Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Тестирование кода

По дисциплине «Основы алгоритмизации и программирования»

На тему «Виды очередей»

Выполнил:

Студент 1 курса 7 группы

Ленкевич Павел Андреевич

Преподаватель: Белодед Н. И.

2024, Минск

# Введение

В этом документе размещено тестирование двух кодов из лекции на тему «Виды очередей». В каждую программу была добавлена новая функция, которая подсчитывает количество элементов в очереди. Большинство строк имеют пояснение в комментариях, а после кода представлен результат запуска и работы программы. Для того, чтоб быстро увидеть определённый код или его результат работы воспользуйтесь навигацией.

# Тестирование кода с лекции

## Дек

#include <iostream> // Подключение стандартной библиотеки c++

using namespace std; // Указываем на использование пространства имён std, что позволяет не писать "std::" перед операторами ввода/вывода

// Определение структуры узла для дека

struct node {

int elem; // элемент узла

node\* sled; // указатель на следующий узел

};

// Определение класса Spisok, который представляет собой дек

class Spisok {

private:

node\* id, \* rd; // указатели на начало и конец дека

int el\_left, el\_right; // элементы слева и справа

public:

// Методы класса Spisok

void POSTROENIE(); // построение дека

void VYVOD(); // вывод дека

// Новая функция COUNT в программе

int COUNT(); // подсчет количества элементов в деке

void VSTAV1(int); // вставка элемента справа

void VSTAV2(int); // вставка элемента слева

int SelElLeft() { return el\_left; } // выбор элемента слева

int SelElRight() { return el\_right; } // выбор элемента справа

void YDALE1(); // удаление элемента справа

void YDALE2(); // удаление элемента слева

void OCHISTKA(); // очистка дека

};

// Главная функция

void main()

// В данном случае эта функция задаёт общий вид выполняемой программы

// Объединяет и использует все представленные в коде функции

{

setlocale(LC\_ALL, "Rus"); // установка русского языка

Spisok A; // создание объекта класса Spisok

int el; // переменная для хранения элемента

// Построение и вывод дека

A.POSTROENIE(); A.VYVOD();

// Выводим количество элементов в деке

cout << "Количество элементов в деке: " << A.COUNT() << endl;

// Добавление элемента справа

cout << "Добавим звено справа.\n";

cout << "Введите элемент добавляемого звена: ";

cin >> el;

A.VSTAV1(el); A.VYVOD();

// Выводим количество элементов в деке

cout << "Количество элементов в деке: " << A.COUNT() << endl;

// Добавление элемента слева

cout << "Добавим звено слева.\n";

cout << "Введите элемент добавляемого звена: ";

cin >> el;

A.VSTAV2(el); A.VYVOD();

// Выводим количество элементов в деке

cout << "Количество элементов в деке: " << A.COUNT() << endl;

// Удаление элемента справа

cout << "Удалим звено справа.\n";

A.YDALE1(); A.VYVOD();

// Выводим количество элементов в деке

cout << "Количество элементов в деке: " << A.COUNT() << endl;

// Удаление элемента слева

cout << "Удалим звено слева.\n";

A.YDALE2(); A.VYVOD();

// Выводим количество элементов в деке

cout << "Количество элементов в деке: " << A.COUNT() << endl;

// Очистка дека

A.OCHISTKA();

cout << "\n";

system("PAUSE"); // Указываем на остановку/завершение программы

}

void Spisok::POSTROENIE()

// Функция используется для построения дека

// Она считывает элементы, введенные пользователем, и добавляет их в дек

{

node\* k; // Временный указатель на узел

int el; // Временная переменная для хранения элемента

// Выводим сообщение о вводе элементов дека

cout << "Вводите содержимое звеньев дека:\n";

cin >> el; // Считываем элемент

// Если элемент не равен нулю, то начинаем построение дека

if (el != 0) {

k = new(node); // Создаем новый узел

(\*k).elem = el; // Присваиваем элементу узла значение введенного элемента

(\*k).sled = NULL; // Устанавливаем указатель на следующий узел в NULL

id = k; // Устанавливаем указатель на начало дека на новый узел

rd = k; // Устанавливаем указатель на конец дека на новый узел

cin >> el; // Считываем следующий элемент

// Пока элемент не равен нулю, продолжаем добавлять элементы в дек

while (el != 0)

{

VSTAV1(el); // Добавляем элемент в конец дека

cin >> el; // Считываем следующий элемент

}

}

else

{

// Если первый введенный элемент равен нулю, то дек пуст

rd = NULL;

id = NULL;

}

}

void Spisok::VYVOD()

// Функция выводит элементы начиная с начала дека и продолжает до конца

// При достижения конца функция завершает вывод и переходит на новую строку

{

node\* k; // Временный указатель на узел

k = id; // Устанавливаем указатель на начало дека

cout << "Дек: "; // Выводим сообщение о начале дека

// Пока указатель на узел не равен NULL, продолжаем выводить элементы

while (k != NULL)

{

cout << (\*k).elem << " "; // Выводим элемент узла

k = (\*k).sled; // Переходим к следующему узлу

}

cout << endl; // Переходим на новую строку после вывода всех элементов

}

int Spisok::COUNT()

// Разработанная мной функция

// Она подсчитывает число элементов внутри дека возвращает это число

{

node\* k; // Временный указатель на узел

int count = 0; // Счетчик элементов

k = id; // Устанавливаем указатель на начало дека

// Пока указатель на узел не равен NULL, продолжаем считать элементы

while (k != NULL)

{

count++; // Увеличиваем счетчик

k = (\*k).sled; // Переходим к следующему узлу

}

return count; // Возвращаем количество элементов в деке

}

void Spisok::VSTAV1(int el)

// Функция создает новый узел, присваивает ему значение введенного элемента и добавляет его в конец дека

// Используется для добавления элемента в конец дека

{

node\* k; // Временный указатель на узел

k = new(node); // Создаем новый узел

(\*k).elem = el; // Присваиваем элементу узла значение введенного элемента

(\*k).sled = NULL; // Устанавливаем указатель на следующий узел в NULL

// Если дек не пуст, то добавляем элемент в конец дека

if (rd != NULL) {

(\*rd).sled = k; // Устанавливаем указатель на следующий узел текущего последнего узла на новый узел

rd = k; // Обновляем указатель на конец дека на новый узел

}

else {

// Если дек пуст, то новый узел становится единственным узлом в деке

rd = k;

id = k;

}

}

void Spisok::VSTAV2(int el)

// Функция создает новый узел, присваивает ему значение введенного элемента и добавляет его в начало дека.

// Используется для добавления элемента в начало дека

{

node\* k; // Временный указатель на узел

k = new(node); // Создаем новый узел

(\*k).elem = el; // Присваиваем элементу узла значение введенного элемента

(\*k).sled = id; // Устанавливаем указатель на следующий узел на текущий первый узел

// Если дек не пуст, то добавляем элемент в начало дека

if (rd != NULL)

id = k;

else {

// Если дек пуст, то новый узел становится единственным узлом в деке

id = k;

rd = k;

}

}

void Spisok::YDALE1()

// Функция удаляет последний элемент дека

{

node\* z; // Временный указатель на узел

node\* k; // Временный указатель на узел

// Если в деке только один элемент

if (rd == id) {

el\_right = (\*rd).elem; // Сохраняем значение элемента

delete rd; // Удаляем узел

id = rd = NULL; // Устанавливаем указатели на начало и конец дека в NULL

cout << "Дек пуст!\n"; // Выводим сообщение о том, что дек пуст

}

else {

// Если в деке больше одного элемента

z = id; // Устанавливаем временный указатель на начало дека

k = (\*id).sled; // Устанавливаем временный указатель на второй узел дека

// Продолжаем, пока не достигнем конца дека

while (k != rd)

{

z = k; // Переходим к следующему узлу

k = (\*k).sled; // Переходим к следующему узлу

}

el\_right = (\*rd).elem; // Сохраняем значение последнего элемента

(\*z).sled = NULL; // Устанавливаем указатель на следующий узел предпоследнего узла в NULL

delete rd; // Удаляем последний узел

rd = z; // Обновляем указатель на конец дека

}

}

void Spisok::YDALE2()

// Функция для удаления элемента из начала дека

{

node\* q; // Временный указатель на узел

// Если дек не пуст

if (id != NULL) {

el\_left = (\*id).elem; // Сохраняем значение первого элемента

q = id; // Сохраняем указатель на первый узел

id = (\*id).sled; // Обновляем указатель на начало дека на второй узел

delete q; // Удаляем первый узел

}

else

cout << "Дек пуст!\n"; // Выводим сообщение о том, что дек пуст

}

void Spisok::OCHISTKA()

// Очистка дека \ удаления всех его элементов

// Функция начинает с начала дека и продолжает до конца, удаляя каждый узел

// Когда достигнут конец дека (NULL), метод завершает удаление узлов

{

node\* k, \* q; // Временные указатели на узлы

k = id; // Устанавливаем указатель на начало дека

// Если дек не пуст

if (k != NULL) {

q = (\*k).sled; // Сохраняем указатель на второй узел

// Пока не достигнем конца дека

while (q != NULL)

{

delete k; // Удаляем текущий узел

k = q; // Переходим к следующему узлу

q = (\*k).sled; // Обновляем указатель на следующий узел

}

delete k; // Удаляем последний узел

}

}

### Результат работы программы

## Очередь

#include <iostream> // Подключение стандартной библиотеки c++

using namespace std; // Указываем на использование пространства имён std, что позволяет не писать "std::" перед операторами ввода/вывода

// Определение структуры для элемента списка

struct node

{

int elem; // Значение элемента

node\* sled; // Указатель на следующий элемент

};

// Определение класса для списка

class Spisok {

private:

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

int klad; // Значение удаленного элемента

public:

void POSTROENIE(); // Функция построения списка

void VYVOD(); // Функция вывода списка

// Новая функция COUNT в программе

int COUNT(); // подсчет количества элементов в очереди

void DOBAVLENIE(int); // Функция добавления элемента в список

int Set\_Udal() { return klad; } // Функция возвращает значение удаленного элемента

void YDALENIE(); // Функция удаления элемента из списка

void OCHISTKA(); // Функция очистки списка

};

// Главная функция

void main()

// В данном случае эта функция задаёт общий вид выполняемой программы

// Объединяет и использует все представленные в коде функции

{

setlocale(LC\_ALL, "Rus"); // установка русского языка

Spisok A; // Создание объекта класса Spisok

int el; // Переменная для хранения элемента

A.POSTROENIE(); // Построение списка

A.VYVOD(); // Вывод списка

cout << "Количество элементов в очереди: " << A.COUNT() << endl; // Вывод количества элементов в очереди

cout << "Введите добавляемый элемент: ";

cin >> el;

A.DOBAVLENIE(el); // Добавление элемента в список

A.VYVOD(); // Вывод списка

cout << "Количество элементов в очереди: " << A.COUNT() << endl; // Вывод количества элементов в очереди

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

A.YDALENIE(); // Удаление элемента из списка

A.VYVOD(); // Вывод списка

cout << "Количество элементов в очереди: " << A.COUNT() << endl; // Вывод количества элементов в очереди

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

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

A.OCHISTKA(); // Очистка списка

cout << "\n";

system("PAUSE"); // Указываем на остановку \ завершение программы

}

void Spisok::POSTROENIE()

// Функция создает новые узлы для каждого введенного значения и добавляет их в конец списка, формируя очередь

{

node\* r; // Указатель на новый узел

int el; // Значение нового элемента

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

cin >> el;

// Если введенное значение не равно 0, начинаем построение списка

if (el != 0)

{

r = new(node); // Создаем новый узел

(\*r).elem = el; // Присваиваем ему введенное значение

(\*r).sled = NULL; // Устанавливаем указатель на следующий элемент в NULL

no = r; // Устанавливаем указатель на начало списка на новый узел

ko = r; // Устанавливаем указатель на конец списка на новый узел

cin >> el; // Считываем следующее значение

// Пока введенное значение не равно 0, продолжаем добавлять элементы в список

while (el != 0)

{

r = new(node); // Создаем новый узел

(\*r).elem = el; // Присваиваем ему введенное значение

(\*r).sled = NULL; // Устанавливаем указатель на следующий элемент в NULL

(\*ko).sled = r; // Устанавливаем указатель на следующий элемент предыдущего узла на новый узел

ko = r; // Устанавливаем указатель на конец списка на новый узел

cin >> el; // Считываем следующее значение

}

}

else

{

// Если введенное значение равно 0, список пуст

r = NULL;

no = r;

ko = r;

}

}

void Spisok::VYVOD()

// Функция вывода элементов очереди

// Функция проходит по всем элементам списка, начиная с первого, и выводит их значения

// Завершает работу при достижении конца

{

node\* r; // Указатель для перемещения по списку

cout << "Очередь: ";

r = no; // Устанавливаем указатель на начало списка

// Пока указатель не равен NULL (пока не достигнем конца)

while (r != NULL)

{

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

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

}

cout << endl;

}

int Spisok::COUNT() {

node\* k; // Временный указатель на узел

int count = 0; // Счетчик элементов

k = no; // Устанавливаем указатель на начало очереди

// Пока указатель на узел не равен NULL, продолжаем считать элементы

while (k != NULL)

{

count++; // Увеличиваем счетчик

k = (\*k).sled; // Переходим к следующему узлу

}

return count; // Возвращаем количество элементов в очереди

}

void Spisok::DOBAVLENIE(int el)

// Функция добавляет элемент в конец очереди

// Функция создаёт новый узел, присваивает ему введённое значение и добавляет в конец

{

node\* r; // Указатель на новый узел

r = new(node); // Создаем новый узел

(\*r).elem = el; // Присваиваем ему введенное значение

(\*r).sled = NULL; // Устанавливаем указатель на следующий элемент в NULL

if (no != NULL) // Если список не пуст

{

(\*ko).sled = r; // Устанавливаем указатель на следующий элемент последнего узла на новый узел

ko = r; // Устанавливаем указатель на конец списка на новый узел

}

else // Если список пуст

{

no = r; // Устанавливаем указатель на начало списка на новый узел

ko = r; // Устанавливаем указатель на конец списка на новый узел

}

}

void Spisok::YDALENIE()

// Удаляет элемент из начала очереди

// Значение удалённого элемента сохраняется в переменной klad

{

node\* q; // Указатель для удаления узла

if (no == NULL) // Если список пуст

{

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

}

else

{

klad = (\*no).elem; // Сохраняем значение удаляемого элемента

q = no; // Сохраняем указатель на удаляемый узел

no = (\*no).sled; // Перемещаем указатель на начало списка на следующий узел

delete q; // Удаляем узел

}

}

void Spisok::OCHISTKA()

// Функция, которая очищает весь список

// Она проходит по всем элементам списка начиная с первого и удаляет их

// При удалении всех элементов указатели устанавливаются на NULL

{

node\* q; // Указатель для удаления узла

q = no; // Устанавливаем указатель на начало списка

// Если список не пуст

if (no != NULL)

{

// Пока не достигнем конца списка

while (no != ko)

{

no = (\*q).sled; // Перемещаем указатель на начало списка на следующий узел

delete q; // Удаляем узел

q = no; // Сохраняем указатель на текущий узел

}

delete no; // Удаляем последний узел

no = ko = NULL; // Устанавливаем указатели на начало и конец списка в NULL

}

}

### Результат работы программы

