**

Колледж космического машиностроения и технологии

КУРСОВОЙ ПРОЕКТ

По дисциплине «Прикладное программирование»

Тема: «**Очередь»**

Выполнил студент

Фомин Даниил Романович

Группа П2-17

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

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

Проверил преподаватель:

Гусятинер Леонид Борисович

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

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

Королев, 2020

Оглавление

[**Введение** 3](#_Toc44138829)

[**1. Теоретическая часть** 4](#_Toc44138830)

[**1.1 Язык программирования C++** 4](#_Toc44138831)

[**1.2 Плюсы и минусы языка программирования C++** 5](#_Toc44138832)

[**1.3 Описание и определение Очереди** 6](#_Toc44138833)

[**1.4 Очередь в C++** 7](#_Toc44138834)

[**1.5 Язык программирования Python** 13](#_Toc44138835)

[**1.6 Плюсы и минусы Python** 13](#_Toc44138836)

[**1.7 Очередь в Python** 15](#_Toc44138837)

[**2. Проектная часть** 19](#_Toc44138838)

[**2.1.Основные требования к программе.** 19](#_Toc44138839)

[**2.2.Описание инструмента разработки программы** 20](#_Toc44138840)

[**3. Приложения** 21](#_Toc44138841)

[**3.1 Листинг ( C++)** 21](#_Toc44138842)

[**3.2 Листинг ( Python )** 23](#_Toc44138843)

[**Заключение** 25](#_Toc44138844)

[**Список литературы** 26](#_Toc44138845)

# **Введение**

Данный курсовой проект посвящен созданию программы для работы с очередью на языке C++.

Цель курсового проекта заключалась в понимании основных принципов работы с очередью

# **1. Теоретическая часть**

**1.1 Язык программирования C++**

**C++** (читается си-плюс-плюс) - 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. В сравнении с его предшественником - языком **C**, - наибольшее внимание уделено поддержке объектно-ориентированного и обобщенного программирования.

**C++** широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также игр. Существует множество реализаций языка **C++**, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это **GCC**, **Visual C++**, **Intel C++ Compiler**, **Embarcadero (Borland) C++ Builder** и другие. **C++** оказал огромное влияние на другие языки программирования, в первую очередь на **Java** и **C#**.

Синтаксис **C++** унаследован от языка **C**. Одним из принципов разработки было сохранение совместимости с **C**. Тем не менее, **C++** не является в строгом смысле надмножеством **C**; множество программ, которые могут одинаково успешно транслироваться как компиляторами **C**, так и компиляторами **C++**, довольно велико, но не включает все возможные программы на **C**.

# 

# **1.2 Плюсы и минусы языка программирования C++**

**Плюсы языка C++**

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

**Минусы языка C++**  
  
1) Наличие множества возможностей, нарушающих принципы типобезопасности приводит к тому, что в С++-программы может легко закрасться трудноуловимая ошибка. Вместо контроля со стороны компилятора разработчики вынуждены придерживаться весьма нетривиальных правил кодирования. По сути эти правила ограничивают С++ рамками некоего более безопасного подъязыка. Большинство проблем типобезопасности С++ унаследовано от С, но важную роль в этом вопросе играет и отказ автора языка от идеи использовать автоматическое управление памятью (например, сборку мусора). Так визитной карточкой С++ стали уязвимости типа "переполнение буфера".  
2) Плохая поддержка модульности. Подключение интерфейса внешнего модуля через препроцессорную вставку заголовочного файла ([#include](http://vk.com/feed?section=search&q=%23include)) серьёзно замедляет компиляцию, при подключении большого количества модулей. Для устранения этого недостатка, многие компиляторы реализуют механизм прекомпиляции заголовочных файлов Precompiled Headers.  
3) Недостаток информации о типах данных во время компиляции (CTTI).  
4) Язык C++ является сложным для изучения и для компиляции.  
5) Некоторые преобразования типов неинтуитивны. В частности, операция над беззнаковым и знаковым числами выдаёт беззнаковый результат.  
6) Препроцессор С++ (унаследованный от С) очень примитивен. Это приводит с одной стороны к тому, что с его помощью нельзя (или тяжело) осуществлять некоторые задачи метапрограммирования, а с другой, в следствии своей примитивности, он часто приводит к ошибкам и требует много действий по обходу потенциальных проблем. Некоторые языки программирования (например, Scheme и Nemerle) имеют намного более мощные и более безопасные системы метапрограммирования (также называемые макросами, но мало напоминающие макросы С/С++).  
7) С конца XX века в сообществе С++ получило распространение так называемое метапрограммирование на базе шаблонов. По сути, оно использует особенности шаблонов C++ в целях реализации на их базе интерпретатора примитивного функционального языка программирования выполняющегося во время компиляции. Сама по себе данная возможность весьма привлекательна, но, в следствии вышесказанного, такой код весьма трудно воспринимать и отлаживать. Языки Lisp/Scheme, Nemerle и некоторые другие имеют более мощные и одновременно более простые для восприятия подсистемы метапрограммирования. Кроме того, в языке D реализована сравнимая по мощности, но значительно более простая в применении подсистема шаблонного метапрограммирования.  
8)Хотя декларируется, что С++ мультипарадигменный язык, реально в языке отсутствует поддержка функционального программирования. Отчасти, данный пробел устраняется различными библиотеками (Loki, Boost) использующими средства метапрограммирования для расширения языка функциональными конструкциями (например, поддержкой лямбд/анонимных методов), но качество подобных решений значительно уступает качеству встроенных в функциональные языки решений. Такие возможности функциональных языков как сопоставление с образцом вообще крайне сложно эмулировать средствами метапрограммирования.

# **1.3 Описание и определение Очереди**

**Очередь** — это структура данных, которая построена по принципу LILO (last in — last out: последним пришел — последним вышел). В C++ уже есть готовый STL контейнер — queue.

**Применение очередей**

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.

 В очереди, если вы добавите элемент, который вошел самый первый, то он выйдет тоже самым первым. Получается, если вы добавите 4 элемента, то первый добавленный элемент выйдет первым.

**1.4 Очередь в C++**

**Объявления очереди**

queue <тип данных> <имя>;

* Сначала нам нужно написать слова queue.
* Дальше в <тип данных> мы должны указать тот тип, которым будем заполнять нашу очередь.
* И в конце нам остается только указать название очереди.

Пример правильного объявления

queue <int> q;

**Методы очереди**

**Метод** — это та же самая функция, но она работает только с контейнерами STL. Для работы с очередью вам понадобится

знать функции: **push(), pop(), front(), back(), empty().**

1. Для добавления в очередь нового элемента нужно воспользоваться функцией — **push()**. В круглых скобках должно находится значение, которое мы хотим добавить.
2. Если нам понадобилось удалить первый элемент нужно оперировать функцией **pop()**. В круглых скобках уже не чего не нужно указывать, но по правилам они в обязательном порядке должны присутствовать!
3. Если вам понадобилось обратиться к первому элементу очереди, то вам понадобится функция **front()**.
4. Чтобы обратиться к последнему элементу в очереди вам поможет функция **back()**.
5. Чтобы узнать пуста ли очередь нужно воспользоваться функцией **empty()**.

Если ваша очередь пуста — возвратит **true**.

Если же в ней что-то есть — возвратит **false**.

**Пример реализации методов:**

#include <iostream>

#include <queue>  // подключили библиотеку queue

using namespace std;

int main() {

  setlocale(LC\_ALL,"rus");

  queue <int> q;  // создали очередь q

  cout << "Пользователь, пожалуйста введите 7 чисел: " << endl;

  for (int h = 0; h < 7; h++) {

    int a;

    cin >> a;

    q.push(a);  // добавляем в очередь элементы

  }

  cout << endl;

  cout << "Самый первый элемент в очереди: " << q.front() << endl;  // выводим первый

                                                                   // элемент очереди

  q.pop();  // удаляем элемент из очереди

  cout << "Новый первый элемент (после удаления): " << q.front() << endl;

  if (!q.empty()) cout << "Очередь не пуста!";  // проверяем пуста ли очередь (нет)

  system("pause");

  return 0;

}

**Создание очереди с помощью массива**

Для реализации нам понадобится создать две дополнительные переменные **start** и **ends**. start будет указывать на первый элемент очереди, a ends на последний элемент.

Чтобы обратится к последнему элементу нам придется использовать эту конструкцию — queue[ends]. Обращение к первому элементу будет выглядеть аналогично queue[start].

Если понадобится удалить элемент из очереди, то придется всего лишь уменьшить переменную start на один.

1. Если результатом условия будет **true**, то очередь пуста.
2. Если же результат условия **false**, значит очередь чем-то заполнен

**Пример создания очереди с помощью массива:**

setlocale(LC\_ALL,"rus");

int q[7];  // создали массив q

int start = 0, ends = 0;  // создали переменные начала и конца очереди

cout << "Пользователь, пожалуйста введите 7 чисел: " << endl;

 for (int h = 0; h < 7; h++) { int a; cin >> a;

   int a;

    cin >> a;

    q[start++] = a;  // добавляем элементы в очередь(массив)

  }

  cout << "Самый первый элемент в очереди: " << q[start - 1] << endl;

  start--;  // удаляем первый элемент(уменьшаем start на 1)

  cout << "Новый первый элемент (после удаления): " << q[start - 1] << endl;

  cout << "Самый последний элемент в очереди: " << q[ends];

// выводим последний элемент очереди

                                                           if (start != ends) cout << "Очередь не пуста!";  // проверяем пуста ли очередь

**Очередь с приоритетом**

**Очередь с приоритетом**(**priority\_queue**) — это обычная очередь, но в ней новый элемент добавляется в то место, чтобы очередь была отсортирована по убыванию.

Так самый большой элемент в приоритетной очереди будет стоять на первом месте.

|  |
| --- |
| priority\_queue <тип данных> <имя>; |

1. В начале нужно написать priority\_queue.
2. Потом в скобках указать тип данных, который будет находится в очереди.

3. И конечно же в самом конце мы должны дать ей имя.

Для добавления элемента в очередь с приоритетом мы должны использовать функцию **push()**. Но чтобы обратится к первому элементу должны использоваться именно функция — **top()**.

**Пример создания очереди с приоритетом:**

setlocale(LC\_ALL,"rus");

  priority\_queue <int> priority\_q;  // объявляем приоритетную очередь

  cout << "Введите 7 чисел: " << endl;

  for (int j = 0; j < 7; j++) { int a; cin >> a;

    priority\_q.push(a);  // добавляем элементы в очередь

  }

  cout << "Первый элемент очереди: " << priority\_q.top();  // выводим первый элемент

**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.

Рассмотрим реализацию очереди на базе массива. Используем массив и две переменные:

* frnt – позиция первого элемента в очереди;
* rear – позиция последнего элемента в очереди

#### Класс Queue

Класс Queue, будет реализован с помощью связного списка. Он будет предоставлять методы Enqueue для добавления элемента, Dequeue для удаления, Peek и Count. Как и класс Stack, он не будет реализовывать интерфейс ICollection<T>, поскольку это коллекции специального назначения.

public class Queue

{

LinkedList \_items = new LinkedList();

public void Enqueue(T value)

{

throw new NotImplementedException();

}

public T Dequeue()

{

throw new NotImplementedException();

}

public T Peek()

{

throw new NotImplementedException();

}

public int Count

{

get;

}

}

**1.5 Язык программирования Python**

**Python** (в русском языке распространено название пито́н) — [высокоуровневый язык программирования](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. [Синтаксис](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) ядра Python минималистичен. В то же время [стандартная библиотека](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_Python) включает большой объём полезных функций.

Python поддерживает [структурное](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [обобщенное](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [объектно-ориентированное](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [функциональное](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [аспектно-ориентированное программирование](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Основные архитектурные черты — [динамическая типизация](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [автоматическое управление памятью](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), полная [интроспекция](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D1%80%D0%BE%D1%81%D0%BF%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), механизм [обработки исключений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9), поддержка [многопоточных вычислений](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C), высокоуровневые [структуры данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Поддерживается разбиение программ на [модули](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), которые, в свою очередь, могут объединяться в пакеты.

Эталонной реализацией Python является интерпретатор [CPython](https://ru.wikipedia.org/wiki/CPython), поддерживающий большинство активно используемых платформ. Он распространяется под [свободной лицензией](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая [проприетарные](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BF%D1%80%D0%B8%D0%B5%D1%82%D0%B0%D1%80%D0%BD%D0%BE%D0%B5_%D0%9F%D0%9E). Есть [реализация интерпретатора для JVM](https://ru.wikipedia.org/wiki/Jython) с возможностью [компиляции](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80), [CLR](https://ru.wikipedia.org/wiki/IronPython), [LLVM](https://ru.wikipedia.org/wiki/LLVM), другие независимые реализации. Проект [PyPy](https://ru.wikipedia.org/wiki/PyPy) использует [JIT-компиляцию](https://ru.wikipedia.org/wiki/JIT-%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F), которая значительно увеличивает скорость выполнения Python-программ.

**1.6 Плюсы и минусы Python**

**Плюсы:**

* **Низкий порог вхождения**. Синтаксис Python более понятный для новичка.
* **Логичный, лаконичный и понятный**. В сравнении с многими другими языками Python имеет легкочитаемый синтаксис, разве что Visual Basic тоже легок.
* **Кроссплатформенный**: подходит для разных платформ: и Linux, и Windows.
* Есть реализация интерпретаторов для мобильных устройств и непопулярных систем.
* **Широкое применение**. Используется для разработки веб-приложений, игр, удобен для автоматизации, математических вычислений, машинного обучения, в области интернета вещей. Существует реализация под названием Micro Python, оптимизированная для запуска на микроконтроллерах (можно писать инструкции, логику взаимодействия устройств, организовывать связь, реализовывать умный дом).
* **Сильное комьюнити** и много конференций. К примеру, недавно в Одессе состоялся PyCon. На конференции в числе всех спикеров выступили 4 иностранных докладчика, которые затронули интересные темы. Лично мне было полезно услышать соавтора библиотек MindsDB, докладчика из Нью-Йорка (рассказывал про искусственный интеллект при разработке средств распознавания лиц на фото), спикера из британской компании Elastic (выступил с докладом на тему «инструменты мониторинга производительности вашего приложения»), спикера из Минска (девушка рассказывала про протокол MQTT для связи устройств интернета вещей).
* Мощная **поддержка компаний-гигантов IT-индустрии**. Такие компании, как Google, Facebook, Dropbox, Spotify, Quora, Netflix, на определенных этапах разработки использовали именно Python.
* **Высокая востребованность** на рынке труда.
* В мире Python **много качественных библиотек**, так что не нужно изобретать велосипед, если надо срочно решить какую-то коммерческую задачу. Для обучения есть много толковых книг, в первую очередь на английском языке, конечно, но и в переводе также издана достойная литература. Сегодня много обучающих материалов на Youtube: видео блоги, записи вебинаров и конференций. Думаю, что сейчас учиться легче, чем в то время, когда я начинал изучение.
* Python отличается строгим требованием к написанию кода (**требует отступы**), что является преимуществом, по моим наблюдениям. Изначально язык способствует писать код организованно и красиво.

И если говорить о **минусах**, то Python — язык с динамической типизацией. С одной стороны код проще и быстрее писать, но производительность уступает таким компилируемым языкам, как C++ и Golang.

Но для большинства задач: для веб-разработки, для скриптов, прототипирования, машинного обучения и работы с большими данными, — один из лучших языков. Я сталкивался также с Ruby, и, если сравнивать с Python, то популярность за последним. Когда-то технологии были на одном уровне в веб-разработке».

**1.7 Очередь в Python**

Для реализации АТД очереди подходящим решением вновь станет создание нового класса. Как и раньше, для построения внутреннего представления очереди мы будем использовать мощь и простоту коллекции “список”.

Нам надо определиться, какой конец списках считать головой, а какой - хвостом. Реализация, показанная в [листинге 1](https://aliev.me/runestone/BasicDS/ImplementingaQueueinPython.html#lst-queuecode) предполагает, что последний элемент очереди находится на его нулевой позиции. Это позволяет использовать функцию `insert` для добавления новых элементов в конец очереди. Операция `pop` будет использоваться для удаления переднего элемента (последнего элемента в списке). Напомним, что также это означает постановку в очередь за O(n), а извлечение - за O(1) времени.

Абстрактный тип данных для очереди определяется следующими структурой и операциями. Как говорилось выше, очередь структурирована как упорядоченная коллекция элементов, которые добавляются с одного конца (“хвоста”), а удаляются с другого (“головы”). Очередь поддерживает свойство упорядочения FIFO. Операции для очереди представлены ниже.

1. Queue() создаёт новую пустую очередь. Не нуждается в параметрах, возвращает пустую очередь.
2. enqueue(item) добавляет новый элемент в конец очереди. Требует элемент в качестве параметра, ничего не возвращает.
3. dequeue() удаляет из очереди передний элемент. Не нуждается в параметрах, возвращает элемент. Очередь изменяется.
4. isEmpty() проверяет очередь на пустоту. Не нуждается в параметрах, возвращает булево значение.
5. size() возвращает количество элементов в очереди (целое число). Не нуждается в параметрах.

**Встроенный список**

Мы можем использовать [обычный список](https://python-scripts.com/lists-tuples-dictionaries) в качестве очереди, но это не очень эффективно с точки зрения [производительности](https://python-scripts.com/cprofile-code-profiling). Списки слишком медленные для этой задачи, так как вставка и удаление элемента с начала требует сдвига всех прочих элементов по одному, на это уходит О(n) времени.

**Как использовать список Python в качестве очереди FIFO:**

q = []

q.append('eat')

q.append('sleep')

q.append('code')

print(q)

# ['eat', 'sleep', 'code']

print(q.pop(0)) # 'eat'

**Класс collections.deque**

Класс **deque** реализует двухконечную очередь, которая поддерживает добавление и удаление элементов с обоих концов в течение О(1) времени. Объекты **deque** представлены в виде двусвязных списков, что дает им превосходную [производительность](https://python-scripts.com/cprofile-code-profiling) для входящих и выходящих элементов, но при этом у него **плохая производительность** O(n) при работе со случайно принимаемыми элементами в середине очереди. В связи с тем, что **deque** поддерживает вставку и удаление элементов одинаково хорошо, они могут поддерживать и очереди и стеки. [collections.deque](https://python-scripts.com/import-collections) это отличное решение, если вы ищите структуру данных очереди в Python в **стандартной библиотеке**.

**Как использовать collections.deque в качестве очереди FIFO:**

from collections import deque

q = deque()

q.append('eat')

q.append('sleep')

q.append('code')

print(q)

# deque(['eat', 'sleep', 'code'])

print(q.popleft()) # 'eat'

print(q.popleft()) # 'sleep'

print(q.popleft()) # 'code'

print(q.popleft())

IndexError: "pop from an empty deque"

**Класс queue.Queue**

Такая реализация очереди в стандартной библиотеке является **синхронизированной** и предоставляет блокировку семантики для поддержки нескольких конкурирующих производителей и потребителей. [Модуль queue](https://python-scripts.com/queues) содержит несколько других классов, реализующих мульти-продюсера, очереди мульти-потребители , которые полезны в параллельных вычислениях. В зависимости от причины использования, закрытие семантики может быть полезным, или просто привести к ненужному перерасходу. В данном случае, вам лучше будет использовать **collections.deque** как очередь общего назначения.

**Как использовать queue.Queue в качестве очереди FIFO:**

from queue import Queue

q = Queue()

q.put('eat')

q.put('sleep')

q.put('code')

print(q)

# <queue.Queue object at 0x1070f5b38>

print(q.get()) # 'eat'

print(q.get()) # 'sleep'

print(q.get()) # 'code'

print(q.get\_nowait()) # queue.Empty

q.get()

**Класс multiprocessing.Queue**

Это реализация очереди с разделенными функциями, которые позволяет находящимся в очереди объектам быть обработанными параллельно несколькими одновременно работающими процессами. Основанная на процессах паралеллизация очень популярна в Python, из-за [глобального блокиратора интерпретатора GIL](https://wiki.python.org/moin/GlobalInterpreterLock). **multiprocessing.Queue** используется при разделении данных между процессами и может хранить любой пригодный объект.

**Как использовать multiprocessing.Queue в качестве очереди FIFO:**

from multiprocessing import Queue

q = Queue()

q.put('eat')

q.put('sleep')

q.put('code')

print(q)

# <multiprocessing.queues.Queue object at 0x1081c12b0>

print(q.get()) # 'eat'

print(q.get()) # 'sleep'

print(q.get()) # 'code'

q.get()

**2. 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. Написание кода, исправление ошибок, тесты.

* 1. **Описание инструмента разработки программы**

При написании курсовой работы был использован язык программирования C++ и среда разработки Code::Blocks .

[**Code::Blocks**](http://www.codeblocks.org/) - это кроссплатформенная, бесплатная, Open Source [среда разработки](http://en.wikipedia.org/wiki/Integrated_development_environment). В ней есть многие приятные вещи, которые современная среда разработки обязана уметь. Class browser, code completion.

Есть импорт воркспейсов Microsoft Visual Studio и проектов [Dev-C++](http://www.bloodshed.net/devcpp.html).

### 3. Приложения

### 3.1 Листинг ( C++)

**#include <iostream>**

**using namespace std;**

**class Queue {**

**static const int SIZE=100;**

**int \*queue;**

**int frnt, rear;**

**public :**

**Queue () ;**

**void push ( int num ) ;**

**void out();**

**int size();**

**void pop();**

**int front();**

**int back();**

**} ;**

**//Конструктор**

**Queue::Queue() {**

**queue = new int[SIZE];**

**frnt = rear = 0 ;**

**}**

**//Вывод элементов очереди**

**void Queue::out() {**

**for(int i=frnt+1;i<rear+1;i++)**

**cout<<" "<<queue[i];**

**}**

**//Помещение элемента в очередь**

**void Queue::push ( int num ) {**

**if ( rear+1 == frnt || ( rear + 1 ==SIZE && !frnt )) {**

**cout << "очередь полна" <<endl ;**

**return ;**

**}**

**rear++;**

**if ( rear==SIZE ) rear = 0 ;**

**queue [ rear ] = num;**

**}**

**// Извлечение элемента из очереди**

**void Queue::pop() {**

**if ( frnt == rear ) {**

**cout << "очередь пуста" <<endl ;**

**return;**

**}**

**frnt++;**

**if ( frnt==SIZE ) frnt = 0 ;}**

**//Определение размера очереди**

**int Queue::size() {**

**int s=0;**

**for(int i=frnt; i<rear; i++)**

**s++;**

**return s;**

**}**

**// Последний элемент очереди**

**int Queue::back() {**

**return queue[rear]; }**

**// Первый элемент очереди**

**int Queue::front() {**

**return queue[frnt+1]; }**

**int main()**

**{**

**setlocale(LC\_ALL, "Russian");**

**Queue queue1;**

**int i;**

**for (i= 1 ; i <= 5 ; i++ )**

**queue1.push ( i ) ;**

**cout<<"Изначальная очередь ";**

**queue1.out();**

**cout<<endl;**

**cout<<"введите еще элемент: ";**

**cin>>i;**

**queue1.push(i);**

**cout<<"теперь очередь имеет следующий вид"<<endl;**

**queue1.out();**

**cout<<endl<<"Размер очереди:"<<endl;**

**cout<<queue1.size();**

**cout<<endl <<"последний элемент:"<< endl;**

**cout<<queue1.back();**

**cout<<endl<<"первый элемент"<<endl;**

**cout<<queue1.front();**

**cout<<endl <<"удаление элемента";**

**queue1.pop();**

**cout<<endl <<"текущие данные";**

**queue1.out();**

**cout<<endl <<"еще одно добавление элемента";**

**queue1.push(i);**

**queue1.out();**

**cout<<endl;**

**cin.get(); cin.get();**

**return 0;**

**}**

### 3.2 Листинг ( Python )

class Queue:

def \_\_init\_\_(self):

self.items = []

def isEmpty(self):

return self.items == []

def enqueue(self, item):

self.items.insert(0,item)

def dequeue(self):

return self.items.pop()

def size(self):

return len(self.items)

q=Queue()

q.isEmpty()

q.enqueue('dog')

q.enqueue(4)

q=Queue()

q.isEmpty()

q.enqueue(4)

q.enqueue('dog')

q.enqueue(True)

### Заключение

В результате выполнения курсового проекта была написана программа с реализацией очереди на языке программирования C++ и Python.

В ходе работы была проанализирована предметная область, существующие разработки, посвященные данному направлению, получены практические навыки с работой в Code::Blocks, PyCharm.

### Список литературы

1. <https://ru.wikipedia.org/wiki/Очередь_(программирование)>

2. <https://prog-cpp.ru/data-queue/>

3. <https://codelessons.ru/cplusplus/ochered-queue-v-c-realizaciya-i-chto-eto-voobshhe-takoe.html>

4. <https://tproger.ru/translations/stacks-and-queues-for-beginners/>

5. <https://ru.wikipedia.org/wiki/Python>

6. <https://python-scripts.com/queues>