МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

**КАФЕДРА ИИТ**

ОТЧЁТ

по лабораторной работе №3

**«Стэк и очередь»**

Выполнил:

Студент 1 курса

группы ПО-9

Харитонович Захар Сергеевич

Проверила:

Войцехович О. Ю.

Брест 2022

**Цель работы:** Изучение принципов организации и работы с абстрактной структурой данных СТЭК в форме односвязного линейного списка и с абстрактной структурой данных ОЧЕРЕДЬ в форме односвязного линейного списка.

**Порядок выполнения работы**

*Задание 1*

Разработать консольное приложение, которое с помощью абстрактной структуры данных СТЭК проверяет соответствие открывающих и закрывающих HTML-тэгов во фрагменте HTML кода, введённого с клавиатуры

**Реализация**

*stack.h*

#ifndef TASK1\_STACK\_H

#define TASK1\_STACK\_H

#include <string>

struct stacknode {

std::string value;

struct stacknode \*next;

};

struct stack {

struct stacknode \*top;

int size;

};

struct stack \*stack\_init();

void stack\_push(struct stack \*s, std::string value);

bool stack\_is\_empty(struct stack \*s);

std::string stack\_top(struct stack \*s);

void stack\_pop(struct stack \*s);

#endif

*stack.cpp*

#include "stack.h"

struct stack \*stack\_init() {

struct stack \*temp = new struct stack;

temp->size = 0;

temp->top = nullptr;

return temp;

}

void stack\_push(struct stack \*s, std::string value) {

struct stacknode \*node = new struct stacknode;

node->next = s->top;

node->value = value;

s->top = node;

s->size++;

}

bool stack\_is\_empty(struct stack \*s) {

if (s->size == 0) return 1;

return 0;

}

std::string stack\_top(struct stack \*s) {

return s->top->value;

}

void stack\_pop(struct stack \*s) {

if (stack\_is\_empty(s)) return;

struct stacknode \*temp;

temp = s->top;

s->size--;

s->top = temp->next;

delete temp;

}

*main.cpp*

#include <iostream>

#include "stack.h"

using namespace std;

int main() {

string input;

getline(cin, input);

struct stack \*tags = stack\_init();

bool isRight = true;

for (int i = 0; i < input.size(); i++) {

if (input[i] == '<') {

string current;

i++;

for (; i < input.length() && input[i] != '>'; i++) {

if (input[i] == ' ') {

while (i < input.length() && input[i] != '>') i++;

break;

}

current += input[i];

}

if (current[0] == '/') {

string temp = current;

current.erase(0, 1);

if (stack\_is\_empty(tags)) {

cout << "No pair for <" << temp << ">" << endl;

isRight = false;

continue;

}

if (current == stack\_top(tags)) {

stack\_pop(tags);

} else {

cout << "No pair for <" << temp << "> or wrong tag closing." << endl;

isRight = false;

continue;

}

} else {

stack\_push(tags, current);

}

}

}

while (!stack\_is\_empty(tags)) {

cout << "No pair for <" << stack\_top(tags) << ">" << endl;

isRight = false;

stack\_pop(tags);

}

if (isRight) {

cout << "Everything is right." << endl;

}

return 0;

}

*Задание 2*

Даны две непустые очереди, которые содержат одинаковое количество элементов. Объединить очереди в одну, в которой элементы исходных очередей чередуются.

Даны две непустые очереди. Элементы каждой из очередей упорядочены по возрастанию. Объединить очереди в одну с сохранением упорядоченности элементов.

**Реализация**

*queue.h*

#ifndef TASK2\_QUEUE\_H

#define TASK2\_QUEUE\_H

struct queuenode {

int value;

struct queuenode \*next;

};

struct queue {

struct queuenode \*head;

struct queuenode \*tail;

int size;

};

struct queue \*queue\_init();

void queue\_push(struct queue \*q, int value);

bool queue\_is\_empty(struct queue \*q);

int queue\_front(struct queue \*q);

void queue\_pop(struct queue \*q);

#endif //TASK2\_QUEUE\_H

*queue.cpp*

#include "queue.h"

struct queue \*queue\_init() {

struct queue \*temp = new struct queue;

temp->size = 0;

temp->head = nullptr;

temp->tail = nullptr;

return temp;

}

void queue\_push(struct queue \*q, int value) {

struct queuenode \*node = new struct queuenode;

node->value = value;

node->next = nullptr;

if (queue\_is\_empty(q)) {

q->head = node;

q->tail = node;

q->size++;

return;

}

q->tail->next = node;

q->tail = node;

q->size++;

}

bool queue\_is\_empty(struct queue \*q) {

if (q->size == 0) return 1;

return 0;

}

int queue\_front(struct queue \*q) {

return q->head->value;

}

void queue\_pop(struct queue \*q) {

if (queue\_is\_empty(q)) return;

struct queuenode \*temp = q->head;

q->head = temp->next;

q->size--;

delete temp;

}

*main.cpp*

#include <iostream>

#include "queue.h"

using namespace std;

void queue\_fill(struct queue \*A, struct queue \*B);

struct queue \*to\_one\_queue(struct queue \*A, struct queue \*B);

struct queue \*to\_one\_queue\_sorted(struct queue \*A, struct queue \*B);

int main() {

struct queue \*A = queue\_init(), \*B = queue\_init();

queue\_fill(A, B);

struct queue \*result;

result = to\_one\_queue(A, B);

cout << "To one queue: ";

while (!queue\_is\_empty(result)) {

cout << queue\_front(result) << " ";

queue\_pop(result);

}

cout << endl;

queue\_fill(A, B);

result = to\_one\_queue\_sorted(A, B);

cout << "To one queue sorted: ";

while (!queue\_is\_empty(result)) {

cout << queue\_front(result) << " ";

queue\_pop(result);

}

cout << endl << endl;

return 0;

}

void queue\_fill(struct queue \*A, struct queue \*B) {

cout << "A: ";

for (int i = 3; i <= 11; i += 2) {

cout << i << " ";

queue\_push(A, i);

}

cout << endl << "B: ";

for (int i = 1; i <= 13; i += 3) {

cout << i << " ";

queue\_push(B, i);

}

cout << endl;

}

struct queue \*to\_one\_queue(struct queue \*A, struct queue \*B) {

struct queue \*total = queue\_init();

while (!queue\_is\_empty(A) && !queue\_is\_empty(B)) {

queue\_push(total, queue\_front(A));

queue\_pop(A);

queue\_push(total, queue\_front(B));

queue\_pop(B);

}

return total;

}

struct queue \*to\_one\_queue\_sorted(struct queue \*A, struct queue \*B) {

struct queue \*total = queue\_init();

while (!queue\_is\_empty(A) || !queue\_is\_empty(B)) {

if (queue\_is\_empty(A)) {

while (!queue\_is\_empty(B)) {

queue\_push(total, queue\_front(B));

queue\_pop(B);

}

break;

}

if (queue\_is\_empty(B)) {

while (!queue\_is\_empty(A)) {

queue\_push(total, queue\_front(A));

queue\_pop(A);

}

break;

}

if (queue\_front(A) < queue\_front(B)) {

queue\_push(total, queue\_front(A));

queue\_pop(A);

} else {

queue\_push(total, queue\_front(B));

queue\_pop(B);

}

}

return total;

}

**Вывод:** изучены принципы работы с абстрактными структурами данных стэк и очередь путём решения поставленных задач, подразумевающих использование изучаемых структур.