Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Реализация очереди через односвязный список и динамический массив»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2024, Минск

1. **Реализация очереди через односвязный список**

#include <iostream>

#include <string>

#include <Windows.h>

#include <regex>

using namespace std;

regex checknum("[\\-]?[0-9]+"); // регулярное выражение для проверки корректности ввода

struct Node { // структура узла очереди

int num; // информационное поле

Node\* next = NULL; // указатель на следующий элемент(по умолчанию NULL)

};

struct Myqueue { // структура для самой очереди

long long int size = 0; // размер очереди

Node\* first = NULL; // указатель на первый узел

Node\* last = NULL; // указатель на последний узел

};

void push(Myqueue\*&, int); // прототип функции добавления элемента

void pop(Myqueue\*&); // прототип функции удаления

int front(Myqueue\*&); // прототип функции, которая возвращает первый элемент очереди

int back(Myqueue\*&); // прототип функции, которая возвращает последний элемент очереди

long long int size(Myqueue\*&); // прототип фукции, которая возвращает количество элементов очереди

bool empty(Myqueue\*&); // прототип функции для проверки очереди на пустоту

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char choice; // переменная для выбора опции

string c;

Myqueue\* q = new Myqueue; // создаем нашу очередь, выделяем память

do {

cout << "Введите:\n";

cout << "1 - для добавления элемента в очередь\n";

cout << "2 - для удаления элемента из очереди\n";

cout << "3 - для проверки на пустоту\n";

cout << "4 - для вывода первого элемента\n";

cout << "5 - для вывода размера очереди\n";

cout << "6 - для вывода последнего элемента\n";

cout << "7 - для выхода\n";

getline(cin, c);

while (c == "") {

getline(cin, c);

}

while (c.length() != 1) {

cout << "Некорректный ввод\n";

getline(cin, c);

}

choice = c[0];

switch (choice) {

case '1': {

int n;

cout << "Введите число, которое хотите добавить в очередь\n";

string b;

getline(cin, b);

while (!regex\_match(b, checknum)) {

cout << "Некорректный ввод\n";

getline(cin, b);

}

n = stoi(b);

push(q, n);

break;

}

case '2': {

pop(q);

break;

}

case '3': {

if (empty(q)) {

cout << "Очередь пуста\n";

}

else {

cout << "Очередь не пуста\n";

}

break;

}

case '4': {

int k = front(q);

if (k) {

cout << k << '\n';

}

break;

}

case '5': {

int k = size(q);

cout << "Количество элементов в очереди " << k << '\n';

break;

}

case '6': {

int k = back(q);

if (k) {

cout << k << '\n';

}

break;

}

case'7': {

break;

}

default: {

cout << "Некорректный ввод\n";

break;

}

}

} while (choice != '7');

}

void push(Myqueue\*& q, int a) { // функция добавления элемента

if (q->first == NULL) {

q->first = q->last = new Node;

q->first->num = q->last->num = a;

}

else {

q->last = q->last->next = new Node;

q->last->num = a;

}

q->size++;

}

void pop(Myqueue\*& q) { // функция удаления элемента

if (q->first == NULL) {

exit(-1);

}

else {

Node\* t = q->first;

q->first = q->first->next;

if (q->first == NULL) {

q->last = NULL;

}

delete t;

q->size--;

}

}

int front(Myqueue\*& q) { // функция возвращения значения первого элемента

if (q->first == NULL) {

cout << "Очередь пуста\n";

return 0;

}

else {

return q->first->num;

}

}

int back(Myqueue\*& q) { // функция возвращения значения последнего элемента

if (q->last == NULL) {

cout << "Очередь пуста\n";

return 0;

}

else {

return q->last->num;

}

}

long long int size(Myqueue\*& q) { // функция возвращения размера очереди(количества элементов)

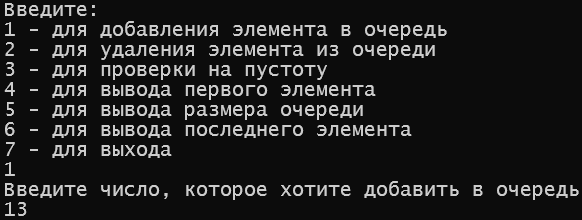
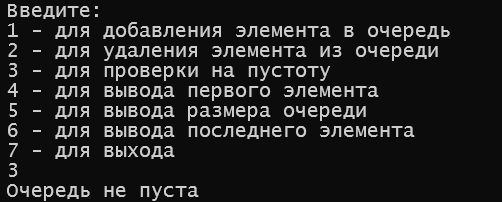
return q->size;

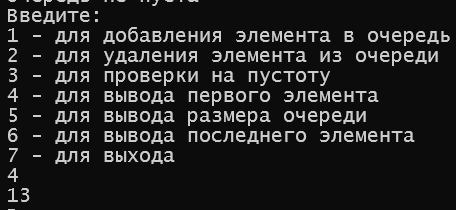
}

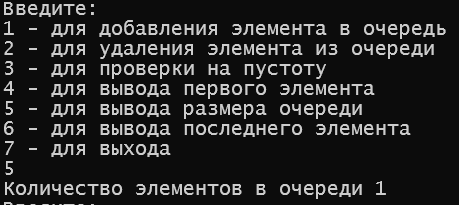
bool empty(Myqueue\*& q) { // функция проверки на пустоту

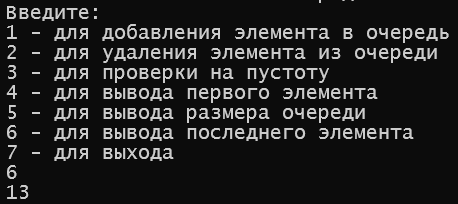
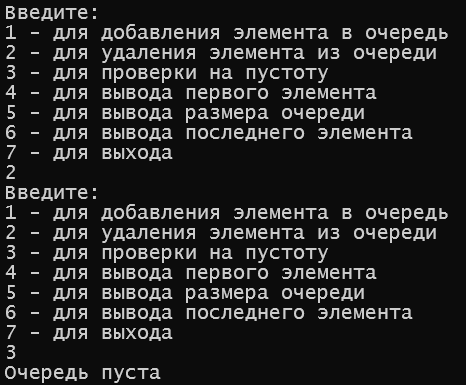
return q->first == NULL;

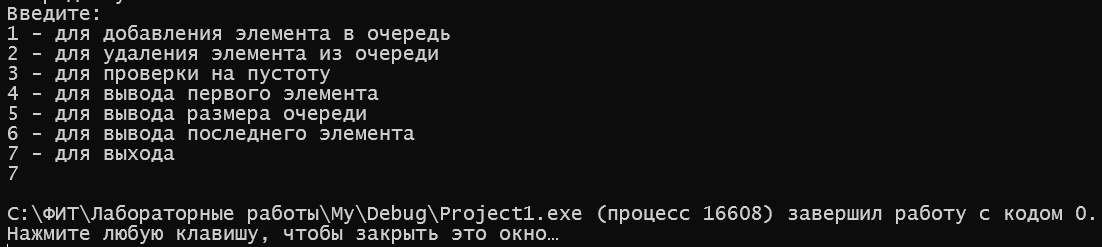
}

Результаты выполнения:









1. **Реализация очереди через динамический массив**

#include <iostream>

#include <string>

#include <Windows.h>

#include <regex>

using namespace std;

int size\_queue = 5; // максимально доступный размер очереди на данный момент

int\* queue = (int\*)malloc(size\_queue \* sizeof(int));// создаем динамический массив под очередь, выделяем память

int back\_q = -1; // индекс конца очереди

regex checknum("[\\-]?[0-9]+"); // решулярное выражение для проверки ввода цифры

void push(int); // прототип функции добавления элемента

void pop(); // прототип функции удаления элемента

int front(); // прототип функции, которая возвращает первый элемент

int back(); // прототип функции, которая возвращает последний элемент

int size(); // прототип функции, которая возвращает количество элементов в очереди

bool empty(); // прототип функции проверки очереди на пустоту

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char choice; // переменная для выбора опции

string c;

do {

cout << "Введите:\n";

cout << "1 - для добавления элемента в очередь\n";

cout << "2 - для удаления элемента из очереди\n";

cout << "3 - для проверки на пустоту\n";

cout << "4 - для вывода первого элемента\n";

cout << "5 - для вывода размера очереди\n";

cout << "6 - для вывода последнего элемента\n";

cout << "7 - для выхода\n";

getline(cin, c);

while (c == "") {

getline(cin, c);

}

while (c.length() != 1) {

cout << "Некорректный ввод\n";

getline(cin, c);

}

choice = c[0];

switch (choice) {

case '1': {

int n;

cout << "Введите число, которое хотите добавить в очередь\n";

string b;

getline(cin, b);

while (!regex\_match(b, checknum)) {

cout << "Некорректный ввод\n";

getline(cin, b);

}

n = stoi(b);

push(n);

break;

}

case '2': {

pop();

break;

}

case '3': {

if (empty()) {

cout << "Очередь пуста\n";

}

else {

cout << "Очередь не пуста\n";

}

break;

}

case '4': {

int k = front();

if (k) {

cout << k << '\n';

}

break;

}

case '5': {

int k = size();

cout << "Количество элементов в очереди " << k << '\n';

break;

}

case '6':{

int k = back();

if (k) {

cout << k << '\n';

}

break;

}

case'7':{

free(queue);

break;

}

default: {

cout << "Некорректный ввод\n";

break;

}

}

} while (choice != '7');

}

void resize(int siz) { // функция для изменения размера очереди(увеличение размера)

int\* temp = (int\*)realloc(queue, siz\* sizeof(int)); // дополнительно выделяем память

queue = temp;

}

void push(int a) { // функция добавления элемента

if (back\_q == size\_queue - 1) { // если элемент не помещается

resize(size\_queue \* 2); // квеличиваем размер очереди

}

back\_q++;

queue[back\_q] = a;

}

void pop() { // функция удаления элемента

if (empty()) {

cout << "Очередь пуста\n";

return;

}

for (int i = 1; i <= back\_q; i++) {

queue[i - 1] = queue[i];

}

back\_q--;

}

int front() { // функция возвращения значения первого элемента

if (empty()) {

cout << "Очередь пуста\n";

return 0;

}

return queue[0];

}

int back() { // функция возвращения значения последнего элемента

if (empty()) {

cout << "Очередь пуста\n";

return 0;

}

return queue[back\_q];

}

int size() { // функция возвращения размера очереди(количества элементов)

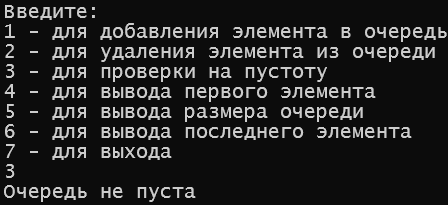
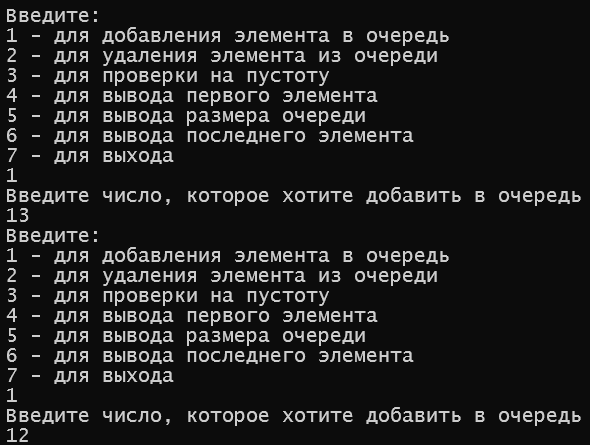
return back\_q + 1;

}

bool empty() { // функция проверки на пустоту

return back\_q == -1;

}

Результат выполнения:

