Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского

Факультет вычислительной математики и кибернетики

Отчёт по лабораторной работе

Очередь на списке

Выполнил:
студент ф-та ИИТММ гр. 381908-01
Козел С. А.

Проверил:
ассистент каф. МОСТ, ИИТММ

Лебедев И.Г.

Нижний Новгород
2020 г.

Содержание

Введение

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.

Постановка задачи

Цель данной работы - разработка структуры данных для хранения очереди с использованием списка и реализация стандартных функций очереди.

Выполнение работы предполагает решение следующих задач:

1. Реализация класса стека Queue.
2. Реализация метода для нахождения максимального элемента в очереди.
3. Реализация метода для нахождения минимального элемента в очереди
4. Реализация методов для ввода/вывода структуры данных в файл
5. Публикация исходных кодов в личном репозиторий на GitHub.

Руководство пользователя

Пользователю нужно запустить файл QueueList.exe.

Откроется консольное приложение для тестирования очереди.

Программа покажет функциональность каждой функции по средствам вывода данных в консоль.

Руководство программиста

Описание структуры программы

Программа состоит из следующих модулей:

- Приложение QueueList
- Статическая библиотека Queue:
  - Queue.h – описание класса очереди
- Приложение main:
  - main.cpp – тестирование работы программы

Описание структур данных

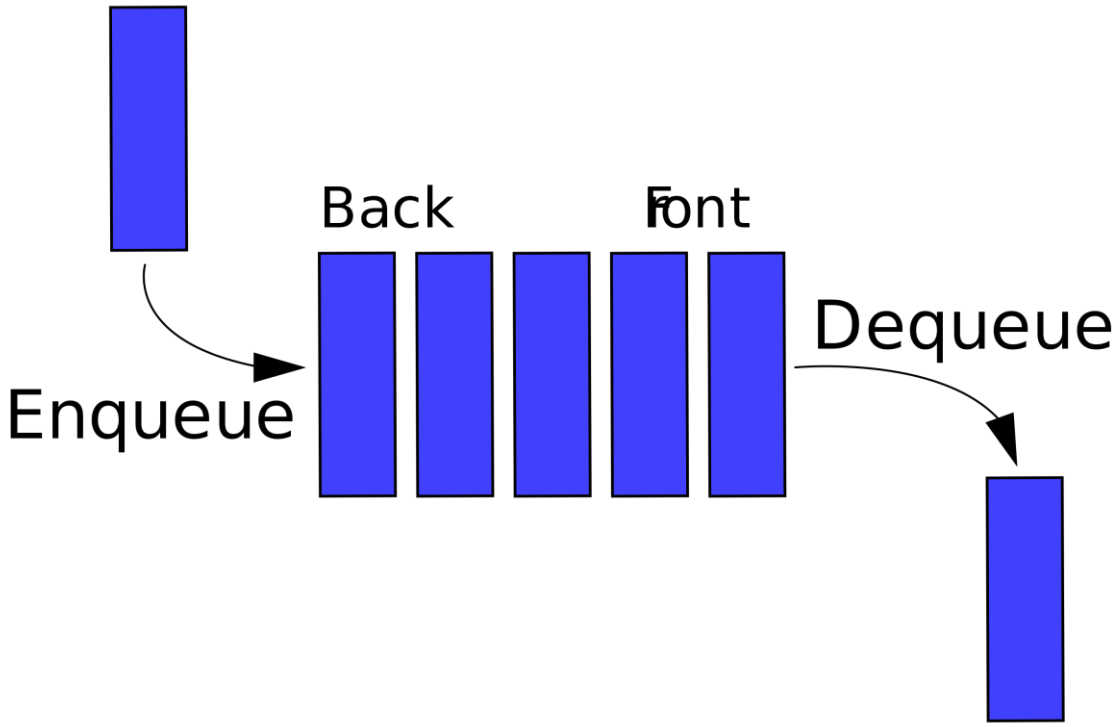
Класс Queue:

```
1. void push(T val) – вставка в конец очереди;
2. void pop() – удаление с начала очереди;
3. bool empty() – проверка на пустоту;
4. int size() – получить кол-во элементов в очереди;
5. T front() – просмотр элемента в начале;
6. T back() – просмотре элемента в конце;
7. T FindMin() – поиск минимального элемента;
8. T FindMax() – поиск максимального элемента;
9. void OutputFile(std::string path) – вывод очереди в файл;
10. void FromFile(std::string path) – чтение очереди с файла;
11. void print() – вывести очередь в консоль;
```

Описание алгоритмов

Принцип работы очереди

Очередь — абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» (FIFO). Добавление элемента возможно лишь в конец очереди, выборка — только из начала очереди, при этом выбранный элемент из очереди удаляется.



Эксперименты

```
Тест Queue
Исходные данные:
14 12 4 13 1
Front - 14 Back - 1 Size - 5
Front - 12 Back - 1 Size - 4
Front - 4 Back - 1 Size - 3
Front - 13 Back - 1 Size - 2
Front - 1 Back - 1 Size - 1
Size - 0

Pop из пустого Queue:
Cannot get element from queue: empty queue.

Front из пустого Queue:
Cannot get element from queue: empty queue.

Back из пустого Queue:
Cannot get element from queue: empty queue.

250 184 231 235 206
Минимальный элемент - 184
Максимальный элемент - 250


Добавление элемента в заполненный stack:
Cannot put element in queue: full queue.File opened and rewritten!
Очередь не изменилась: 184 231 235 206 250
File is open and read!
Очередь после чтения из файла: 41 94 41 45 43
Для продолжения нажмите любую клавишу . . .
```

Заключение

При выполнении данной работы мною была полностью изучена и успешно реализована структура данных очередь с использованием списка.

Литература

1. https://habr.com/ru/post/457068/
2. https://prog-cpp.ru/data-queue/
3. https://ru.wikipedia.org/wiki/Очередь_(программирование)

Приложения

Приложение 1

Queue.h

```
template<class T>
class Queue
{
private:
    std::list<T> list;
public:
    void push(T val);
    void pop();
    bool empty();
    int size();
    T front();
    T back();

    T FindMin();
    T FindMax();

    void OutputFile(std::string path);
    void FromFile(std::string path);

    void print();
};

template<class T>
void Queue<T>::push(T val)
{
    list.push_front(val);
}

template<class T>
void Queue<T>::pop()
{
    if (empty()) { throw std::out_of_range("Queue is empty"); }
    list.pop_back();
}

template<class T>
bool Queue<T>::empty()
{
    if (list.size() == 0) { return true; }
    else { return false; }

}

template<class T>
int Queue<T>::size()
{
    return list.size();
}

template<class T>
T Queue<T>::front()
{
    if (empty()) { throw std::out_of_range("Queue is empty"); }
    return list.back();
}

template<class T>
T Queue<T>::back()
{
    if (empty()) { throw std::out_of_range("Queue is empty"); }
    return list.front();
}
```

```cpp
template<class T>
T Queue<T>::FindMin()
{
    if (empty()) { throw std::out_of_range("Queue is empty"); }
    int countElements = size();
    Queue<T> tmp;
    T min = front();

    for (int i = 0; i < countElements; i++)
    {
        if (front() < min) { min = front(); }
        tmp.push(front());
        pop();
    }
    for (int i = 0; i < countElements; i++)
    {
        T tmpVal = tmp.front();
        push(tmpVal);
        tmp.pop();
    }
    return min;
}

template<class T>
T Queue<T>::FindMax()
{
    if (empty()) { throw std::out_of_range("Queue is empty"); }
    int countElements = size();
    Queue<T> tmp;
    T max = front();

    for (int i = 0; i < countElements; i++)
    {
        if (front() > max) { max = front(); }
        tmp.push(front());
        pop();
    }
    for (int i = 0; i < countElements; i++)
    {
        T tmpVal = tmp.front();
        push(tmpVal);
        tmp.pop();
    }
    return max;
}

template<class T>
void Queue<T>::OutputFile(std::string path)
{
    if (empty()) { throw std::out_of_range("Queue is empty"); }
    std::fstream fs;
    fs.open(path, std::ios::out | std::ios::trunc);
    if (!fs.is_open())
    {
        std::cout << "Error open file!" << std::endl;
    }
    else
    {
        int countElements = size();
        Queue<T> tmp;
        for (int i = 0; i < countElements; i++)
        {
            tmp.push(front());
            pop();
        }
        for (int i = 0; i < countElements; i++)
        {
            T tmpVal = tmp.front();
            push(tmpVal);
            fs << tmpVal << " ";
            tmp.pop();
        }
        std::cout << "File opened and rewritten!" << std::endl;
    }
    fs.close();
}
```

```cpp
template<class T>
void Queue<T>::FromFile(std::string path)   // only int
{
    std::fstream fs;
    fs.open(path, std::ios::in | std::ios::app);
    if (!fs.is_open())
    {
        std::cout << "Error open file!" << std::endl;
    }
    else
    {
        std::string str;
        std::string num;
        getline(fs, str);

        for (int i = 0; i < str.size(); i++)
        {
            if (((str[i] == 45) || ((int(str[i]) > 47) && (int(str[i] < 58)))))
            {
                num += str[i];
            }
            else
            {
                push(atoi(num.c_str()));
                num.clear();
            }
        }
    }
    fs.close();
}


template<class T>
void Queue<T>::print()
{
    if (empty()) { throw std::out_of_range("Queue is empty"); }
    int countElements = size();
    Queue<T> tmp;
    for (int i = 0; i < countElements; i++)
    {
        tmp.push(front());
        pop();
    }
    for (int i = 0; i < countElements; i++)
    {
        T tmpVal = tmp.front();
        push(tmpVal);
        std::cout << tmpVal << " ";
        tmp.pop();
    }
}
```

Приложение 2

Main.cpp

```cpp
#define PATH_FOR_FILE_INT "C:\\outINT.txt"

int main()
{
    setlocale(LC_ALL, "RUS");

    Queue<int> q1;
    int countElements = 8;
    srand(time(NULL));
    for (int i = 0; i < countElements; i++)
    {
        int num = -5 + rand() % 30;
        q1.push(num);
        std::cout << num << " ";
    }
    std::cout << std::endl;

    q1.print();

    std::cout << "\nMin: " << q1.FindMin();
    std::cout << "\nMax: " << q1.FindMax() << std::endl;

    q1.OutputFile(PATH_FOR_FILE_INT);

    Queue<int> q2;
    q2.FromFile(PATH_FOR_FILE_INT);
    std::cout << "После прочтения из файла: ";
    q2.print();
    std::cout << std::endl << std::endl;

    //Tests
    std::cout << std::endl;
    Queue<int> q_t1;
    std::cout << "Empty из пустой очереди: " << q_t1.empty() << std::endl;  // 1 - pustoi, 0 - ectb element

    q_t1.push(1);
    q_t1.push(10);
    q_t1.push(100);
    std::cout << std::endl;
    std::cout << "Очередь: ";
    q_t1.print();
    std::cout << "\nTop: " << q_t1.front() << " Back: " << q_t1.back() << std::endl;
    q_t1.pop();
    std::cout << "Top: " << q_t1.front() << " Back: " << q_t1.back() << std::endl;
    q_t1.pop();
    std::cout << "Top: " << q_t1.front() << " Back: " << q_t1.back() << std::endl;
    q_t1.pop();

    std::cout << std::endl;
    std::cout << "Pop из пустой очереди: ";
    try
    {
        q_t1.pop();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    std::cout << "Front из пустой очереди: ";
    try
    {
        q_t1.front();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;
```

```
    std::cout << "Back из пустой очереди: ";
    try
    {
        q_t1.back();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    std::cout << "FindMax из пустой очереди: ";
    try
    {
        q_t1.FindMax();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    std::cout << "FindMin из пустой очереди: ";
    try
    {
        q_t1.FindMin();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    std::cout << "OutputFile из пустой очереди: ";
    try
    {
        q_t1.OutputFile(PATH_FOR_FILE_INT);
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    std::cout << "Print из пустой очереди: ";
    try
    {
        q_t1.print();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    system("pause");
    return 0;
```