**

Колледж космического машиностроения и технологии

КУРСОВОЙ ПРОЕКТ

По дисциплине «Прикладное программирование»

Тема: **«Дек»**

Выполнил студент

Фомин Ярослав Александрович

Группа П2-17

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_(Дата сдачи работы)

Проверил преподаватель:

Гусятинер Леонид Борисович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(оценка)

Королев, 2020

**Содержание**

[**Введение** 3](#_Toc44142316)

[**1.** **Теоретическая часть** 4](#_Toc44142317)

[**1.1** **Описание и определение Дек** 4](#_Toc44142318)

[**1.2** **Стек и Очередь** 5](#_Toc44142319)

[**1.3** **Краткое введение. Итераторы и контейнеры в C++.** 7](#_Toc44142320)

[**1.4** **Абстрактный тип данных Дек в языке программирования C++** 8](#_Toc44142321)

[**2.** **Проектная часть** 10](#_Toc44142322)

[**2.1** **Основные требования к программе** 10](#_Toc44142323)

[**2.2** **Выбор инструмента разработки программы** 10](#_Toc44142324)

[**Заключение** 12](#_Toc44142325)

[**Список используемой литературы** 13](#_Toc44142326)

[**Приложение** 14](#_Toc44142327)

# **Введение**

Данный курсовой проект посвящен созданию программ для работы с очередью на языке C++. Цель курсового проекта заключалась в понимании основных принципов работы с Деком.

1. **Теоретическая часть**
   1. **Описание и определение Дек**

**Дек (Двусвязная очередь)** — абстрактный тип данных, в котором элементы можно добавлять или удалять как с начала, так и с конца очереди. Может быть реализована при помощи двусвязного списка. Двусвязная очередь, реализованная таким способом, имеет два существенных недостатка: ограниченный размер и линейное время

**Дек**— симбиоз стека и очереди, то есть дисциплинами обслуживания являются одновременно LIFO и FIFO. Обычно для реализации дека используют двусвязный линейный список, то есть на основе готовых функций списка создают новые для дека.

Типовые операции Дек**:**

* Deque() создаёт новый пустой дек. Не нуждается в параметрах и возвращает пустой дек.
* addFront(item) добавляет новый элемент в голову дека. Параметр (элемент) необходим; ничего не возвращает.
* addRear(item) добавляет новый элемент в хвост дека. Параметр (элемент) необходим; ничего не возвращает.
* removeFront() удаляет первый элемент из дека. Не нуждается в параметрах и возвращает элемент. Дек модифицируется.
* removeRear() удаляет последний элемент из дека. Не нуждается в параметрах и возвращает элемент. Дек модифицируется.
* isEmpty() проверяет дек на пустоту. Не нуждается в параметрах и возвращает булево значение.
* size() возвращает количество элементов в деке. Не нуждается в параметрах и возвращает целое число.
  1. **Стек и Очередь**

**Стеком** называется упорядоченный набор элементов, в котором размещение новых и удаление существующих происходит с одного конца, называемого вершиной. Принцип работы стека сравнивают со стопкой листов бумаги: чтобы взять второй сверху, нужно снять верхний. В стеке реализуется дисциплина обслуживания LIFO:

1. **LAST**— последний
2. **INPUT** — вошел
3. **FIRST** — первый
4. **OUTPUT** — вышел

Различают аппаратный и программный стек.  
  
**Аппаратный** стек используется для хранения адресов возврата из функций и их аргументов.  
**Программный стек** – это пользовательская модель (структура) данных.

**Операции для работы со стеком**

Над стеком реализованы следующие операции:

* инициализация стека init(s), где s — стек
* помещение элемента в стек push(s, i), где s — стек, i — помещаемый элемент;
* удаление элемента из стека i=pop(s);
* получение верхнего элемента стека без его удаления i=stkTop(s), где s — стек
* получение количества элементов стека
* определение, пуст ли стек isempty(s) возвращает 1 если стек пустой и 0 в противном случае.
* вывод элементов стека stkPrint(s), где s — стек

**Способы реализации стека**

Существует несколько способов реализации стека:

* с помощью одномерного массива;
* с помощью связанного списка;
* с помощью класса объектно-ориентированного программирования.

**Очередью** называется упорядоченный набор элементов, которые могут удаляться с её начала и помещаться в её конец.

Очередь в программировании используется, как и в реальной жизни, когда нужно совершить какие-то действия в порядке их поступления, выполнив их последовательно. Примером может служить организация событий в Windows. Когда пользователь оказывает какое-то действие на приложение, то в приложении не вызывается соответствующая процедура (ведь в этот момент приложение может совершать другие действия), а ему присылается сообщение, содержащее информацию о совершенном действии, это сообщение ставится в очередь, и только когда будут обработаны сообщения, пришедшие ранее, приложение выполнит необходимое действие.  
  
Очередь организована, в отличие от стека, согласно дисциплине обслуживания FIFO:

1. **FIRST** — первый
2. **INPUT** — вошел
3. **FIRST** — первый
4. **OUTPUT** — вышел  
     
   Существует несколько способов реализации очереди:

* с помощью одномерного массива;
* с помощью связанного списка;
* с помощью класса объектно-ориентированного программирования.

Простейшие операции с очередью:

* init() инициализация очереди.
* insert (q, x) — помещение элемента x в конец очереди q (q — указатель на очередь);
* x=remove (q) — удаление элемента x из очереди q;
* isempty(q) — возвращает 1, если очередь пуста и 0 в противном случае;
* print(q) – вывод элементов очереди q.

# **Краткое введение.** [**Итераторы и контейнеры в C++**](https://codelessons.ru/cplusplus/iteratory-v-c-vvedenie.html)**.**

**Итератор** — это такая структура данных, которая используется для обращения к определенному элементу в контейнерах STL. Обычно из используют с контейнерами set, list , а у вектора для этого применяют индексы.

Кстати по мере того, как мы будем изучать итераторы, вам все больше будет казаться, что итераторы и есть [указатели](https://codelessons.ru/cplusplus/ukazateli-v-c-podrobnoe-rukovodstvo.html) (Но не стоит путать).

Как создать итератор:

* Для создания итератора мы должны с самого начала программы подключить библиотеку <iterator>.

#inlcude <iterator>

* Далее для его создании нам потребуется использовать вот эту схему:

<контейнер> <его тип> :: iterator <имя итератора>;

* <контейнер> — указываем требуемый контейнер, на который и будет ссылаться итератор. Например deque, map, vector, list.
* <его тип> — указываем тип контейнера.

Вам нужно помнить! Если вы создали итератор и случайно ввели не тот тип данных, который указали при создании контейнера, то ваша программа будет работать неправильно и вообще может не запуститься.

Методы начала и конца контейнеров:

У каждого контейнера имеются два метода, которые, как указатели передают итератору начало или конец контейнера — begin() и end().

* Метод begin() отправит итератор на начала контейнера.
* А метод end() отправит на конец. А если точнее, то на одну ячейку больше последней. Если мы попытаемся вывести эту ячейку у нас появятся проблемы с компилятором 🙂 .

Их мы можем использовать даже без подключения библиотеки <iterator>, что очень удобно.

* 1. **Абстрактный тип данных Дек в языке программирования C++**

Стандартная библиотека C++ предоставляет специальные средства работы с двусторонней очередью. Для этого в ней предусмотрен контейнер "deque". Он позволяет осуществлять вставку и удаление элементов. Методы контейнера deque в C++:

* "front" – возврат значения первого элемента;
* "back" – возврат значения последнего элемента;
* "push\_front" – добавление элемента в начало;
* "push\_back" – добавление элемента в конец;
* "pop\_front" – удаление первого элемента;
* "pop\_back" – удаление последнего элемента;
* "size" – возврат числа элементов дека;
* "clear" – очистка дека.

Пример собственного кода реализации по 1.3 – 1.4 пункту , с использованием Дека и Итератора:

#include <iostream>

#include <deque> // подключаем заголовочный файл деков

#include <iterator> // заголовок итераторов

#include <locale.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

int dequeSize = 0;

cout << "Введите размер дека: ";

cin >> dequeSize; //Ввели

deque <char> myDeque(dequeSize); // Cоздаем дек и резервируем для него память

cout << "Введите элементы дека: " // Заполняем дек с клавиатуры

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

{

cin >> myDeque[i];

}

cout << "\nВведенный дек: ";

if (!myDeque.empty()) // Если дек не пуст

{

copy( myDeque.begin(), myDeque.end(), ostream\_iterator<char>(cout," ") ); // вывод на экран элементов дека

}

return 0;

}

1. **Проектная часть**
   1. **Основные требования к программе**

Программа должна корректно работать, обладать быстродействием, исключать ошибочное распознания.

Стадии и этапы разработки:

1. Разработка технического задания.
2. Разработка функций.
3. Разработка интерфейса.
4. Написание кода, исправление ошибок, тесты.
   1. **Выбор инструмента разработки программы**

При написании курсовой работы была использована: Интегрированая среда разработки Code::Blocks и язык программирования С++.

Code::Blocks — это бесплатная кроссплатформенная среда разработки на языке C/C++. На данный момент это лучшая бесплатная среда разработки на языке Си.

Code::Blocks разрабатывается для Windows, Linux и Mac OS X.

В среде Windows скачать эту среду удобнее всего в составе сборки Си-экспресс, в которой уже есть все необходимые библиотеки для начала работы. Сборка не требует установки и работает по принципу: «Распаковал и работай».

**C++** - компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщенное программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. Именно поэтому мой выбор остановился на нём.

**Достоинства языка C++**

Масштабируемость. На языке C++ разрабатывают программы для самых различных платформ и систем.Возможность работы на низком уровне с памятью, адресами, портами.Что, при неосторожном использовании, может легко превратиться в недостаток.Возможность создания обобщенных алгоритмов для разных типов данных, их специализация, и вычисления на этапе компиляции, используя шаблоны.

### **Заключение**

В заключение хочется сказать, что разработка данной программы принесла большое количество, как положительных, так и отрицательных эмоций. Достаточно хорошо укрепились знания работы с Деком, Итератором, Массивом.

### **Список используемой литературы**

1. https://tproger.ru/translations/stacks-and-queues-for-beginners/
2. https://cjblogr.blogspot.com/2015/02/blog-post.html
3. https://ru.wikipedia.org/wiki/Code::Blocks
4. https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D1%83%D1%85%D1%81%D1%82%D0%BE%D1%80%D0%BE%D0%BD%D0%BD%D1%8F%D1%8F\_%D0%BE%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C
5. https://prog-cpp.ru/category/struct-posts/
6. https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9\_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA

### **Приложение**

#include <iostream>

#include <iterator>

using namespace std;

const int N=5; //размер дека

struct Deque

{

int data[N]; //массив данных

int last; //указатель на конец

};

void Creation(Deque \*D) //создание дека

{

D->last=0;

}

bool Full(Deque \*D) //проверка дека на пустоту

{

if (D->last==0) return true;

else return false;

}

void AddL(Deque \*D) //добавление элемента в начало

{

if (D->last==N)

{

cout<<"\nДек заполнен\n\n";

return;

}

int value;

cout<<"\nЗначение > ";

cin>>value;

for (int i=D->last; i>0; i--)

D->data[i]=D->data[i-1];

D->data[0]=value;

D->last++;

cout<<endl<<"Элемент добавлен\n\n";

}

void AddR(Deque \*D) //добавление элемента в конец

{

if (D->last==N)

{

cout<<"\nДек заполнен\n\n";

return;

}

int value;

cout<<"\nЗначение > ";

cin>>value;

D->data[D->last++]=value;

cout<<endl<<"Элемент добавлен\n\n";

}

void DeleteL(Deque \*D) //удаление первого элемента

{

for (int i=0; i<D->last; i++) //смещение элементов

D->data[i]=D->data[i+1];

D->last--;

}

void DeleteR(Deque \*D) //удаление последнего элемента

{

D->last--;

}

int OutputL(Deque \*D) //вывод первого элемента

{

return D->data[0];

}

int OutputR(Deque \*D) //вывод последнего элемента

{

return D->data[D->last-1];

}

int Size(Deque \*D) //размер дека

{

return D->last;

}

int main() //главная функция

{

setlocale(LC\_ALL,"Rus");

Deque D;

Creation(&D);

char number;

do

{ cout<<" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ "<<endl;

cout<<" / \\"<<endl;

cout<<"|| Небольшое меню ||"<<endl;

cout<<"||1. Добавить элемент в начало ||"<<endl;

cout<<"||2. Добавить элемент в конец ||"<<endl;

cout<<"||3. Удалить первый элемент ||"<<endl;

cout<<"||4. Удалить последний элемент ||"<<endl;

cout<<"||5. Вывести первый элемент ||"<<endl;

cout<<"||6. Вывести последний элемент ||"<<endl;

cout<<"||7. Узнать размер дека ||"<<endl;

cout<<"||0. Выйти ||\n \\ /\n";

cout<<" Введите номер команды >> ";

cin>>number;

switch (number)

{

case '1':

AddL(&D);

break;

case '2':

AddR(&D);

break;

case '3':

if (Full(&D)) cout<<endl<<"Дек пуст\n\n";

else

{

DeleteL(&D);

cout<<endl<<"Элемент удален из дека\n\n";

}

break;

case '4':

if (Full(&D)) cout<<endl<<"Дек пуст\n\n";

else

{

DeleteR(&D);

cout<<endl<<"Элемент удален\n\n";

}

break;

case '5':

if (Full(&D)) cout<<endl<<"Дек пуст\n\n";

else cout<<"\nПервый элемент: "<<OutputL(&D)<<"\n\n";

break;

case '6':

if (Full(&D)) cout<<endl<<"Дек пуст\n\n";

else cout<<"\nПоследний элемент: "<<OutputR(&D)<<"\n\n";

break;

case '7':

if (Full(&D)) cout<<endl<<"Дек пуст\n\n";

else cout<<"\nРазмер дека: "<<Size(&D)<<"\n\n";

break;

case '0':

break;

default:

cout<<endl<<"Команда не определена\n\n";

break;

}

}

while(number!='0');

system("pause");

}