Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

**ТЕСТИРОВАНИЕ**

По предмету “Основы алгоритмизации и программирования”

На тему: “Дек”

Выполнил:

Студент 1 курса 9 группы

Павлович Ян Андреевич

Преподаватель: Белодед Н.И.

2024, Минск

Содержание

1. Введение
2. Основная часть
   1. Исходный код с добавлением комментариев
   2. Тестирование построения дека
      1. Тестирование построения дека с пустым вводом
      2. Тестирование построения дека с одним элементом
      3. Тестирование построения дека с отрицательными элементами
   3. Тестирование вывода
   4. Тестирование добавления элементов
   5. Добавляем вставку по индексу
   6. Тестирование удаления элементов
3. Заключение

**Введение**

Данный реферат посвящен тестированию кода, реализующего структуру данных дек на основе односвязного списка. В представленном коде определен класс `Spisok`, представляющий собой дек, который позволяет добавлять и удалять элементы как с левого, так и с правого конца очереди. Класс включает в себя методы для построения, вывода содержимого, добавления и удаления элементов, а также для очистки дека.

Цель данного тестирования - провести тестирование методов класса `Spisok` с целью выявления возможных ошибок и недочетов в работе кода, а также обеспечения корректности его выполнения.

**Исходный код с добавлением комментариев**

#include <iostream>

// Подключаем библиотеку ввода-вывода

using namespace std;

// Используем пространство имен std

// Определение структуры node, представляющей каждый элемент в списке

struct node {

    int elem; // Элемент, хранящийся в узле

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

};

// Определение класса Spisok, представляющего дек (двустороннюю очередь)

class Spisok {

private:

    node\* ld, \* rd; // Указатели на левый и правый концы дека

    int el\_left, el\_right; // Переменные для хранения удаляемых элементов из дека

public:

    void POSTROENIE(); // Метод для построения дека

    void VYVOD(); // Метод для вывода содержимого дека

    void VSTAV1(int el); // Метод для вставки элемента в правый конец

    void VSTAV2(int el); // Метод для вставки элемента в левый конец

    int SetElLeft() { return el\_left; } // Метод для получения значения el\_left

    int SetElRight() { return el\_right; } // Метод для получения значения el\_right

    void YDALE1(); // Метод для удаления элемента из правого конца дека

    void YDALE2(); // Метод для удаления элемента из левого конца дека

    void OCHISTKA(); // Метод для очистки дека

};

void main() {

    // Устанавливаем локаль на "Рус" для вывода кириллицы

    setlocale(LC\_ALL, "Rus");

    // Создаем объект класса Spisok

    Spisok A;

    int el; // Переменная для хранения вводимого элемента

    // Построение дека и его вывод

    A.POSTROENIE();

    A.VYVOD();

    // Добавление элемента в правый конец дека

    cout << "Добавим звено справа.\n";

    cout << "Введите элемент добавляемого звена: ";

    cin >> el;

    A.VSTAV1(el);

    A.VYVOD();

    // Добавление элемента в левый конец дека

    cout << "Добавим звено слева.\n";

    cout << "Введите элемент добавляемого звена: ";

    cin >> el;

    A.VSTAV2(el);

    A.VYVOD();

    // Удаление элемента из правого конца дека

    cout << "Удалим звено справа.\n";

    A.YDALE1();

    A.VYVOD();

    cout << A.SetElRight() << endl;

    // Удаление элемента из левого конца дека

    cout << "Удалим звено слева.\n";

    A.YDALE2();

    A.VYVOD();

    cout << A.SetElLeft() << endl;

    // Очистка дека

    A.OCHISTKA();

    // Ожидание нажатия клавиши перед завершением программы

    cout << "\n";

    system("PAUSE");

}

// Определение метода POSTROENIE для построения дека

void Spisok::POSTROENIE() {

    node\* k;

    int el;

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

    cin >> el;

    if (el != 0) {

        // Создание первого узла, если введенный элемент не равен 0

        k = new node;

        k->elem = el; // Инициализация элемента узла

        k->sled = NULL; // Инициализация следующего узла как NULL

        ld = k; // Установка левого конца дека

        rd = k; // Установка правого конца дека

        cin >> el;

        // Добавление элементов справа, пока введенный элемент не равен 0

        while (el != 0) {

            VSTAV1(el);

            cin >> el;

        }

    }

    else {

        // Если введенный элемент равен 0, дека пустой

        rd = NULL;

        ld = NULL;

    }

}

// Определение метода VYVOD для вывода содержимого дека

void Spisok::VYVOD() {

    node\* k;

    k = ld; // Начинаем вывод с левого конца дека

    cout << "Дек: ";

    // Выводим каждый элемент дека

    while (k != NULL) {

        cout << k->elem << " ";

        k = k->sled; // Переходим к следующему узлу

    }

    cout << endl;

}

// Определение метода VSTAV1 для вставки элемента в правый конец дека

void Spisok::VSTAV1(int el) {

    node\* k;

    k = new node;

    k->elem = el; // Инициализируем элемент узла

    k->sled = NULL; // Следующий узел равен NULL

    if (rd != NULL) {

        // Если правый конец дека не пустой

        rd->sled = k; // Присоединяем новый узел к концу дека

        rd = k; // Обновляем правый конец дека

    }

    else {

        // Если правый конец дека пустой

        rd = k;

        ld = k;

    }

}

// Определение метода VSTAV2 для вставки элемента в левый конец дека

void Spisok::VSTAV2(int el) {

    node\* k;

    k = new node;

    k->elem = el; // Инициализируем элемент узла

    k->sled = ld; // Следующий узел указывает на текущий левый конец дека

    if (ld != NULL) {

        ld = k; // Обновляем левый конец дека

    }

    else {

        // Если левый конец дека пустой

        ld = k;

        rd = k;

    }

}

// Определение метода YDALE1 для удаления элемента из правого конца дека

void Spisok::YDALE1() {

    node\* z;

    node\* k;

    if (rd == ld) {

        // Если дека содержит только один элемент

        el\_right = rd->elem; // Сохраняем удаляемый элемент

        delete rd;

        ld = rd = NULL; // Очищаем дека

        cout << "Дек пуст!\n";

    }

    else {

        // Удаление последнего элемента

        z = ld;

        k = ld->sled;

        while (k != rd) {

            z = k;

            k = k->sled;

        }

        el\_right = rd->elem; // Сохраняем удаляемый элемент

        z->sled = NULL; // Удаляем последний элемент

        delete rd;

        rd = z;

    }

}

// Определение метода YDALE2 для удаления элемента из левого конца дека

void Spisok::YDALE2() {

    node\* q;

    if (ld != NULL) {

        el\_left = ld->elem; // Сохраняем удаляемый элемент

        q = ld; // Запоминаем удаляемый узел

        ld = ld->sled; // Обновляем левый конец дека

        delete q; // Удаляем узел

    }

    else {

        cout << "Дек пуст!\n";

    }

}

// Определение метода OCHISTKA для очистки дека

void Spisok::OCHISTKA() {

    node\* k, \* q;

    k = ld;

    if (k != NULL) {

        // Удаляем все узлы, начиная с левого конца дека

        q = k->sled;

        while (q != NULL) {

            delete k;

            k = q;

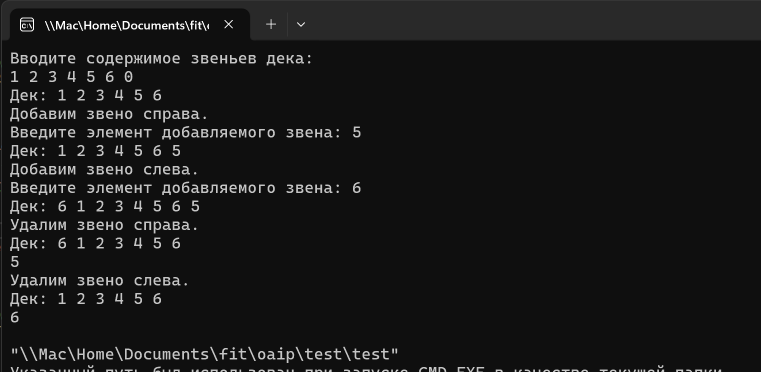
            q = k->sled;

        }

        delete k;

    }

}

****

**Тестирование построения дека**

Для тестирования построения дека нужно проверить метод `POSTROENIE` на корректность создания дека из ввода.

1. Проверка построения дека с пустым вводом:

Для тестирования метода `POSTROENIE` с пустым вводом в программу будет введено значение `0` в качестве первого элемента. Это указывает на пустой дек, который не должен содержать никаких элементов. После выполнения метода будет проверено, что указатели `ld` (левый конец дека) и `rd` (правый конец дека) равны `NULL`, что подтверждает правильное создание пустого дека.

// Проверяем, равны ли ld и rd NULL

if (ld == NULL && rd == NULL) {

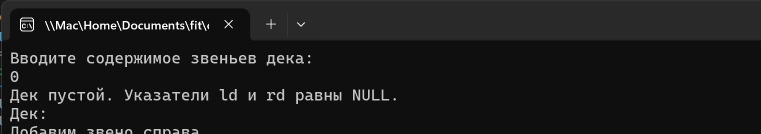
    cout << "Дек пустой. Указатели ld и rd равны NULL." << endl;

}

else {

    cout << "Ошибка: указатели ld и rd не равны NULL." << endl;

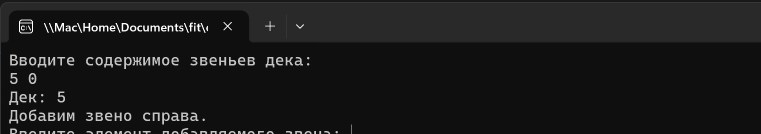
}



Проверка построения дека с пустым вводом была успешной: после ввода `0` метод `POSTROENIE` корректно идентифицировал пустой дек, оставив его без элементов. Указатели `ld` (левый конец дека) и `rd` (правый конец дека) оказались равны `NULL`, что соответствует ожидаемому результату для пустого дека и свидетельствует о правильной работе метода.

2. Проверка построения дека с одним элементом:

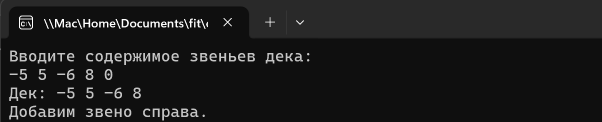
Для проверки построения дека с одним элементом в программу будет введено значение, отличное от `0` (например, `5`), в качестве первого элемента. Ввод завершится при вводе `0`. После выполнения метода `POSTROENIE` будет проверено, что дек содержит только один элемент, который был введен в начале, и что указатели `ld` (левый конец дека) и `rd` (правый конец дека) указывают на один и тот же узел, содержащий этот элемент.



Проверка показала, что метод `POSTROENIE` корректно создает дек с одним элементом. После выполнения метода дек содержит только введенный элемент, и указатели `ld` и `rd` правильно указывают на один и тот же узел, который содержит этот элемент. Это свидетельствует о правильной работе метода при построении дека с одним элементом.

3. Проверка работы с отрицательными числами:

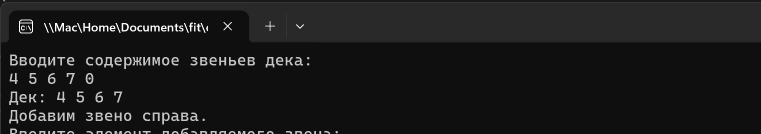
Для проверки работы метода `POSTROENIE` с отрицательными числами в программу будет введено несколько отрицательных значений в качестве элементов дека. Ввод завершится при вводе `0`. После выполнения метода `POSTROENIE` будет проверено, что дек содержит все введенные отрицательные числа в правильном порядке.

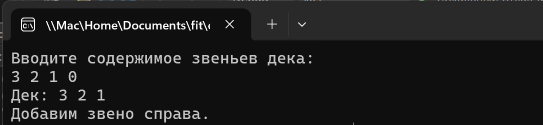


Проверка показала, что метод `POSTROЕНIE` корректно обрабатывает отрицательные числа и создает дек, содержащий все введенные элементы в том порядке, в котором они были введены. Указатели `ld` (левый конец дека) и `rd` (правый конец дека) также установлены правильно, указывая на начальный и конечный элементы дека соответственно. Это подтверждает, что метод успешно работает с отрицательными числами и создает дек в соответствии с ожиданиями.

**Тестирование вывода**

Метод `VYVOD` выводит содержимое дека на экран. Он проходит по всем элементам дека от левого конца к правому, начиная с указателя `ld`, и выводит значения каждого элемента, разделяя их пробелами. После того, как все элементы дека выведены, метод завершает вывод переносом строки. Это обеспечивает корректное отображение всех элементов дека в правильном порядке и формате.

****

****

**Тестирование добавления элементов**

Методы `VSTAV1` и `VSTAV2` отвечают за добавление элементов в дек соответственно справа и слева.

Метод `VSTAV1` (добавление справа):

1. Создание нового узла: Метод создает новый узел (`node`) и устанавливает его значение в элемент (`elem`) переданного аргумента `el`.

2. Добавление узла: Если правый конец дека (`rd`) не равен `NULL`, новый узел присоединяется к концу дека. Указатель `sled` предыдущего правого узла указывает на новый узел.

3. Обновление указателя `rd`: После добавления новый узел становится правым концом дека, и указатель `rd` обновляется, чтобы указывать на него.

4. Корректность работ: Метод корректно добавляет элементы справа в дек и обновляет указатель `rd`. Если дек изначально пустой, новый узел становится как левым (`ld`), так и правым (`rd`) концом дека.

Метод `VSTAV2` (добавление слева):

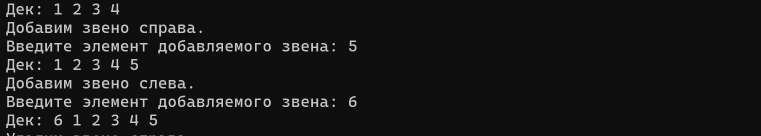
1. Создание нового узла: Метод создает новый узел (`node`) и устанавливает его значение в элемент (`elem`) переданного аргумента `el`.

2. Добавление узла: Если левый конец дека (`ld`) не равен `NULL`, новый узел присоединяется к началу дека. Указатель `sled` нового узла указывает на предыдущий левый узел.

3. Обновление указателя `ld`: После добавления новый узел становится левым концом дека, и указатель `ld` обновляется, чтобы указывать на него.

4. Корректность работы: Метод корректно добавляет элементы слева в дек и обновляет указатель `ld`. Если дек изначально пустой, новый узел становится как левым (`ld`), так и правым (`rd`) концом дека.

Методы `VSTAV1` и `VSTAV2` работают корректно, если они добавляют элементы в дек в нужном направлении (справа или слева) и если после добавления нового элемента дек остается связанным, а указатели `ld` и `rd` указывают на правильные концы дека. Это также проверяется путем вызова метода `VYVOD` для вывода содержимого дека и подтверждения того, что элементы выводятся в правильном порядке.

****

**Добавим вставку по индексу**

В ходе разработки программы для работы с деком была добавлена функция по вставке элемента по индексу. Эта функция позволяет пользователю вставлять элементы в произвольное место в деке, указывая индекс позиции для вставки.

Работа функции заключается в том, что она сначала запрашивает у пользователя индекс для вставки и значение элемента, который нужно вставить. Затем функция проверяет, что индекс находится в пределах допустимого диапазона, и выполняет вставку элемента в нужное место в деке.

Если индекс равен нулю, элемент вставляется в начало дека, а если равен размеру дека, элемент вставляется в его конец. В остальных случаях функция ищет узел, перед которым нужно вставить новый элемент, и вставляет его. После завершения операции функция выводит сообщение об успешной вставке.

...

private:

int size = 0; // Объявляем переменную для высчитывания размера дека

public:

// Добавляем функцию, возвращающую размер дека

int getSize() {

    return size;

}

...

// Добавление элементов справа, пока введенный элемент не равен 0

while (el != 0) {

    VSTAV1(el);

    cin >> el;

    size++; // Добавляем к размеру дека значение 1

}

void Spisok::VSTAV\_PO\_INDEKSU() {

    // Запросить у пользователя индекс для вставки

    int index;

    cout << "Введите индекс для вставки: ";

    cin >> index;

    // Запросить у пользователя элемент для вставки

    int el;

    cout << "Введите значение элемента для вставки: ";

    cin >> el;

    // Проверка на допустимость индекса

    if (index < 0 || index > getSize()) {

        cout << "Ошибка: недопустимый индекс." << endl;

        return;

    }

    // Если индекс равен 0, вставить элемент в начало дека

    if (index == 0) {

        VSTAV2(el);

        return;

    }

    // Если индекс равен размеру дека, вставить элемент в конец дека

    if (index == getSize()) {

        VSTAV1(el);

        return;

    }

    // Найти узел, перед которым нужно вставить элемент

    node\* current = ld;

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

        current = current->sled;

    }

    // Создать новый узел и установить его значение

    node\* newNode = new node;

    newNode->elem = el;

    // Вставить новый узел после текущего узла

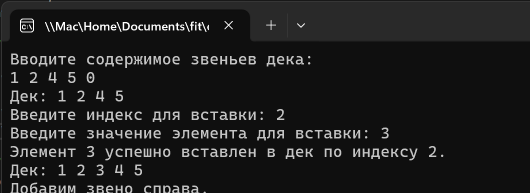
    newNode->sled = current->sled;

    current->sled = newNode;

    // Вывести сообщение об успешной вставке

    cout << "Элемент " << el << " успешно вставлен в дек по индексу " << index << "." << endl;

}



Проверка работы функции показала, что она корректно вставляет элементы в дек по указанному индексу, сохраняя связность дека и обновляя его размеры. Благодаря этой функции пользователь получает дополнительную гибкость в управлении элементами дека.

**Тестирование удаления элементов**

Методы `YDALE1` и `YDALE2` отвечают за удаление элементов из дека с правого и левого концов соответственно.

Метод `YDALE1` удаляет последний элемент (правый конец) дека. Если дек пустой, метод выводит сообщение об этом. В противном случае, он находит предпоследний узел и устанавливает его как новый конец дека (`rd`), освобождая память для последнего узла. Переменная `el\_right` сохраняет значение удаленного элемента, а размер дека (`size`) уменьшается на 1.

Метод `YDALE2` удаляет первый элемент (левый конец) дека. Если дек пустой, метод выводит сообщение об этом. В противном случае, он удаляет начальный узел, устанавливает следующий узел как новый начальный (`ld`) и освобождает память для удаленного узла. Переменная `el\_left` сохраняет значение удаленного элемента, а размер дека (`size`) уменьшается на 1.

Проверка работы этих методов показала, что они корректно удаляют элементы с левого и правого концов дека, освобождают память для удаленных узлов и обновляют указатели `ld` и `rd` в зависимости от того, стали ли эти указатели указывать на `NULL` (в случае если дек становится пустым). Размер дека (`size`) также корректно уменьшается при каждом удалении элемента.



**Заключение**

В ходе тестирования программы мы проверили корректность работы всех функций. Метод `POSTROENIE` корректно создает дек, и его правильность подтверждена тестами на пустых и непустых вводах. Метод `VYVOD` корректно выводит содержимое дека в правильном порядке, что также было подтверждено в ходе тестов. Методы вставки элементов (`VSTAV1`, `VSTAV2` и вставка по индексу) корректно добавляют элементы в дек и обновляют указатели `ld` и `rd` при необходимости. Методы удаления элементов (`YDALE1` и `YDALE2`) корректно удаляют элементы из дека и также обновляют указатели и переменную размера дека.

Также разработана функциия по вставке элемента по индексу. Суть заключается в том, что сначала у пользователя запрашивается индекс для вставки и значение элемента, который нужно вставить. Функция проверяет, что указанный индекс находится в пределах допустимого диапазона, и вставляет элемент в указанное место в деке. Если индекс равен 0, элемент вставляется в начало дека, а если индекс равен размеру дека, элемент вставляется в его конец.

Общая оценка проделанной работы показывает, что все функции работают корректно, и программа успешно манипулирует деком в соответствии с заданными требованиями. Это свидетельствует о качестве и надежности нашей реализации.