**кафедра комп’ютерних технологій, ДНУ**

**2016/2017 н.р.**

**ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №4**

**за курсом «Програмування»**

**студента групи ПК-15-1**

**Гулого Тараса Олександровича**

# 1. Постановка задачі

**Постановка задачі**.

Розробити програму – консольний застосунок для ОС Windows, що демонструє можливості мови С++, а також його стандартної бібліотеки по роботі з контейнерами та ітераторами.

Використання всіх вищезгаданих можливостей є обов’язковим в кожному варіантів індивідуального завдання.

В кожному з варіантів завдань Вами повинні бути реалізовані основні можливості бібліотеки стандартних шаблонів: ітератор(и), пошук, упорядкування, сума або щось аналогічне зі згоди викладача.

При написанні слід враховувати необхідність забезпечення зборки у двох режимах:

1. Створення виконуваного файлу із підключенням/використанням типів і алгоритмів бібліотеки стандартних шаблонів(STL).
2. Створення виконуваного файлу без застосування STL.

**Індивідуальне завдання №8**

Визначити чергу використовуючи (тільки) два стеки

# 2. Опис розв’язку

**Основна концепція розв’язку**

Черга – структура-адаптер послідовності, яка є елементом бібліотеки стандартних шаблонів. Особливістю черги є те що прямий доступ можливий лише до першого та останнього елементів всі інші є недосяжними доти. Всі елементи які додаються ідуть «в кінець черги» тобто розміщенні у черзі у порядку їх внесення. Основними дієвими елементами у стандартній черзі є функції push і pop, які вносять елемент в кінець черги і видаляють звідти перший відповідно. Серед інших функцій size (кількість елементів у черзі), front(доступ до першого елементу) back(доступ до останнього елементу) empty (перевірка наявності елементів в черзі).

На відміну від черги стек дає доступ і видалення відбувається з кінця. Ці дві структури відрізняються порядком обробки («First-in-first-out» у черги і «First-in-last-out» у стека)

Принцип визначення черги за допомогою двох стеків досить простий. Пояснення на прикладі:

Уявимо що на підлозі лежить стос скляних вікон. Нам необхідно взяти з цього стосу перше вікно. Прямий доступ до вікна не можливий з декількох причин (на ньому лежить велика вага, можливе пошкодження інших вікон зі стосу і. т. д). Для того щоб дістатися до першого вікна ми будемо виймати вікна з стосу почергово і перекладати їх на вільне місце(тим самим утворюючи новий стос).

Pop()

Top()

Push()

Stack1

Stack2

«Прямий доступ»

Доступ на основі 2 стосів

На даному прикладі видно, що на основі двох стосів можна організувати доступ як і у черзі. Отже, у класі черга необхідно мати функції які будуть робити перенесення із першого стеку до другого. Всі інші функції визначаються автоматично(кінці черги це верхній елемент першого стосу перший елемент, це верхній елемент другого стосу, розмір та наявність елементів визначається як сума розмірів стеків та відсутність елементів у кожному з них).

**Шаблонність класу «Черга» і перемикач з SLT/без SLT**

Базуючись на прикладі представлення принципу шаблонності шаблон класу «Черга» може бути інстанційований з будь якими типами внутрішніх елементів, у своїй лабораторній роботі я привів приклад інстанціювання трьох черг: «Черга цілих чисел», «Черга чисел з плаваючою точку» і «Черга рядків без пробілу(слів)».

Так як по завданню черга повинна бути визначена на двох стеках інстанційований черги будуть містити у собі інстанційовані стеки з таким же самим типом як і черга. Таким чином ми матимемо чергу на стеці і в залежності від перемикача буде обрано між стандартною і своєю реалізацією стека. Перемикач також буде перемикати три шаблонні функції які виконують певні дії с контейнером за допомогою ітераторів.

Нажаль у стандартному стеку немає ітераторів. Тому щоб обробити елементи черги на консультації з викладачем було вирішено реалізувати також допоміжний шаблонний клас «Список» у який будуть заноситися всі елементи черги, опрацьовуватися за допомогою ітераторів та повертатися назад до тієї самої черги.(для цього реалізовані дві допоміжні функції)

Всі реалізації своїх контейнерів та функції винесені у окремі хедер файли.

**Стандартні алгоритми та їх реалізація**

До обраних стандартних алгоритмів які можна робити з стандартними контейнерами ввійшли: sort, find, min\_element, і for\_each, який обробляє всі елементи контейнеру за допомогою предиката print. Предикат для функції for\_each було реалізовано самостійно. В залежності від режиму функції перемикаються зі стандартних на створені власноруч.

**Власна реалізація контейнерів**

До стандартних контейнерів ввійшли: стек, список, рядок. Реалізація кожного з них базована на відомостях про вміст кожного з контейнерів в бібліотеці стандартних шаблонів. Всі необхідні функції були реалізовані. Детальніше про реалізовані функції стандартних контейнерів можна прочитати у коментарях до коду.

# 3. Текст вихідної програми

code.cpp

#include <iostream>

#include <fstream>

#include <locale>

#include <conio.h>

#include <Windows.h>

#define \_flag\_ 0;

#ifdef \_flag\_

#include "Mylist.h"

#include "Mystack.h"

#include "Myalgorithm.h"

#include "Mystring.h"

#endif

#ifndef \_flag\_

#include <stack>

#include <string>

#include <algorithm>

#include <list>

using namespace std;

#endif

template<class T>

class Queue

{

stack <T> S1, S2;

public:

Queue(){}

bool empty() { return S1.empty() && S2.empty(); }

int size() { return S1.size()+S2.size(); }

T& front(){S1toS2(); return S2.top(); }

T& back(){S2toS1(); return S1.top(); }

void push(T& x) { S1.push(x); }

void pop() { S1toS2(); S2.pop(); }

private:

void S1toS2()

{

if (S2.empty())

while (!S1.empty()) {S2.push(S1.top()); S1.pop();}

}

void S2toS1()

{

if (S1.empty())

while (!S2.empty()) {S1.push(S2.top()); S2.pop();}

}

};

class print// пердикат вывода в поток

{

std::ostream &o;

public:

print(std::ostream &ok):o(ok){}// конструктор определяет поток

template<class t3> std::ostream& operator()(t3& s){o << s << std::endl; return o;}// перегруженый скобки производят вывод

};

template <class t1>

void QtoL(Queue<t1> &Q, list<t1> &L)// функция перевода очереди в список

{

while (!Q.empty())

{L.push\_back(Q.front());

Q.pop();}

}

template <class t2>

void LtoQ(Queue<t2> &Q, list<t2> &L)// функция перевода списка в очередь

{

list<t2>::iterator b = L.begin();

list<t2>::iterator e = L.end();

for (; b!=e; b++) {Q.push(\*b);}

}

template <typename S>

void menu(Queue<S> &Q)

{

while(true)

{

system("cls");

std::cin.clear();

\_flushall();

std::cout<<"Главное меню очереди"<<std::endl;

std::cout<<"1-push (добаивть елемент в очередь)"<<std::endl;

std::cout<<"2-pop(удалить елемент из очереди)"<<std::endl;

std::cout<<"3-front (доступ к первому елементу)"<<std::endl;

std::cout<<"4-back(доступ к последнему елементу)"<<std::endl;

std::cout<<"5-size(размер очереди)"<<std::endl;

std::cout<<"6-sort (отсортировать очередь с помощью списком)"<<std::endl;

std::cout<<"7-find (найти в очереди первое вхождение елемента c помощью списка)"<<std::endl;

std::cout<<"8-min\_element(найти минимальный елемент очереди с помощью списка)"<<std::endl;

std::cout<<"9-print (вывести очередь с помощью списка и функции for\_each)"<<std::endl;

std::cout<<"0-выйти из главного меню очереди"<<std::endl;

std::cout<<"---------------------------------------------------------------"<<std::endl;

std::cout<<std::endl;

switch(getch())

{

case '1':{S q;std::cout<<"Введите елемент который будет внесен в очередь"<<std::endl; std::cin>>q; Q.push(q); system("pause"); system("cls"); continue;}

case '2':{if(Q.empty()){std::cout<<"Очередь пуста"<<std::endl; system("pause"); system("cls"); continue;}std::cout<<"Был удален елемент очереди :"<<Q.front()<<std::endl;Q.pop(); system("pause"); system("cls"); continue;}

case '3':{if(Q.empty()){std::cout<<"Очередь пуста"<<std::endl; system("pause"); system("cls"); continue;}std::cout<<"Первый елемент на очереди :"<<Q.front()<<std::endl; system("pause"); system("cls"); continue;}

case '4':{if(Q.empty()){std::cout<<"Очередь пуста"<<std::endl; system("pause"); system("cls"); continue;}std::cout<<"Последний елемент на очереди :"<<Q.back()<<std::endl; system("pause"); system("cls"); continue;}

case '5':{std::cout<<"В очереди "<<Q.size()<<" елементов"<<std::endl; system("pause"); system("cls"); continue;}

case '6':{if(Q.empty()){std::cout<<"Очередь пуста"<<std::endl; system("pause"); system("cls"); continue;}list<S> L; QtoL(Q,L); L.sort(); LtoQ(Q,L); std::cout<<"Очередь была отсортирована"<<std::endl; system("pause"); system("cls"); continue;}

case '7':{if(Q.empty()){std::cout<<"Очередь пуста"<<std::endl; system("pause"); system("cls"); continue;}list<S> L; list<S>::iterator f; S val; QtoL(Q,L);

std::cout<<"Введите елемент который хотите найти"<<std::endl;

std::cin>>val;

f=find(L.begin(),L.end(),val);

if(L.end()==f) std::cout<<"Елемент "<<val<<" не найден"<<std::endl;

else std::cout<<"Елемент "<<val<<" найден"<<std::endl;

LtoQ(Q,L);

system("pause"); system("cls"); continue;}

case '8':{if(Q.empty()){std::cout<<"Очередь пуста"<<std::endl; system("pause"); system("cls"); continue;}list<S> L; list<S>::iterator f; QtoL(Q,L);

f=min\_element(L.begin(),L.end());

std::cout<<"Наименьший елемент очереди "<<\*f<<std::endl;

LtoQ(Q,L);

system("pause"); system("cls"); continue;}

case '9':{if(Q.empty()){std::cout<<"Очередь пуста"<<std::endl; system("pause"); system("cls"); continue;}list<S> L; list<S>::iterator f; QtoL(Q,L);

std::cout<<"Куда вы хотите вывести содержимое очереди ?"<<std::endl;

std::cout<<"1-Вывод в консоль"<<std::endl;

std::cout<<"2-Вывод в файл"<<std::endl;

switch(getch())

{

case '1':{for\_each(L.begin(),L.end(), print(std::cout)); LtoQ(Q,L); system("pause"); system("cls"); continue;}

case '2':{

std::ofstream fout;

char buff[26];

std::cout<<"Введите имя файла в который будете выводить"<<std::endl;

std::cin>>buff;

fout.open(buff);

for\_each(L.begin(),L.end(),print(fout));

LtoQ(Q,L); system("pause"); system("cls"); continue;}

}

LtoQ(Q,L);

system("pause"); system("cls"); continue;}

case '0':{return;}

}

}

}

int main()

{

Queue <int> qi;

Queue <float> qf;

Queue <string> qs;

setlocale(LC\_ALL, "russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::cout<<"Добро пожаловть в Лабораторную работу №4"<<std::endl;

std::cout<<"Задание: Опеределить очередь через два (и не больше) стэка"<<std::endl;

bool i=\_flag\_;

if(i) std::cout<<"Программа скомпилирована без SLT"<<std::endl;

else std::cout<<"Программа скомпилирована c SLT"<<std::endl;

while(1)

{

std::cin.clear();

\_flushall();

std::cout<<"С какой очередью вы хотите работать?"<<std::endl;

std::cout<<"1- С очередью int"<<std::endl;

std::cout<<"2- С очередью float"<<std::endl;

std::cout<<"3- С очередью string (слова без пробелов)"<<std::endl;

std::cout<<"0- Выйти из программы"<<std::endl;

switch(getch())

{

case '1':{menu(qi); system("cls"); continue;}

case '2':{menu(qf); system("cls"); continue;}

case '3':{menu(qs); system("cls"); continue;}

case '0':{return 0;}

}

}

}

Myalgorithmt.h

#pragma once

template<class t1, class itor>

itor find(itor b, itor e, t1 what)// функция нахождения елемента в контейнере

{

for(;b!=e; b++) if((\*b)==what) return b;//обходим весь контейнер пока не находим искомый елемент и возвращем его итератор

return e;

}

template<class itor>

itor min\_element(itor b, itor e)//функция нахождения минимального елемента в контейнере

{

itor temp=b;

for(;b!=e; b++) if((\*b)<(\*temp)) temp=b;// обхлжим контейнер сравнивая минимальний елемент с елементами контейнера

return temp;//возвращяем минимальний

}

template<class iter, class Predicate>

void for\_each(iter b, iter e, Predicate pred)// функция "для каждого елемента"

{

iter i=b;

for(i; i!=e; i++)//для каждого елемента контейнера выполняем действией

pred(\*i);

}

Mystack.h

#pragma once

#include "Mylist.h"

template <class T>

class stack// стак основаный на списке

{

list<T> l;

public:

stack(){}

void push(T& p){l.push\_back(p);}//добавить в конец списка

void pop(){l.pop\_back();}//достать с конца списка

bool empty(){return l.empty();}// пустой ли список?

int size(){return l.size();}//размер списка

T& top(){return l.back();}//доступ к последнему елементу

};

Mystring.h

#pragma once

#include <iostream>

#define MAX\_SIZE 1024

#define NULL 0

int strlen(char \*word)// функция длинны строки

{

int count = 0;

while (\*word++) ++count;

return count;

}

int strcmp(char \*a, char \*b)//функция сравнение строк

{

int c = 0;

while (a[c] != '\0'|| b[c] != '\0')

{

if (a[c] > b[c]) return 1;

if (a[c] < b[c]) return -1;

c++;

}

return 0;

}

void strcpy(char\* Dest, char\* Sourse)//функция копирования строки в строку(без дописи)

{

int i;

for (i = 0; i < strlen(Sourse); i++)

Dest[i] = Sourse[i];

Dest[i] = '\0';

}

char\* strstr(char \*a, char \*b)//нахождение подстроки

{

for (int i = 0; a[i] != '\0'; i++)

if (a[i] == b[0])

return &a[i];

return NULL;

}

class string

{

char \*str;

public:

string():str(NULL){}

string(char\* strn){str=new char[strlen(strn)+1]; strcpy(str,strn);}

string(const string &s){str=new char[strlen(s.str)+1]; strcpy(str,s.str);}

bool operator>=(string &s)

{

for(int i=0; str[i]!='\0'||s.str[i]!='\0'; i++)

{

if (str[i]>s.str[i]) return 1;

else if (str[i]<s.str[i]) return 0;

}

if(s.size()==this->size()) return 1;

return 0;

}

bool operator<=(string &s)

{

for(int i=0; str[i]!='\0'||s.str[i]!='\0'; i++)

{

if (str[i]<s.str[i]) return 1;

else if (str[i]>s.str[i]) return 0;

}

if(s.size()==this->size()) return 1;

return 0;

}

string& operator=(string &s)

{

if (!str){str=new char[strlen(s.str)+1]; strcpy(str,s.str);}

else{str=new char[strlen(s.str)+1]; strcpy(str,s.str);}

if (strcmp(str,s.str)){delete[]str;}

return \*this;

}

bool operator>(string &s)

{

for(int i=0; str[i]!='\0'||s.str[i]!='\0'; i++)

{

if (str[i]>s.str[i]) return 1;

else if (str[i]<s.str[i]) return 0;

}

return 0;

}

bool operator<(string &s)

{

for(int i=0; str[i]!='\0'||s.str[i]!='\0'; i++)

{

if(str[i]<s.str[i]) return 1;

else if(str[i]>s.str[i]) return 0;

}

return 0;

}

bool operator==(string &s){if (!strcmp(str,s.str))return true; else return false;}

int size(){return strlen(str);}

friend std::ostream& operator<<(std::ostream &o,string &s)

{

return o<<s.str;

}

friend std::istream& operator>>(std::istream &i,string &s)

{

char tmpString[MAX\_SIZE];

i >> tmpString;

if (s.str!=NULL) delete[]s.str;

s.str = new char[strlen(tmpString)+1];

strcpy(s.str,tmpString);

return i;

}

~string(){delete[]str;}

};

Mylist.h

#pragma once

template <class T>

class list

{

struct node // структура узла списка

{

T info; // поле с данными

node\* next; // указатель на предыдущий узел

node\* prev; // указатель на следующий узел

node(T& in){info=in;next=NULL;prev=NULL;}

node(){next=NULL; prev=NULL;}

};

node\* head; //указатель на голову списка

public:

class iterator// итератор

{

public:

node \*ptr;// поле итератора(указатель на структуру "узел")

iterator():ptr(NULL){}//конструктор по умолчанию

iterator(node \*pl):ptr(pl){}// конструктор с параметрами (переводит указатель на узел в объект класса итератор

T& operator\*(){return ptr->info;}//операция разименования итератора (возвращает данные по итератору)

void operator++(){ptr = ptr->next;}//инкримента итератора (переходит на следующий елемент в списке)

iterator operator+(int x) {iterator tmp(ptr); for (int i=0; i<x; i++) tmp++; return tmp;}

bool operator!=(iterator i){return ptr != i.ptr;}// операция "не равно" (сравнивает поле данных двух итераторов)

bool operator==(iterator i){return ptr == i.ptr;}

};

iterator begin(){return iterator(head);}// вернуть итератор первого елемента елемент списка

iterator end(){return iterator(NULL);}// вернуть итератор конечного елемента списка (тот который находиться за последний узлом)

list():head(NULL){}// конструктор по умолчанию

void push\_back(T &n)// функция "добавить в конец"

{

node \*elem = new node(n);//создаем новый узел с данными

if(!head){head=elem;}// если головы еще нет, ставим указательголовы на новый елемент

else// в противном случае

{ node\* temp=head;// временным указателем указывем на голову

for(temp; temp->next!=NULL; temp=temp->next);//двигаемся пока не находим предпоследний узел

temp->next=elem;//указатель на следующий елемент пердпоследнего узла ставим на новый созданный

elem->prev=temp;//указатель нового на предыдущий ставим на предпоследний узел

}

}

T& front() {return head->info;}//доступ к первому елементу

T& back() { node\* temp=head; while(temp->next!=NULL) temp=temp->next; return temp->info;}//доступ к последнему елементу

void swap(T &a, T &b)// поменять местами два елемента

{

T temp;

temp = a;

a = b;

b = temp;

}

void sort() //сортировать список

{

for (iterator i(begin());i!=end();i++)

for (iterator j(i + 1);j!=end();j++)

if (\*i>\*j) swap(\*i,\*j);

}

bool empty(){ return (!head)? 1 : 0;}// список пуст?

int size()//размер списка

{

if(!head) return 0;

node \*temp=head;

int c=0;

while(temp!=NULL){temp=temp->next; c++;}

return c;

}

void pop\_back()//убрать елемент с конца

{

if(head==NULL) return;

if(head->next==NULL) {node\* temp=head; head=NULL; delete temp; return;}

node\* temp=head;

while(temp->next!=NULL) temp=temp->next;

temp->prev->next=NULL;

delete temp;

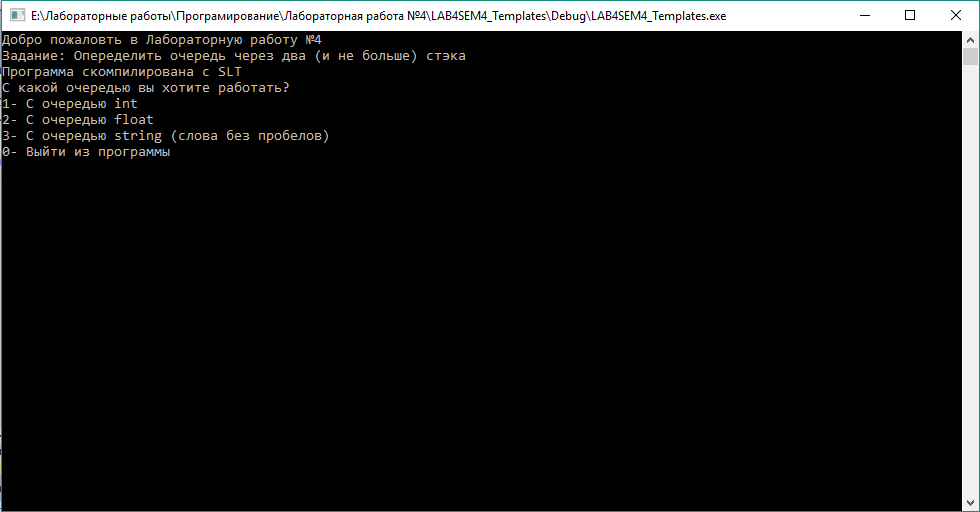
}

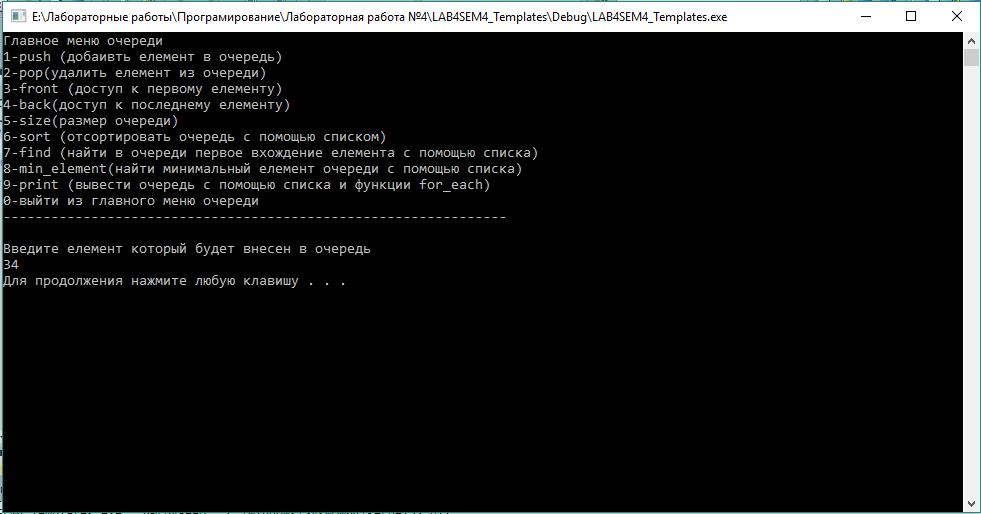
};

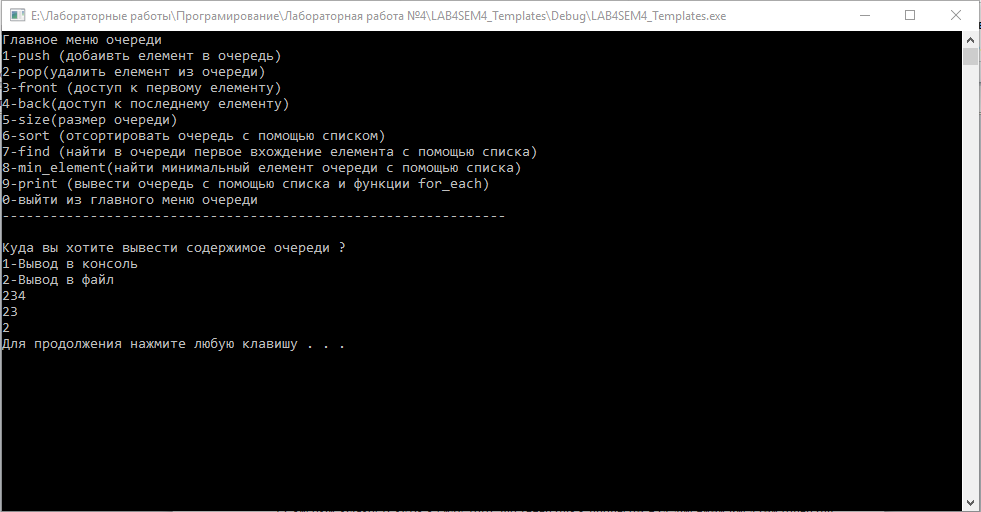
# 4. Опис інтерфейсу

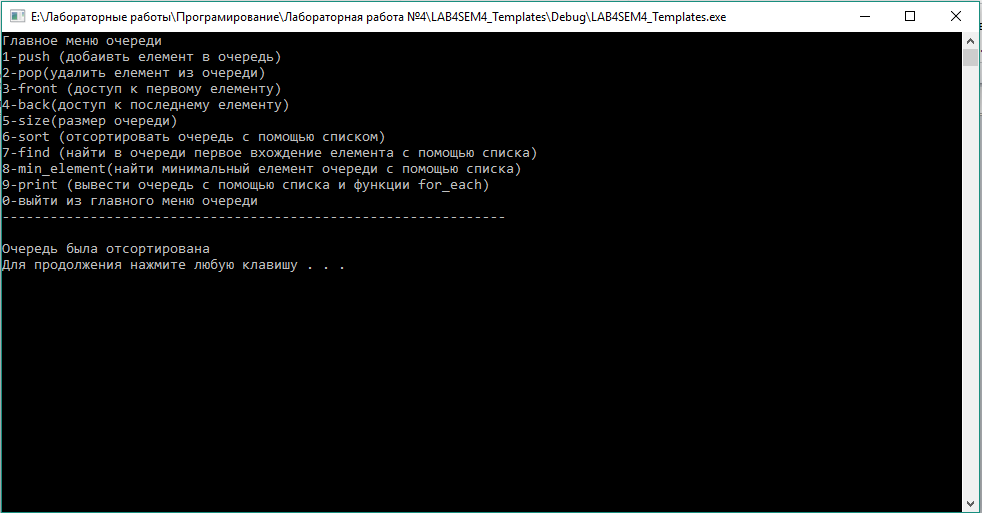
Програма дотримується всіх принципів дружнього інтерфейсу. Користувач працює з чергою через меню. Для розвантаження екрану використовується системне очищення консолі. Після виконання операцій у меню чергою користувач повертається до меню з вибору вже інстанційованих черг і працює в ньому доти доки не забажає вийти.

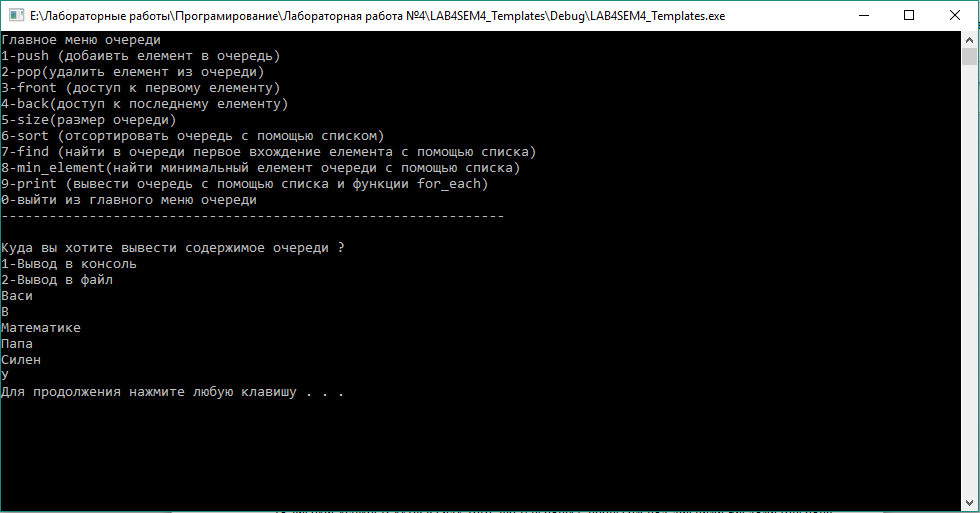
# 5. Описання тестових прикладів











# 6.Аналіз помилок

Зауваження 1 Як вже було помічено неможливе використання ітераторів ні черги ні класу бо їх не існує тому було створено штучний приклад с опрацюванням ітераторів.

Проблема 1 Неможна винести функції шаблонного класу як модульні (у окремий сpp файл) бо для шаблонних функцій програма повинна мати прямий доступ.

Проблема 2 Сортування списку можливе тільки функцією sort списку. Стандартне сортування не працю бо у стандартному класі списку не визначена операція -