БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

**Динамические структуры данных. Очередь. Стек. Тестирование кода.**

**По:** Основы алгоритмизации и программирования

**Выполнил:** Корнелюк

Валентин Владимирович

1 курс 4 группа ПОИТ

**Преподаватель:**

Белодед Николай Иванович

**г. Минск**

2023 г.

**Оглавление**

[Построение и просмотр очереди, добавление и удаление звеньев из очереди. 3](#_Toc131274956)

[Приведем пример программы, реализующей действия со стеком. 10](#_Toc131274957)

Построение и просмотр очереди, добавление и удаление звеньев из очереди.Добавлена функция приоритетной вставки элемента, в зависимости от значения его информационного поля

#include<iostream> // Подключение библиотеки для ввода-вывода на консоль

using namespace std; // Использование стандартного пространства имен

struct node // Определение структуры узла списка

{

int elem; // Целочисленное значение, хранимое в узле

node\* sled; // Указатель на следующий узел в списке

};

class Spisok { // Определение класса списка

private:

node\* no, \* ko; // Указатели на начало и конец списка соответственно

int klad; // Поле для хранения удаленного элемента

public:

Spisok() { no = ko = NULL; } // Конструктор, устанавливающий начальные значения полей

void POSTROENIE(); // Метод для построения списка

void VYVOD(); // Метод для вывода списка на консоль

void DOBAVLENIE(int); // Метод для добавления элемента в конец списка

int Set\_Udal() { return klad; }// Метод для получения удаленного элемента

void YDALENIE(); // Метод для удаления элемента из начала списка

void OCHISTKA(); // Метод для освобождения памяти, занятой списком

void PRIORITY\_VKL(int el);

};

void main() // Определение главной функции программы

{

setlocale(LC\_ALL, "Rus"); // Установка локали для корректного отображения русских символов на консоли

Spisok A; // Создание объекта списка

int el; // Целочисленная переменная для ввода элементов

A.POSTROENIE(); // Вызов метода для построения списка

A.VYVOD(); // Вызов метода для вывода списка на консоль

cout << "Введите добавляемый элемент: ";

cin >> el; // Ввод элемета с клавиатуры

A.DOBAVLENIE(el); A.VYVOD(); // Добавление элемента в конец списка и вывод списка на консоль

cout << "Удалим элемент из очереди.\n";

A.YDALENIE(); A.VYVOD(); // Удаление элемента из начала списка и вывод списка на консоль

el = A.Set\_Udal(); // Получение удаленного элемента

cout << "Информационное поле удаленного звена: " << el << endl;

cout << "Введите элемент для приоритетной вставки: ";

cin >> el;

A.PRIORITY\_VKL(el);

A.VYVOD();

A.OCHISTKA(); // Освобождение памяти, занятой списком

cout << endl;

cout << "\n";

system("PAUSE"); // Приостановка работы программы до нажатия пользователем любой клавиши

}

void Spisok::POSTROENIE()

//Построение очереди на базе однонаправленного списка

//без заглавного звена.

//no - указатель на начало очереди.

//ko - указатель на конец очереди.

{

node\* r; // Создание указателя на структуру node

int el; // Создание переменной el для хранения элементов очереди, вводимых пользователем

cout << "Вводите элементы очереди:\n";

cin >> el; // Чтение первого элемента очереди

if (el != 0)

{

r = new (node); // Выделение памяти для нового элемента и присвоение указателю r его адреса

(\*r).elem = el; // Присвоение новому элементу значения el

(\*r).sled = NULL; // Установка указателя на следующий элемент в NULL

no = r; // Установка указателя no на начало очереди (на созданный новый элемент)

ko = r; // Установка указателя ko на конец очереди (также на созданный новый элемент)

cin >> el; // Чтение следующего элемента очереди

while (el != 0)

{

r = new (node); // Выделение памяти для нового элемента и присвоение указателю r его адреса

(\*r).elem = el; // Присвоение новому элементу значения el

(\*r).sled = NULL; // Установка указателя на следующий элемент в NULL

(\*ko).sled = r; // Установка указателя на следующий элемент для последнего элемента в списке (ко)

ko = r; // Установка указателя ko на последний элемент в списке

cin >> el; // Чтение следующего элемента очереди

}

}

else

{

r = NULL; // Если в очереди нет элементов, то указатель r устанавливается в NULL

no = r; // Установка указателя no на начало очереди (NULL)

ko = r; // Установка указателя ko на конец очереди (NULL)

}

}

void Spisok::VYVOD()

//Вывод содержимого очереди.

//no - указатель на начало очереди.

//ko - указатель на конец очереди.

{

node\* r; // Создание указателя на структуру node

cout << "Очередь: "; r = no; // начало очереди устанавливается в r

while (r != NULL) // пока не достигнут конец очереди

{

cout << (\*r).elem << " "; // выводится значение элемента

r = (\*r).sled; // переход к следующему элементу

}

cout << endl; // переход на новую строку

}

void Spisok::DOBAVLENIE(int el)

//Добавление звена с информационным полем el к очереди,

//определенной указателями no и ko.

{

node\* r; // Создание указателя на структуру node

r = new (node); //Выделение памяти под новый узел

(\*r).elem = el; (\*r).sled = NULL; //Инициализация полей нового узла

if (no != NULL) //Если очередь не пуста

{

(\*ko).sled = r; //Указатель на следующий узел последнего элемента указывает на новый узел

ko = r; //Указатель на последний элемент теперь указывает на новый узел

}

else //Если очередь пуста

{

no = r; //Указатель на первый элемент теперь указывает на новый узел

ko = r; //Указатель на последний элемент теперь указывает на новый узел

}

}

void Spisok::YDALENIE()

//Удаление звена из очереди, определенной указателями

//no и ko, с помещением его информационного поля в

//параметр klad.

{

node\* q; // Объявление указателя на структуру node

if (no == NULL) // Проверяем, не является ли очередь пустой

cout << "Удалить нельзя, так как очередь пуста!\n";

else

{

klad = (\*no).elem; // Записываем значение информационного поля первого звена в параметр klad

q = no; // Запоминаем указатель на первое звено

no = (\*no).sled; // Сдвигаем указатель на начало очереди на следующее звено

delete q; // Освобождаем память, занимаемую удаленным звеном

}

}

void Spisok::OCHISTKA()

//Возврат выделенной памяти в "кучу".

{

node\* q;

q = no;// Объявление указателя на первый элемент очереди

if (no != NULL) // Если очередь пуста, то нет необходимости освобождать выделенную память

{

while (no != ko) // Проходимся по всем элементам очереди, начиная с первого

{

no = (\*q).sled; // Запоминаем указатель на текущий элемент

delete q; // Удаляем элемент

q = no; // Перемещаем указатель на следующий элемент

}

delete no; // Удаляем последний элемент

no = ko = NULL; // Обнуляем указатели на первый и последний элементы очереди

}

}

void Spisok::PRIORITY\_VKL(int el)

{

node\* p = no; // Указатели на текущий элемент, предыдущий элемент и новый элемент

node\* q = nullptr;

node\* r = new node;

r->elem = el; // Инициализация нового элемента

r->sled = nullptr;

if (no == nullptr) { // Если очередь пуста, добавляем первый элемент

no = r;

ko = r;

}

else { // Ищем элемент с меньшим приоритетом

while (p != nullptr && p->elem < el) {

q = p;

p = p->sled;

}

// Вставляем элемент в нужное место

if (p == no) { // Вставка в начало очереди

r->sled = no;

no = r;

}

else if (p == nullptr) { // Вставка в конец очереди

ko->sled = r;

ko = r;

}

else { // Вставка в середину очереди

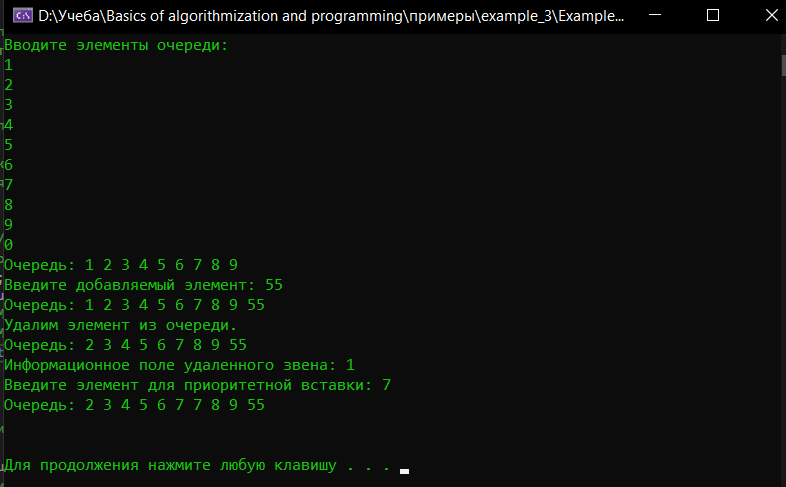
r->sled = p;

q->sled = r;

}

}

}



Приведем пример программы, реализующей действия со стеком. Формирование и вывод содержимого стека на экран дисплея. Удаление и вставка звена в стек. Добавлены функции: функция проверки на существование заданного элемента в стеке, функция, возвращающая верхний элемент стека, функция проверяющая есть ли в стеке хоть один элемент, функция вставки элемент после заданного, функция вставки элемента перед заданным, функция вставки вместо заданного.

#include<iostream>

using namespace std;

struct node // Описаниe структуры узла связанного списка

{

int elem; // информационное поле узла

node\* sled; // указатель на следующий узел

};

class Spisok // Описание класса Список

{

private:

node\* stk; // указатель на вершину стека

int klad; // переменная, хранящая значение удаленного элемента

public:

Spisok() { stk = NULL; } // конструктор класса, инициализирует указатель на NULL

int Set\_Stack() { return klad; } // метод класса, возвращает значение переменной klad

void POSTROENIE(); // метод класса, выполняющий построение списка

void VYVOD(); // метод класса, выводящий на экран содержимое списка

void W\_S(int); // метод класса, добавляющий элемент в стек

void YDALENIE(); // метод класса, удаляющий элемент из стека

void OCHISTKA(); // метод класса, освобождающий выделенную память для списка

bool contains(int el); //метод класса, проверяющий есть ли заданный элемент в стеке

int GETFIRST(); // метод класса, возвращающий верхний элемент стека

bool ISEMPTY(); // метод класса, проверяющий содержит ли стек элементы

void ADDBEFORE(int value, int after); // метод класса, добавляющий элемент в стек перед заданным числом

void ADDAFTER(int value, int after); // метод класса, добавляющий элемент в стек после заданного числа

void ADD\_ELM\_VMESTO(int old\_el, int new\_el); //метод класса, добавляющий элемент в стек вместо заданного числа

};

void main() // Определение главной функции программы

{

setlocale(LC\_ALL, "Rus"); // установка локали для отображения русского текста в консоли

Spisok A; // создание объекта класса Список

int el; // переменная для ввода элемента в стек

int t; // переменная для хранения значения удаленного элемента с верхушки стека

A.POSTROENIE(); // вызов метода POSTROENIE() для построения списка

A.VYVOD(); // вывод на экран содержимого списка

cout << "Введите вставляемый элемент: ";

cin >> el; // Ввод элемета с клавиатуры

A.W\_S(el); // добавление элемента в стек

A.VYVOD(); // вывод на экран содержимого списка

cout << "Верхним элементом стека является число " << A.GETFIRST() << endl;

cout << "Удалим элемент из стека.\n";

A.YDALENIE(); // удаление элемента из стека

t = A.Set\_Stack(); // получение значения удаленного элемента

cout << "Из стека было извлечено число... " << t << endl;

A.VYVOD(); // вывод на экран содержимого списка

cout << "Введите число, перед которым хотите вставить элемент: ";

int el1;

cin >> el1;

cout << "Введите число, которое хотите вставить: ";

cin >> el;

A.ADDBEFORE(el, el1);

A.VYVOD();

cout << "Введите число послу которого хотите вставить элемент: ";

cin >> el1;

cout << "Введите число, которое хотите вставить: ";

cin >> el;

A.ADDAFTER(el, el1);

A.VYVOD();

cout << "Введите число вместо которого хотите вставить элемент: ";

cin >> el1;

cout << "Введите число, которое хотите вставить: ";

cin >> el;

A.ADD\_ELM\_VMESTO(el1, el);

A.VYVOD();

cout << "Введите элемент для проверки на существование в стеке: ";

cin >> el;

if (A.contains(el))

cout << "Элемент есть в стеке" << endl;

else

cout << "Данного элемента нет в стеке";

A.OCHISTKA(); // освобождение выделенной памяти для списка

cout << "\n";

system("PAUSE"); // приостановка работы программы до нажатия любой клавиши

}

void Spisok::POSTROENIE()

//Построение стека, заданного указателем stk.

{

node\* t; // Создание указателя на структуру node

int el; // Создание переменной el для хранения элементов стеки, вводимых пользователем

cout << "Вводите элементы стека: ";

cin >> el; // Ввод элемета с клавиатуры

while (el != 0) // пока не введен 0

{

t = new (node); // выделение памяти под новый элемент стека

(\*t).elem = el; (\*t).sled = stk; // инициализация нового элемента стека

stk = t; // указатель на новый элемент стека становится вершиной стека

cin >> el; // Ввод элемета с клавиатуры

}

}

void Spisok::VYVOD() // Вывод содержимого стека, заданного указателем stk.

{

node\* t; // создаем указатель на узел

cout << "Содержимое стека: ";

t = stk; // присваиваем указателю на узел адрес верхушки стека

while (t != NULL) // пока указатель не равен нулю

{

cout << (\*t).elem << " "; // выводим значение элемента стека

t = (\*t).sled; // переходим к следующему элементу стека

}

cout << endl; // выводим символ новой строки

}

void Spisok::W\_S(int el)

//Помещение элемента el в стек stk.

{

node\* q; // Создаем указатель на узел списка

q = new (node); // Выделяем память под новый узел списка

(\*q).elem = el; // Присваиваем значение элемента в информационное поле нового узла

(\*q).sled = stk; // Присваиваем указателю на следующий элемент в списке значение текущего верхнего элемента стека

stk = q; // Обновляем указатель на верхний элемент стека на новый элемент

}

void Spisok::YDALENIE()

//Удаление элемента из стека, заданного указателем stk.

//Значение информационного поля удаляемого элемента

//помещается в параметр klad.

{

node\* q; // Создаем указатель на узел списка

if (stk == NULL) // проверяем, что стек не пуст

cout << "Стек пуст!\n";

else

{

klad = (\*stk).elem; q = stk; // сохраняем значение информационного поля верхнего элемента

stk = (\*stk).sled; delete q; // удаляем верхний элемент

}

}

void Spisok::OCHISTKA()

//Возврат выделенной памяти в "кучу".

{

node\* t, \* q;

t = stk;

if (t != NULL) // проверяем, что стек не пустой

{

q = (\*t).sled; // устанавливаем указатель q на следующий элемент стека

while (q != NULL) // пока не достигнут конец стека

{

delete t; // удаляем текущий элемент

t = q; // устанавливаем указатель t на следующий элемент

q = (\*q).sled; // устанавливаем указатель q на следующий элемент стека

}

delete t; // удаляем последний элемент

}

}

bool Spisok::contains(int el) { // Поиск элемента в стеке, заданном указателем stk.

// Функция возвращает true, если элемент el присутствует в стеке, и false, если отсутствует.

node\* t = stk; // начинаем поиск с вершины стека

while (t != NULL) { // пока не достигнут конец стека

if ((\*t).elem == el) { // если текущий элемент равен искомому

return true; // возвращаем true

}

t = (\*t).sled; // переходим к следующему элементу

}

return false; // если элемент не найден, возвращаем false

}

bool Spisok::ISEMPTY() {

if (stk == NULL) {

return false; // стек пуст

}

else {

return true; // стек содержит элементы

}

}

int Spisok::GETFIRST()

{

if (stk == NULL) // если стек пустой

{

cout << "Стек пустой!\n";

return 0;

}

else // если стек не пустой

{

return (\*stk).elem; //возвращаем верхний элемент стека

}

}

void Spisok::ADDBEFORE(int value, int before) {

if (stk == NULL) { // Если стек пуст, выводим сообщение об ошибке и прерываем функцию

cout << "Ошибка: стек пуст!" << endl;

return;

}

node\* t = stk; // устанавливаем указатель на начало стека

while (t != NULL) { // Проходим по элементам стека

if ((\*t).elem == before) { // Если найден элемент со значением after, добавляем новый элемент перед ним и прерываем функцию

node\* new\_node = new node; // Создаем новый узел

(\*new\_node).elem = value; // Устанавливаем значение нового узла

(\*new\_node).sled = (\*t).sled; // Устанавливаем связь нового узла со следующим элементом

(\*t).sled = new\_node; // Устанавливаем связь текущего элемента со вставленным элементом

return;

}

t = (\*t).sled; // переходим к следующему элементу

}

cout << "Ошибка: элемент " << before << " не найден в стеке!" << endl; // Если элемент не найден, выводим сообщение об ошибке

}

void Spisok::ADDAFTER(int value, int after) {

node\* t = stk; // устанавливаем указатель на начало стека

node\* prev = NULL; // Устанавливаем указатель на предыдущий элемент

while (t != NULL) { // Перебираем элементы стека

if ((\*t).elem == after) { // Если нашли элемент, после которого нужно вставить новый элемент

node\* newNode = new node; // Создаем новый элемент и заполняем его значениями

(\*newNode).elem = value;

(\*newNode).sled = t;

if (prev == NULL) { // Если предыдущий элемент не задан, значит, вставляем новый элемент перед первым элементом

stk = newNode;

}

else { // Иначе вставляем новый элемент между двумя существующими элементами

(\*prev).sled = newNode;

}

return; // Завершаем функцию после добавления элемента

}

prev = t; // Перемещаем указатель на предыдущий элемент

t = (\*t).sled; // Перемещаем указатель на следующий элемент

}

// Если элемент не найден, выводим сообщение об ошибке

cout << "Ошибка: элемент " << after << " не найден в стеке" << endl;

}

void Spisok::ADD\_ELM\_VMESTO(int old\_el, int new\_el) {

node\* t = stk; // устанавливаем указатель на начало списка

node\* prev = NULL; // устанавливаем указатель на предыдущий элемент

node\* to\_del = NULL; // устанавливаем указатель на элемент, который необходимо удалить

while (t != NULL) { // проходим по всему списку, пока не достигнем конца

if (t->sled->elem == old\_el) // проверяем, является ли следующий элемент текущего элемента искомым элементом

prev = t; // если да, то сохраняем указатель на предыдущий элемент

if ((\*t).elem == old\_el) { // если текущий элемент - искомый элемент

node\* new\_node = new node; // создаем новый элемент

(\*prev).sled = new\_node; // предыдущий элемент теперь должен указывать на новый элемент

(\*new\_node).elem = new\_el; // присваиваем новому элементу значение new\_el

(\*new\_node).sled = (\*t).sled; // новый элемент теперь должен указывать на элемент, который шел после old\_el

delete t; // удаляем элемент, который содержал значение old\_el

return;

}

t = (\*t).sled; // переходим к следующему элементу списка

}

cout << "Элемент " << old\_el << " не найден в стеке" << endl; // если искомый элемент не найден, выводим сообщение об ошибке

}

