ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №4

за курсом «Об’єктно-орієнтоване програмування»

студента групи ПА-18-1

кафедра комп’ютерних технологій, ДНУ 2019/2020

Варіант №8

**Постановка задачі**

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

В індивідуальноу варіанті завдання повинні бути реалізовані осно- вні можливості бібліотеки стандартних шаблонів.

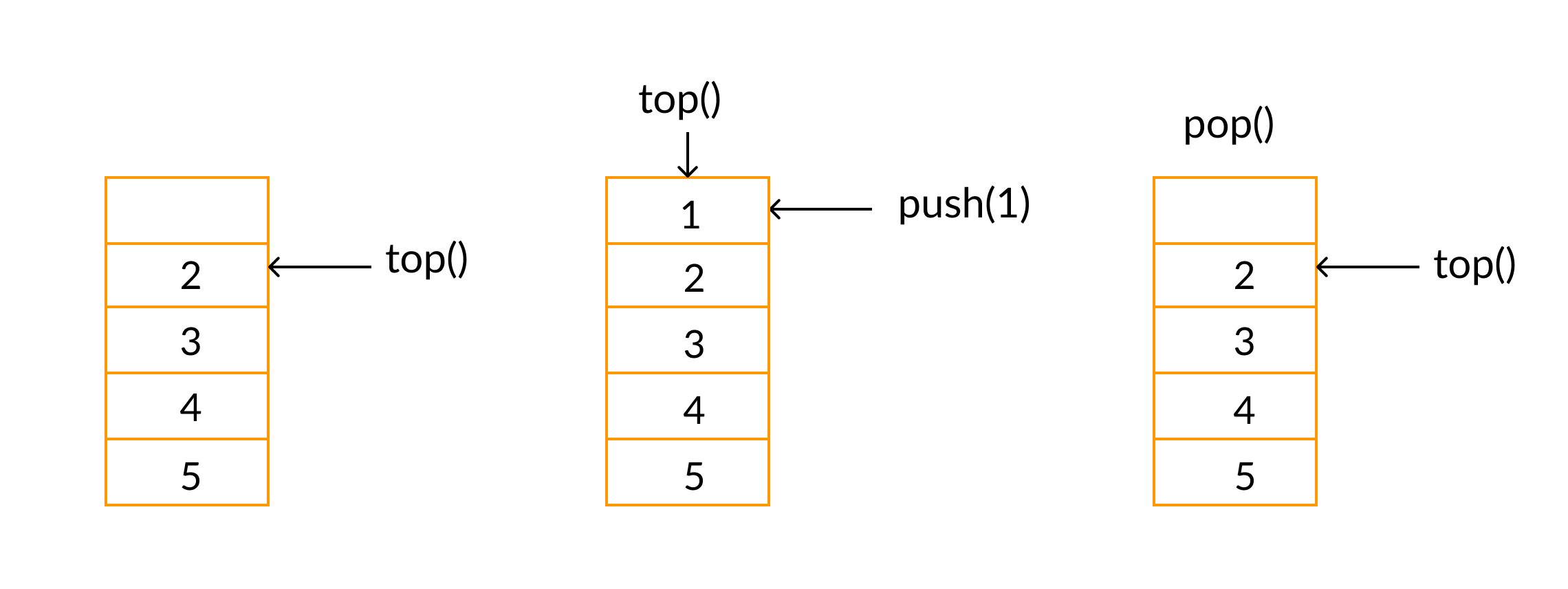
При написанні програми слід враховувати необхідність забезпечення її зборки у двох режимах: 1. Створення виконуваного файлу із підключенням/використанням типів і алгоритмів бібліотеки стандартних шаблонів (STL); 2. Створення виконуваного файлу без застосування STL. Необхідні зміни вихідного тексту для переналаштування з/на STL бажано робити за допомогою директив умовної компіляції.

Індивідуальне завдання: визначити чергу, використовуючи (тільки) два стека.

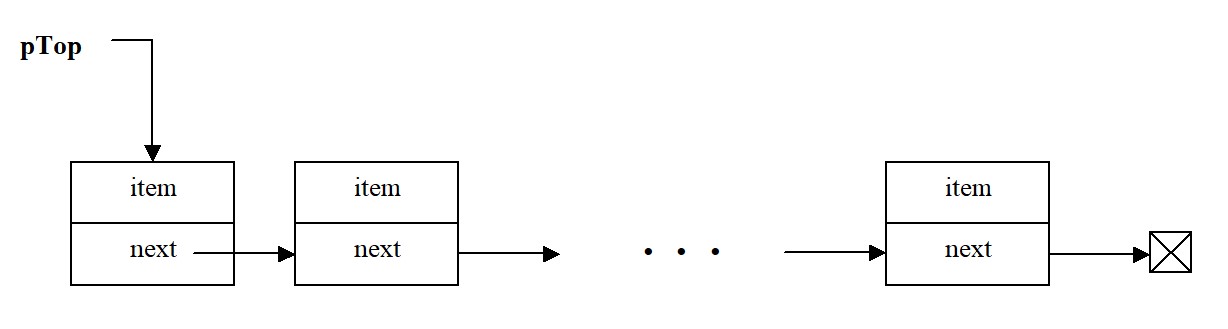
**Опис розв’язку**

Стек – це структура даних, яка працює за принципом LIFO (last in – first out – “останнім прийшов – першим вийшов”). Операції можна проводити лише з одним, верхнім елементом, тобто тим, який останнім додали до стека. Отже, для цієї структури не важливо, з якою швидкістю ми будемо перебирати її елементи (тому що ми і не зможемо їх перебирати в звичайному розумінні, лише видаляти та отримувати нову верхівку (top), як показано на малюнку нижче), але критично важливими є операції вставки та видалення (причому лише з одної сторони). Тому в основу стека при його реалізації без використання бібліотеки стандартних шаблонів я вирішив покласти однозв'язний список. Ця структура даних має декілька важливих переваг порівняно з динамічним масивом. Для нашої задачі, зокрема, важливо те, що в ній набагато ефективніше (за час О(1), тобто незалежно від кількості елементів) виконуються процедури додавання та вилучення елементів. Масиви в свою чергу набагато кращі в операціях, потребуючих безпосереднього доступу до кожного елементу, але це, як вже зазначалось вище, не є необхідним для стека, в якому навіть немає ітератора.

В списках також не існує проблеми «розширення», яка рано чи пізно виникає в масивах фіксованого розміру, коли виникає необхідність включити в нього додаткові елементи. Точно так, фіксований масив, з якого було видалено багато елементів (або вони просто не використовуються) є дуже неефективним з точки зору використання пам'яті. Функціонування списків можливо в ситуації, коли пам'ять комп'ютера фрагментована, тоді як масиви переважно потребують неперервної області для зберігання.

****

***Вставка та видалення елемента стека***

Кожний вузол однозв’язного списку містить два поля: елемент даних data деякого узагальненого типу T та вказівник next на наступний елемент типу Node. Верхній елемент стеку – вказівник head. В останньому вузлі значення next = nullptr.

***Стек у вигляді однозв’язного списку***

Наш клас однозв’язного списку перш за все має підтримувати додавання вузла на його початок, видалення останнього доданого вузла та отримання значення верхнього елемента. Окрім того, реалізуємо можливість перебирання елементів списка за допомогою ітераторів.

Додавання вузла в список виконується в самий початок. Кожен новий вузол стає головним елементом списку:

template<class T>

void list<T>::push\_front(T data)

{

try { head = new Node(data, head); }

catch (bad\_alloc e) { cout << e.what() << endl; return; }

N++;

}

При видаленні вузла вилучається його головний елемент. Якщо в списку ще є вузли, то новим головним елементом стає наступний за попереднім головним вузол:

template<class T>

void list<T>::pop\_front()

{

if (head == nullptr)

{

cout << ":(\n";

}

else {

Node\* tmp = head;

head = head->next;

tmp->next = NULL;

delete tmp;

N--;

}

}

Список – це динамічна структура. При додаванні нових елементів виділяється пам’ять, яку необхідно звільняти, щоб не виникав витік пам’яті:

~list()

{

while (N)

pop\_front();

}

У якості ітераторів для початку та кінця списку повернемо наступне:

iterator begin() { return iterator(head); }

iterator end() { return iterator(NULL); }

Сам клас ітератор з деякими перевантаженими для коректної роботи алгоритмів операціями має вигляд:

class iterator {

public:

typedef typename Node\* pointer;

iterator(pointer p) : p(p) {}

void operator++(int) { return iterator(p = p->next); }

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

value\_type& operator\*() { return p->data;; }

private:

pointer p;

};

Клас стек я визначив не як окремий контейнер, а як адаптер контейнеру list, фрагменти реалізації якого наведені вище. У якості адаптера стек надає обмежений інтерфейс до контейнера. Зокрема, у ньому невизначений клас ітератор, бо його наявність може порушити головний принцип роботи структури (наприклад, якщо ми через ітератори застосуємо який-небудь алгоритм сортування або зможемо міняти елементи місцями). Реалізація адаптера досить компактна, тому наведу її повністю:

#pragma once

#include "MyList.h" // файл заголовка з власною реалізацією класа list

template <class T, class C = list<T>> class stack {

C c;

public:

typedef typename C::value\_type value\_type;

typedef typename C::size\_type size\_type;

explicit stack(const C& a = C()) : c(a) {}

value\_type& top() { return \*c.begin(); }

size\_type size() { return c.size(); }

bool empty() { return c.empty(); }

void push(const value\_type& x) { c.push\_front(x); }

void pop() {

if (c.empty())

cout << "Stack Underflow\n";

else

c.pop\_front();

}

};

Отже, stack – це лише інтерфейс для контейнера, тип якого передається як аргумент шаблона. Все, що робить stack – це прибирає з інтерфейса нестекові операції для свого контейнера і дає операціям begin(), pop\_front(), push\_front() загальноприйняті імена top(), pop(), push(). Зауважимо, що на відміну від черги, в стеку відсутня можливість переглянути нижній елемент, тобто з ітератором end() класу list ми не взаємодіємо, хоча для подальшої реалізації черги через два стеки це могло бути корисним.

Клас queue буде зберігати в собі два екземпляри класу stack: s1 і s2. Операція push() буде виконуватись наступним чином: доти, доки s1 не порожній, перекладаємо його елементи до s2 послідовним викликом операцій push() та pop() (додаємо верхній елемент s1 до s2, після чого видаляємо його). Тепер в s2 елементи лежать в зворотному (для черги) порядку, додамо новий елемент на верхівку і знову перекладемо елементи, але вже в s1. Таким чином верхній елемент s1 завжди буде відповідати верхньому елементу черги, тому операці pop() та top() будемо проводити лише з ним.

**Вихідний текст програми розв’язку задачі**

Файл MyList.h

#include<iostream>

using namespace std;

template<class T>

class list

{

public:

typedef typename T value\_type;

typedef typename int size\_type;

private:

class Node

{

public:

Node\* next;

value\_type data;

Node(T data = T(), Node\* next = nullptr) : data(data), next(next) {}

};

Node\* head;

size\_type N;

public:

class iterator {

public:

typedef typename Node\* pointer;

iterator(pointer p) : p(p) {}

void operator++(int) { return iterator(p = p->next); }

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

value\_type& operator\*() { return p->data;; }

private:

pointer p;

};

iterator begin() { return iterator(head); }

iterator end() { return iterator(NULL); }

list() : head(NULL), N(0) {

/\*cout << "Вызвался конструктор\t\t\t" << this << "\n"; \*/

}

list(Node\* head, size\_type size) : head(head), N(size) {}

list(const list& other): head(NULL)

{

/\*cout << "Вызвался конструктор копирования \t" << this << "\n";\*/

Node\* tmp = other.head;

while (tmp != nullptr)

{

this->push\_back(tmp->data);

tmp = tmp->next;

}

}

~list()

{

/\*cout << "Вызвался деструктор\t\t\t" << this << "\n";\*/

while (N)

{

pop\_front();

}

}

void pop\_back();

void pop\_front();

void push\_back(T data);

void push\_front(T data);

void insert(T data, int index);

void RemoveAt(int index);

void clear();

void display();

value\_type& operator[](int index);

list& operator= (const list& other)

{

this->head = nullptr;

Node\* tmp = other.head;

while (tmp != nullptr)

{

this->push\_back(tmp->data);

tmp = tmp->next;

}

return \*this;

}

size\_type size() { return N; }

bool empty()

{

if (head == nullptr && N==0)

return true;

else

return false;

}

};

template <class T>

struct iter\_traits {

typedef typename T::value\_type value\_type;

};

template <class T>

struct iter\_traits <T\*> {

typedef typename T value\_type;

};

template<class T>

void list<T>::pop\_back()

{

RemoveAt(N - 1);

}

template<class T>

void list<T>::pop\_front()

{

if (head == nullptr)

{

cout << ":(\n";

}

else {

Node\* tmp = head;

head = head->next;

tmp->next = NULL;

delete tmp;

N--;

}

}

template<class T>

void list<T>::push\_back(T data)

{

if (head == nullptr)

{

head = new Node(data);

}

else

{

Node\* current = this->head;

while (current->next != nullptr)

current = current->next;

try { current->next = new Node(data); }

catch (bad\_alloc e) { cout << e.what() << endl; return; }

}

N++;

}

template<class T>

void list<T>::push\_front(T data)

{

try { head = new Node(data, head); }

catch (bad\_alloc e) { cout << e.what() << endl; return; }

N++;

}

template<class T>

void list<T>::insert(T data, int index)

{

if (index == 0)

push\_front(data);

else

{

Node\* previous = this->head;

for (int i = 0; i < index - 1; i++)

previous = previous->next;

Node\* newnode = new Node(data, previous->next);

N++;

}

}

template<class T>

void list<T>::RemoveAt(int index)

{

if (index == 0)

pop\_front();

else

{

Node\* previous = this->head;

for (int i = 0; i < index - 1; i++)

previous = previous->next;

Node\* todelete = previous->next;

previous->next = todelete->next;

delete todelete;

N--;

}

}

template<class T>

void list<T>::clear()

{

while (N)

pop\_front();

}

template<class T>

void list<T>::display()

{

if (head == nullptr)

cout << "stack underflow\n";

else

{

for (int i = 0; i < N; i++)

cout << this->operator[](i) << endl;

}

}

template<class T>

T& list<T>::operator[](int index)

{

int counter = 0;

Node\* current = this->head;

while (current != nullptr)

{

if (counter == index)

return current->data;

current = current->next;

counter++;

}

}

Файл MyStack.h

#pragma once

//#define STL

#ifndef STL

#include "MyList.h"

#else

#include <iostream>

using namespace std;

#include<list>

#endif

template <class T, class C = list<T>> class stack {

C c;

public:

typedef typename C::value\_type value\_type;

typedef typename C::size\_type size\_type;

explicit stack(const C& a = C()) : c(a) {}

value\_type& top() { return \*c.begin(); }

size\_type size() { return c.size(); }

bool empty() { return c.empty(); }

void push(const value\_type& x) { c.push\_front(x); }

void pop() {

if (c.empty())

cout << "Stack Underflow\n";

else

c.pop\_front();

}

stack& operator= (const stack& other) {

this->c = other.c;

return \*this;

}

};

Файл MyQueue.h

#pragma once

//#define STL

#ifndef STL

#include "MyStack.h"

#else

#include <iostream>

using namespace std;

#include<stack>

#endif

template <class T, class S = stack<T>> class queue {

S s1;

S s2;

public:

typedef typename S::value\_type value\_type;

typedef typename S::size\_type size\_type;

bool empty() {

return s1.empty();

}

size\_type size () {

if (!s1.empty())

return s1.size();

else

return 0;

}

void pop()

{

s1.pop();

}

void push(const value\_type& x)

{

while (!s1.empty())

{

s2.push(s1.top());

s1.pop();

}

s2.push(x);

while (!s2.empty())

{

s1.push(s2.top());

s2.pop();

}

}

value\_type& front()

{

return s1.top();

}

value\_type& back()

{

value\_type tmp;

s2 = s1;

while (!s2.empty())

{

if (s2.size() == 1)

tmp = s2.top();

s2.pop();

}

return tmp;

}

queue& operator= (const queue& other) {

this->s1 = other.s1;

this->s2 = other.s2;

return \*this;

}

};

Файл Main.cpp

#include <iostream>

#include <fstream>

using namespace std;

//#define STL

#ifndef STL

#include "MyQueue.h"

#else

#include <queue>

#endif

void menu()

{

cout << "\t\tМеню" << endl;

cout << "1. Добавить элемент в очередь" << endl;

cout << "2. Добавить несколько элементов в очередь" << endl;

cout << "3. Удалить элемент из очереди" << endl;

cout << "4. Показать верхний элемент" << endl;

cout << "5. Показать нижний элемент" << endl;

cout << "6. Показать все элементы очереди" << endl;

cout << "7. Удалить все элементы из очереди" << endl;

cout << "0. Выход" << endl;

}

int main()

{

setlocale(LC\_ALL, "RUS");

typedef typename queue<int>::value\_type value\_type;

int arr[]{ 1, 2, 3, 4, 5 };

const int n = sizeof(arr) / sizeof(int);

queue<value\_type> q;

queue<value\_type> q2;

ifstream fin;

ofstream fout;

int ch, count;

value\_type ne;

#ifndef STL

cout << "Программа работает без подключения контейнеров библиотеки стандартных шаблонов" << endl;

#else

cout << "Программа работает с подключением контейнеров библиотеки стандартных шаблонов" << endl;

#endif // STL

menu();

do

{

cout << "\nВведите номер задания: ";

cin >> ch;

switch (ch)

{

case 1:

cout << "Введите новый элемент: ";

cin >> ne;

q.push(ne);

break;

case 2:

cout << "Для добавления данных с файла введите 1, для добавления с консоли - 2\n";

cin >> ch;

if (ch == 1)

{

string tname;

cout << "Введите название текстового файла: ";

cin >> tname;

fin.open(tname);

if (fin.is\_open())

{

fin >> count;

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

{

fin >> ne;

q.push(ne);

}

}

else

cout << "Ошибка открытия файла" << endl;

fin.close();

}

else if (ch==2) {

cout << "Введите количество элементов: ";

cin >> count;

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

{

cin >> ne;

q.push(ne);

}

}

break;

case 3:

if (!q.empty())

{

ne = q.front();

q.pop();

cout << "Элемент " << ne << " удален из очереди" << endl;

}

else

cout << "Очередь пуста" << endl;

break;

case 4:

if (!q.empty())

{

cout << "Верхний элемент: " << q.front() << endl;

}

else

cout << "Очередь пуста" << endl;

break;

case 5:

if (!q.empty())

{

cout << "Нижний элемент: " << q.back() << endl;

}

else

cout << "Очередь пуста" << endl;

break;

case 6:

if (q.empty())

cout << "Очередь пуста" << endl;

else

{

cout << "Для вывода элементов в файл введите 1, для вывода на консоль - 2\n";

cin >> ch;

q2 = q;

if (ch == 2)

{

cout << endl;

while (!q2.empty())

{

cout << q2.front() << endl;

q2.pop();

}

}

else if (ch == 1)

{

string tname;

cout << "Введите название файла: ";

cin >> tname;

fout.open(tname);

if (fout.is\_open())

{

while (!q2.empty())

{

fout << q2.front() << endl;

q2.pop();

}

}

else

cout << "Ошибка открытия файла" << endl;

fout.close();

}

else

cout << "Ошибка! Повторите попытку ещё раз" << endl;

}

break;

case 7:

if (q.empty())

cout << "Очередь пуста" << endl;

else

{

while (!q.empty())

{

cout << "Элемент " << q.front() << " удален из очереди" << endl;

q.pop();

}

}

break;

case 0:

cout << "Завершение работы программы" << endl;

break;

default:

cout << "Ошибка! Повторите попытку ещё раз" << endl;

break;

}

} while (ch != 0);

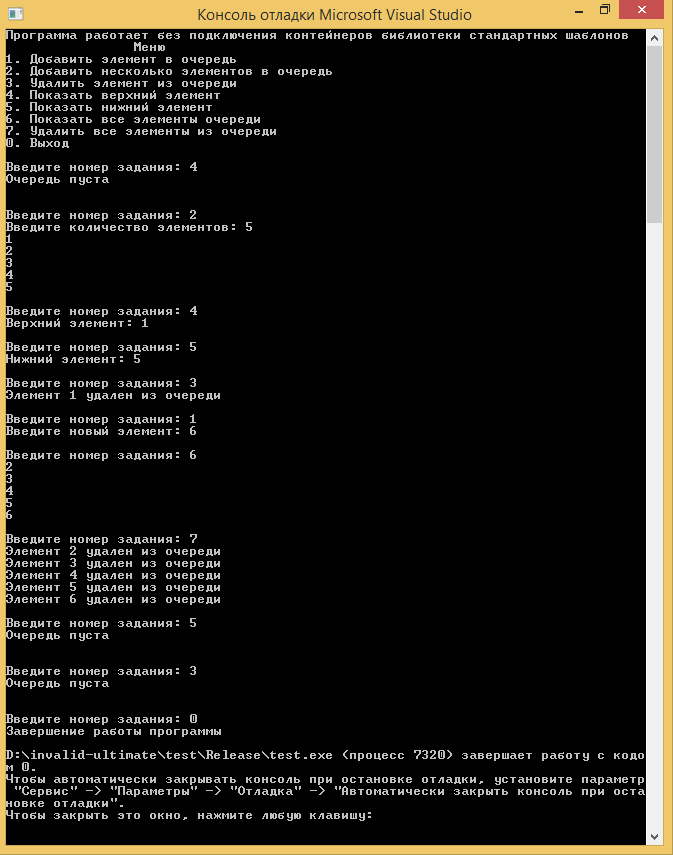
return 0;

}

**Опис інтерфейсу (керівництво користувача)**

Для запуску програми необхiдно запустити її виконавчий файл. Пiсля цього на екран виводиться ряд завдань або запитань, вiдповiдi на якi вводяться з клавiатури. Для завершення роботи програми достатньо натиснути будь-яку клавішу клавіатури.

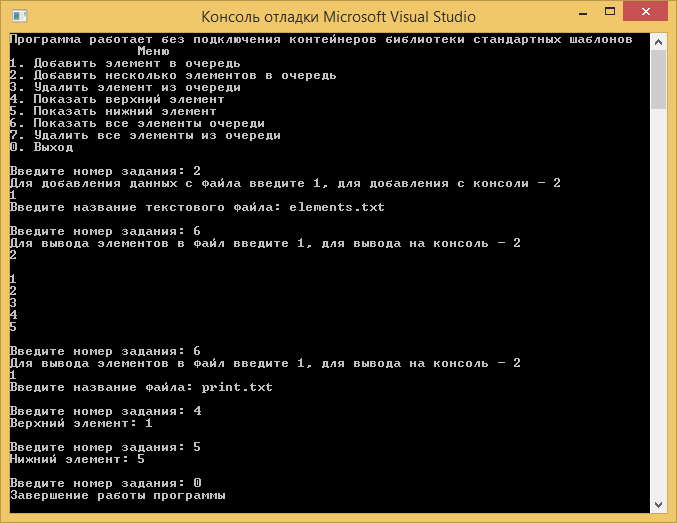
**Опис тестових прикладів**



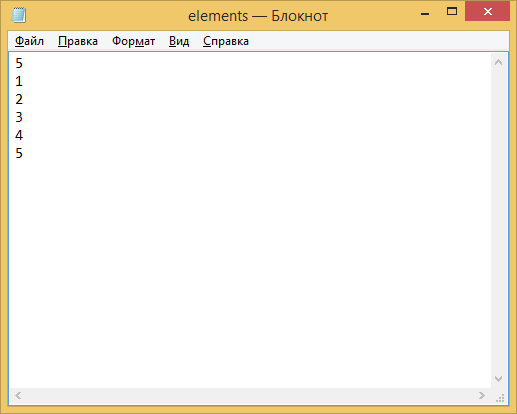
***приклад роботи програми з даними типу int без використання стандартних контейнерів***



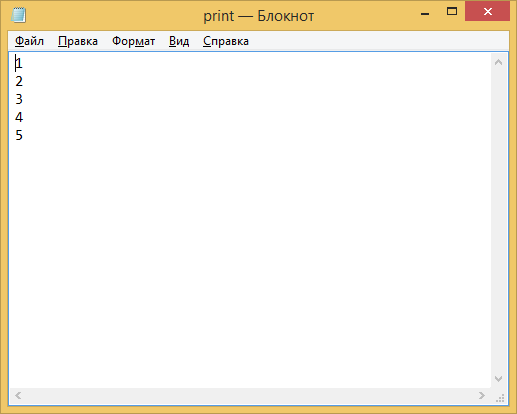
***приклад роботи програми з даними типу string***



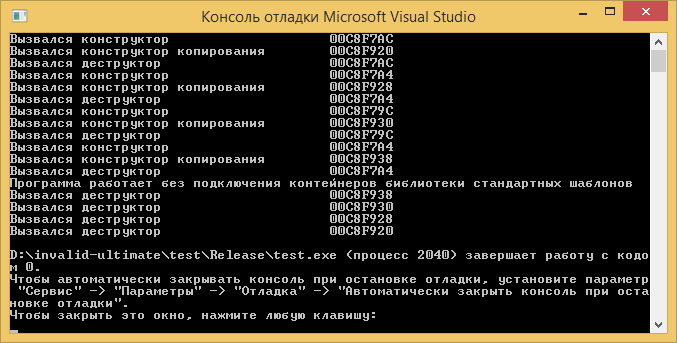
***Приклад роботи з файлами***



***Зміст файлу elements***



***Зміст файлу print***



***Перевірка коректності роботи з динамічною пам’яттю***



***Приклад роботи програми с підключенням бібліотеки стандартних шаблонів***

**Аналіз помилок (опис усунення зауважень)**

Цікавих помило при виконанні лабораторного завдання не виникало.