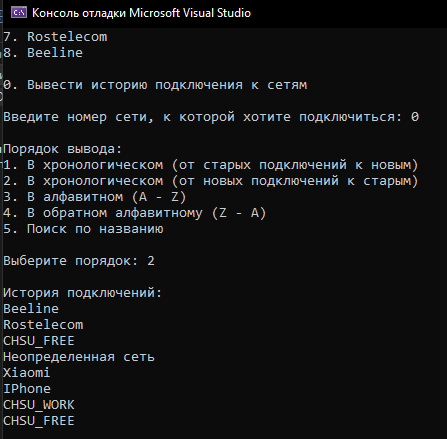
1. Определяется статическая переменная **WiFi::work**
2. Задается функция **NetworkError** для обработки ошибок:
   * Если **err = 1**, на экран выводится фраза «*Такой WiFi-сети нет!*»
   * Если **err = 2**, на экран выводится фраза «*Пароль неверный!*»
   * Если **err = 3**, на экран выводится фраза «*Такого варианта нет!*»
3. Инициализируются целочисленные переменные **number** и **const**, а также строки **password** и **name**. Переменной **const** присваивается значение 0.
4. С помощью конструктора по умолчанию создается объект **q** класса **Queue** (очередь).
5. Вызывается конструктор класса **WiFi** для объектов **Network\_1**, **Network\_2**, **Network\_3**, **Network\_4**, **Network\_5**, **Network\_6**, **Network\_7**, **Network\_8**
   * Класс **WiFi** содержит поля **login**, **password**, **work**, **protection**, **speed**, **radius**, **frequency**
   * Конструктор инициализирует поля **login**, **password**, **protection**, **speed**, **radius**, **frequency** как параметры, полю **work** присваивается значение «*Да*»
6. На экран выводятся доступные сети (значение поля **login**) с помощью геттера **getLogin**
7. Программа просит пользователя ввести значение переменной **number** – номер сети. Если введен 0 – тогда программа выведет историю подключения к сетям
8. Если пользователь ввел значение меньше 0 или больше 8 (т.е. указал номер несуществующей сети), вызывается функция **NetworkError** с параметром **err = 1**, затем выполнение программы завершается
9. Если пользователь ввел 0, чтобы узнать историю подключений к сети, и если очередь **q** не пуста (т.е. подключения были), то программа попросит пользователя ввести значение переменной **num**, чтобы выбрать сортировку сетей.
10. Если пользователь ввел значение меньше 1 или больше 5 (т.е. указал несуществующий вариант сортировки), вызывается функция **NetworkError** с параметром **err = 3**, затем выполнение программы завершается
11. Программа выполняет выбранный вариант сортировки с помощью функций **print\_OldNew** (от старых к новым), **print\_NewOld** (от новых к старым), **print\_AZ** (от A до Z), **print\_ZA** (от Z до A). Если **num = 5** (т. е. пользователь выбрал поиск сети), тогда программа запрашивает у пользователя поисковый запрос названия сети и записывает его в строку name. Затем, с помощью функции **search**, программа находит все сети, которые начинаются с введенной строкой (или полностью с ней совпадают).
12. Если пользователь ввел 0, а очередь была пуста (т.е. подключений еще не было), то программа выводит на экран фразу «Подключений не было!»
13. Если пользователь выбрал одну из сетей, программа просит пользователя ввести пароль (в строку **password**)
14. Если значение переменной password равно значению поля password класса **WiFi** (для сравнения вызывается геттер **getPassword**) вызывается функция **NetworkInfo**, которая выводит значения всех полей, а значение поля login записывается в очередь **q**.
15. Если значение переменной **password** не равно значению поля **password** класса **WiFi**, вызывается функция **NetworkError** с параметром **err = 2**, затем выполнение программы завершается
16. Вызываются деструкторы классов **~Queue** и **~WiFi**, которые уничтожают классы **Queue** и **WiFi**

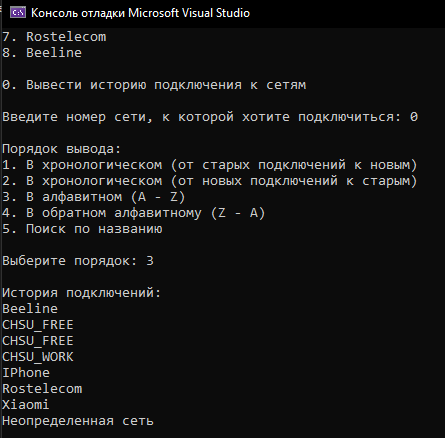
**Результаты тестов**

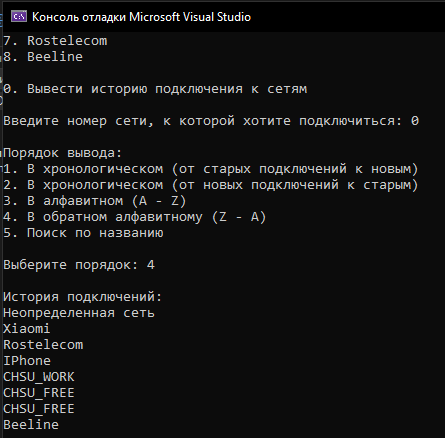












**Контрольные вопросы**

1. *Каковы особенности синтаксиса и семантики конструкторов и деструкторов?*

У конструктора нет явного типа возвращаемого значения. Имя конструктора совпадает с именем класса. В классе можно объявлять несколько конструкторов с различными параметрами (перегрузка конструкторов). Конструкторы используются для инициализации членов класса.

Имя деструктора совпадает с именем класса, но предваряется тильдой (~). Деструкторы используются для освобождения ресурсов, выделенных объекту в течение его жизненного цикла. Деструктор вызывается автоматически при выходе объекта за пределы области видимости или при вызове оператора delete для динамически выделенных объектов. Если не определить свой деструктор в классе, компилятор использует неявный деструктор по умолчанию.

1. *В чем заключаются особенности применения конструкторов по умолчанию?*

Конструктор по умолчанию – это конструктор, который не принимает никаких аргументов. Он автоматически создается компилятором, если в классе не определено ни одного конструктора.

Конструктор по умолчанию используется для инициализации членов класса значениями по умолчанию. Он вызывается, когда объект создается без передачи аргументов.

Если в классе определен хотя бы один конструктор, конструктор по умолчанию не будет автоматически создан компилятором. Это приведет к ошибке, если объект будет создаваться без передачи аргументов. Однако, конструктор по умолчанию можно явно определить в классе.

1. *В каких случаях вызывается копирующий конструктор?*

Копирующий конструктор используется для создания нового объекта, который является копией существующего объекта. Копирующий конструктор вызывается в следующих случаях:

* когда объекту присваивается значение другого объекта того же типа
* когда объект передается в функцию в качестве параметра по значению
* когда объект возвращается из функции по значению
* когда новый объект инициализируется существующим объектом того же типа

1. *При каких условиях конструктор может использоваться для преобразования типа?*

Преобразование типов при помощи конструктора в C++ может происходить в нескольких случаях:

* Конструктор с одним параметром может быть использован для неявного преобразования типа
* Конструктор с одним параметром может быть использован для явного преобразования типа с использованием оператора приведения типа
* Конструкторы могут использоваться для определения пользовательских преобразований между различными классами

1. *В каких ситуациях необходимы инициализаторы конструктора, синтаксис и семантика их использования?*

Инициализаторы конструктора используются для инициализации членов класса в теле конструктора.

Инициализаторы конструктора могут присваивать значения константным членам данных при создании объекта.

Инициализаторы конструктора позволяют использовать значения других членов класса для инициализации.

Инициализаторы конструктора позволяют указать порядок инициализации членов класса, что может быть важным для правильного функционирования класса

Инициализаторы конструктора облегчают инициализацию членов класса с нестандартными значениями по умолчанию.

Инициализаторы конструктора включаются в список инициализации конструктора и выглядят как часть его объявления. Они размещаются после двоеточия, а затем идет перечисление членов класса с соответствующими значениями.

1. *Явные и неявные вызовы конструкторов и деструктора*

Вызовы конструкторов и деструкторов могут быть явными и неявными.

Явный вызов конструктора происходит, когда объект создается с использованием оператора **new**:

**Class\* obj = new Class;**

Также явный вызов происходит, при вызове конструктора класса в коде, например, при создании временного объекта:

**Class obj = Class();**

Неявный вызов конструктора происходит, когда объект создается без использования оператора new:

**Class obj;**

Также при выделении массива объектов конструктор вызывается неявным образом для каждого элемента массива:

**Class\* objArray = new Class[5];**

Явный вызов деструктора происходит, когда объект, созданный с помощью оператора new, уничтожается с помощью оператора **delete**:

**Class\* obj = new Class;**

**delete obj;**

Также возможен другой вариант явного вызова деструктора в коде:

**obj.~MyClass();**

При выходе из области видимости или при удалении объекта, созданного на стеке, деструктор вызывается неявным образом.

При удалении массива объектов с использованием оператора **delete[]** вызывается деструктор для каждого элемента массива.

**delete[] objArray;**

**Вывод**

В ходе выполнения лабораторной работы были изучены способы определения классов, правила доступа к элементам, принципы и механизмы создания абстрактных типов данных, синтаксис и семантика определения и вызова конструкторов и деструкторов, особенности применения различных видов конструкторов. Также были приобретены практические навыки работы с объектами класса, создания и уничтожения объектов.

Были разработаны алгоритм и программа, реализующие абстрактный тип данных (АТД) – класс, обеспечивающий хранение объектов класса из лабораторной работы №1 в очереди.

Интерфейс класса содержит функции добавления, удаления, вывода содержимого и обработки (сортировка по полю класса, поиск по условию).