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

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

Описание структур данных

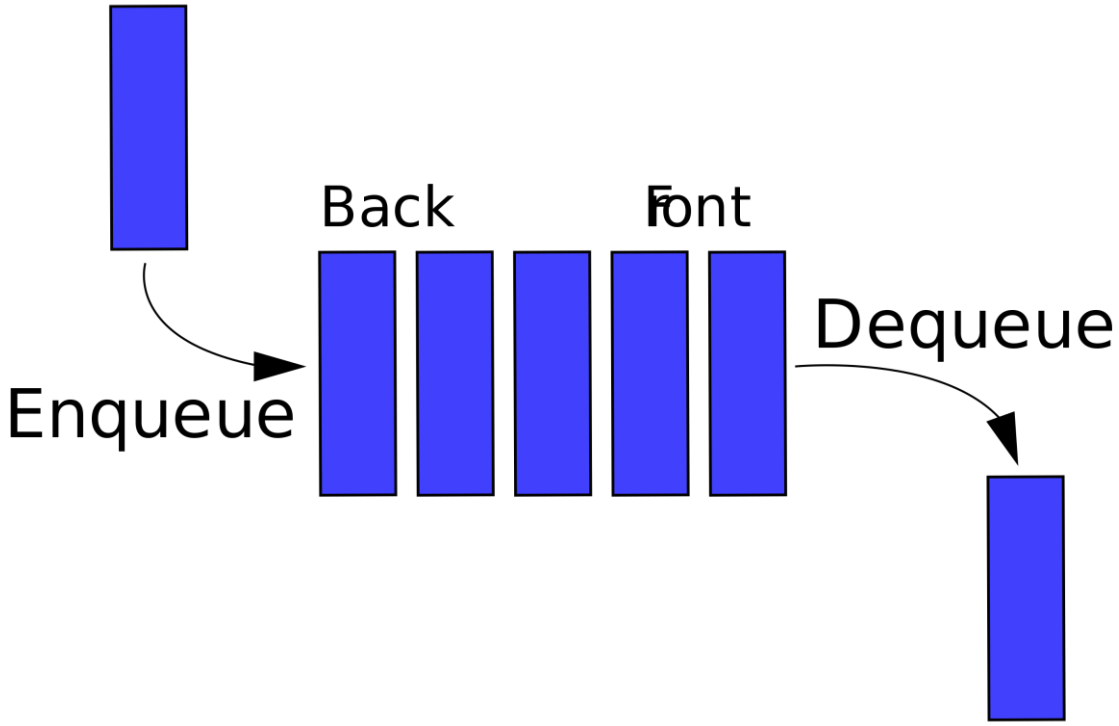
Класс Queue:

```
1. queue(int Size = MemorySize) - конструктор;
2. ~queue() - диструктор;
3. bool Empty(void) const – проверка пуст ли стек;
4. bool Full(void) const – проверка полон ли стек;
5. void Put(const T& Val) – вставка элемента;
6. int  Size()const – получить кол-во элементов в очереди;
7. T Pop(void) – вытягивание первого элемента из очереди;
8. T Back(void)const – просмотр последнего элемента;
9. T Front(void)const – просмотр первого элемента;
10. void InFile(std::string file_name) – вывод СД в файл;
11. void FromFile(std::string file_name) – чтение очереди с файла;
12. T FindMax(void) – поиск максимума;
13. T FindMin(void) – поиск минимума;
14. void Print(void) – вывод 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.

Литература

1. https://habr.com/ru/post/457068/
2. https://ru.wikipedia.org/wiki/Очередь_(программирование)

Приложения

Приложение 1

Queue.h

```h
#pragma once
#define MemorySize 15           // Размер памяти по умолчанию
#include <iostream>
#include <fstream>
#include <string>

template <class T>
class queue {
protected:
    T*   pMem;
    int  hi;
    int  li;
    int  MemSize;
    int  DataCount;
    int  GetNextIndex(int index);

public:
    queue(int Size = MemorySize);
    ~queue();

    bool Empty(void) const;
    bool Full(void) const;
    void Put(const T& Val);
    int  Size()const;

    T Pop(void);
    T Back(void)const;
    T Front(void)const;

    void InFile(std::string file_name);
    void FromFile(std::string file_name);

    T FindMax(void);
    T FindMin(void);
    void Print(void);
};
```

```cpp
template <class T>
queue<T>::queue(int Size)
{
    MemSize = Size;
    DataCount = 0;
    hi = -1;
    li = 0;
    pMem = new T[MemSize];
}

template<class T>
queue<T>::~queue()
{
    delete[] pMem;
    pMem = NULL;
}

template <class T>
int queue<T>::GetNextIndex(int index)
{
    return ++index % MemSize;
}

template <class T>
bool queue<T>::Full(void) const
{
    return DataCount == MemSize;
}

template <class T>
void queue<T>::Put(const T& Val)
{
    if (pMem == NULL) {
        throw std::logic_error("Cannot put element in queue: empty memory.");
    }
    else if (Full()) {
        throw std::logic_error("Cannot put element in queue: full queue.");
    }
    else {
        hi = GetNextIndex(hi);
        pMem[hi] = Val;
        DataCount++;
    }
}
```

```cpp
template<class T>
inline int queue<T>::Size() const
{
    return DataCount;
}

template <class T>
bool queue<T>::Empty(void) const
{
    return DataCount == 0;
}

template <class T>
T queue<T>::Front()const
{
    if (pMem == NULL) {
        throw std::logic_error("Cannot get element from queue: empty memory.");
    }
    else if (Empty()) {
        throw std::logic_error("Cannot get element from queue: empty queue.");
    }
    else {
        return pMem[li];
    }
}

template <class T>
T queue<T>::Back()const
{
    if (pMem == NULL) {
        throw std::logic_error("Cannot get element from queue: empty memory.");
    }
    else if (Empty()) {
        throw std::logic_error("Cannot get element from queue: empty queue.");
    }
    else {
        return pMem[hi];
    }
}

template <class T>
T queue<T>::Pop()
{
    if (pMem == NULL) {
        throw std::logic_error("Cannot get element from queue: empty memory.");
    }
    else if (Empty()) {
        throw std::logic_error("Cannot get element from queue: empty queue.");
    }
    else {
        DataCount--;
        li = GetNextIndex(li);
        return pMem[li];
    }
}
```

```cpp
void queue<T>::InFile(std::string file_name)
{
    std::fstream fs;
    fs.open(file_name, std::fstream::in | std::fstream::out);
    if (!fs.is_open())
    {
        std::cout << "Error open file!" << std::endl;
        return;
    }
    queue<T> tmp(MemSize);
    while (!Empty())
    {
        T val = Pop();
        tmp.Put(val);
        fs << val << " ";
    }
    while (!tmp.Empty())
    {
        Put(tmp.Pop());
    }
    std::cout << "File opened and rewritten!" << std::endl;
    fs.close();
}

template<class T>  <T> Укажите аргументы примера шаблона для IntelliSense
void queue<T>::FromFile(std::string file_name)
{
    std::fstream fs;
    fs.open(file_name, std::fstream::in | std::fstream::out);
    if (!fs.is_open())
    {
        std::cout << "Error open file." << std::endl;
        return;
    }
    std::string str;
    getline(fs, str);

    char num[20];                   // Запись числа из файла
    int* nums = new int[MemSize];   // Храним числа из файла

    int j = 0, k = 0;
    bool isNum = true;
    for (int i = 0; i < str.size(); i++)
    {
        if (str[i] == ' ')
        {
            isNum = false;
            nums[k] = atoi(num);
            k++;
            j = 0;
            for (int m = 0; m < 20; m++)
            {
                num[m] = ' ';
            }
        }
        else { isNum = true; }

        if (isNum)
        {
            if((str[i] >= 48) && (str[i] <= 57))
            {
                num[j] = str[i];
                j++;
            }
            else
            {
                Put(str[i]);
                j++;
            }
        }
    }

    for (int i = 0; i < MemSize; i++)
    {
        try
        {
            Put(nums[i]);
        }
        catch(std::logic_error err)
        {
            break;
        }
    }

    std::cout << "File is open and read!" << std::endl;

    fs.close();
}
```

```cpp
template <class T>
inline T queue<T>::FindMax(void)
{
    queue<T> tmp(MemSize);
    T val = Pop();
    tmp.Put(val);
    while (!Empty())
    {
        if (val < Front())
        {
            val = Front();
        }
        tmp.Put(Pop());
    }
    while (!tmp.Empty())
    {
        Put(tmp.Pop());
    }

    return val;
}

template<class T>
inline T queue<T>::FindMin(void)
{
    queue<T> tmp(MemSize);
    T val = Pop();
    tmp.Put(val);
    while (!Empty())
    {
        if (val > Front())
        {
            val = Front();
        }
        tmp.Put(Pop());
    }
    while (!tmp.Empty())
    {
        Put(tmp.Pop());
    }

    return val;
}

template <class T>
void queue<T>::Print()
{
    for (int i = 0; i < DataCount; i++)
    {
        std::cout << pMem[i] << " ";
    }
    std::cout << "\n";
}
```

Приложение 2

Main.cpp

```cpp
#include "Queue.h"
#include <locale>

#define PATH_FOR_FILE  "C:\\out.txt"

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

    int sizeQueue = 5;

    std::cout << "Тест Queue\n";
    queue<int> qu(sizeQueue);

    std::cout << "Исходные данные:\n";
    srand(time(NULL));

    for (int i = 0; i < sizeQueue ; i++)
    {
        int val = rand() % 15;
        std::cout << val << " ";
        qu.Put(val);
    }
    std::cout << "\n";

    std::cout << "Front - " << qu.Front() << " Back - " << qu.Back() << " Size - " << qu.Size() << "\n";
    qu.Pop();

    std::cout << "Front - " << qu.Front() << " Back - " << qu.Back() << " Size - " << qu.Size() << "\n";
    qu.Pop();

    std::cout << "Front - " << qu.Front() << " Back - " << qu.Back() << " Size - " << qu.Size() << "\n";
    qu.Pop();

    std::cout << "Front - " << qu.Front() << " Back - " << qu.Back() << " Size - " << qu.Size() << "\n";
    qu.Pop();

    std::cout << "Front - " << qu.Front() << " Back - " << qu.Back() << " Size - " << qu.Size() << "\n";
    qu.Pop();
    std::cout << "Size - " << qu.Size() << "\n\n";

    std::cout << "Pop из пустого Queue:\n";
    try
    {
        qu.Pop();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }

    std::cout << "\n\n";
```

```
    std::cout << "Front из пустого Queue:\n";
    try
    {
        qu.Front();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }

    std::cout << "\n\n";

    std::cout << "Back из пустого Queue:\n";
    try
    {
        qu.Back();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }

    std::cout << "\n\n";

    for (int i = 0; i < sizeQueue; i++)
    {
        int val = 1 + rand() % 300;
        qu.Put(val);
    }
    qu.Print();

    std::cout << "Минимальный элемент - " << qu.FindMin() << "\n";
    std::cout << "Максимальный элемент - " << qu.FindMax() << "\n";

    std::cout << "\n";

    std::cout << "\nДобавление элемента в заполненный stack:\n";
    try
    {
        qu.Put(3);
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }

    qu.InFile(PATH_FOR_FILE);
    std::cout << "Очередь не изменилась: ";
    qu.Print();

    queue<int> qu_2(sizeQueue);
    qu_2.FromFile(PATH_FOR_FILE);
    std::cout << "Очередь после чтения из файла: ";
    while (!qu_2.Empty())
    {
        std::cout << qu_2.Pop() << " ";
    }

    std::cout << "\n";

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