**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе № 3**

**по дисциплине «Программирование»**

**Тема: Линейные структуры данных: стек, очередь, дек**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Любчук Д.В. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Цель работы.**

Познакомиться с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование стека, очереди и дека для решения задач.

**Постановка задачи.**

Задание:

За один просмотр заданного файла F (типа file of real) и без использования дополнительных файлов вывести элементы файла F в следующем порядке: сначала - все числа, меньшие a, затем - все числа на отрезке [a, b] и наконец - все остальные числа, сохраняя исходный взаимный порядок в каждой из этих групп чисел (a и b задаются пользователем, a < b).

При ршении задачи испольовать очередь.

**Основные теоретические положения.**

Очередь представляет собой разновидность линейного списка.

При задании спецификации линейных списков использовалась модель последовательности. Доступ к каждому элементу последовательности можно получить, продвигаясь по списку от одного элемента к другому. В модели выделяется пройденная часть и текущий элемент, с которого начинается еще не пройденная часть. При этом определяют функции над последовательностями:

* **First** возвращает первый элемент последовательности
* **Last** возвращает последний элемент последовательности
* **Rest** возвращает исходную последовательность, в которой присутствуют все элементы за исключением первого,
* **Lead** возвращает исходную последовательность, в которой присутствуют все элементы за исключением последнего,
* **Prefix** добавляет элемент в начало последовательности
* **Postfix** добавляет элемент в конец последовательности.

Доступ к элементам последовательности (чтение и запись) , осуществляется только через ее начало и конец. Функции First, Rest, Last, Lead − селекторы, а функции Prefix и Postfix − конструкторы.

Если использовать только набор функций First, Rest, Postfix (или только Last, Lead, Prefix) получим структуру данных, которая называется очередь (англ. queue). Во все эти структуры данных необходимо добавить еще предикат-индикатор Null, идентифицирующий пустую последовательность, и либо константу, обозначающую пустую последовательность ∆, либо функцию Crtate, порождающую пустую последовательность.

Ссылочная реализация очереди в динамической памяти в основном аналогична ссылочной реализации линейных списков. Упрощение связано с отсутствием необходимости работать с текущим элементом списка. Идеи такой реализации ясны из рисунка 3.3.

Для ссылочной реализации дека естественно использовать двунаправленный список.

Поскольку для очереди доступ к элементам осуществляется только через начало и конец последовательности, то эта структура данных допускает и эффективную непрерывную реализацию на базе вектора. При этом используется одномерный массив Mem: array [0..n] of α и переменная Верх :  -1..n.

Для пустого стека Верх = = -1, для целиком заполненного стека Верх = = n.

Непрерывная реализация ограниченной очереди на базе вектора требует двух переменных Начало и Конец.

Особенностью такого представления является наличие ситуации, когда последовательность элементов очереди по мере их добавления может выходить за границу вектора, продолжаясь с его начала (вектор имитирует здесь так называемый кольцевой буфер). Эта ситуация изображена на рис. 1.

***0***

***1***

***2***

***n***

***Mem:***

**X X X X X X X X X X X**

***Начало***

***Конец***

Рис. 1 - Непрерывное представление очереди в кольцевом буфере

Переменная **Начало** может принимать значение **Конец+1** в случаях как пустой, так и полной очереди. Чтобы различать эти ситуации, надо ввести еще одну переменную **Длина**. Для пустой очереди Длина = 0, а для полной очереди Длина = n+1.

**Спецификация программы.**

*Назначение программы*.

Программа предназначена для вывода содержимого файла F.txt в определённом порядке.

*Описание программы*.

Программа написана на языке С++ с использованием компилятора ml64 9.0.30729.1. Входными данными для программы является файл F.txt, в котором записано какое-то количество чисел и 2 числа (a и b) типа double. Проверка входных данных из файла отсутствует (есть проверка на наличие самого файла), реализована проверка чисел a и b. Выходными данными является информация, есть ли данный элемент в списке или нет.

*Пример диалога с пользователем*.

Введите а: 2

Введите b: 6

Содержимое файла:

11 2 4 10 14 4 3 3 7 14 5 5

После преобразования:

2 4 4 3 3 5 5 11 10 14 7 14

*Реализация*.

Функции :

1. Queue();

Конструктор класса. Выделяет память под store, устанавливает метки начала и конца в 0, присваивает size значение n, флаг заполненности – в false.

1. ~Queue();

Деструктор класса. Освобождает выделенную очередь память.

1. void Queue::put(double a) // функция добавления элемента a в очередь
2. double Queue::take() // функция получения элемента из очереди

Выходные параметры:

Новая последовательность чисел.

**Тестирование:**

|  |  |  |
| --- | --- | --- |
| № | Исходные данные | Результат |
| 1 | a = 3.5  b = 10.8  F = {2 5.8 1 9.8 3 11 12 10.9 7} | 2 1 3 5.8 9.8 7 11 12 10.9 |
| 2 | a = 1  b = 8  F = {3 4 5 6 7 5.5 9 3.3} | 3 4 5 6 7 5.5 3.3 9 |
| 3 | a = 35  b = 88  F = {6 5 8 9 2 3 4 0 8 9 6 2 4 7 33 1 6} | 6 5 8 9 2 3 4 0 8 9 6 2 4 7 33 1 6 |
| 4 | a = 8  b = 9  F = {} | Файл пуст! |
| 5 | a = 3  b = 7  F = {} | Файл пуст! |
| 6 | a = 3  b = 6  F = {9 8 7 6 5 4 3 2 1 1 2 3 4 5 6 7 8 9} | 1 2 2 1 3 4 5 6 6 5 4 3 7 8 9 9 8 7 |
| 7 | a = -2,5  b = 2,5  F = {-4,5 8 0 9,6 7 2 0 1 -3 9 9 7 5 8} | -4.5 -3 0 2 0 1 8 9.6 7 9 9 7 5 8 |
| 8 | a = 10  b = 15  F = {0 1 2 3 4 5 6 7 8 9} | 0 1 2 3 4 5 6 7 8 9 |
| 9 | a = 9  b = 17  F = {25 23 29 27 55 3} | 25 23 29 27 55 3 |
| 10 | a = 3  b = 7  F = {4,1 5 6,1 4} | 4,1 5 6,1 4 |

**Выводы.**

В ходе выполнения лабораторной работы по изучению линейных структур данных мы познакомились с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование стека.

**Приложение А.**

**Исходный код.**

Queue.h

#pragma once

#include <stdlib.h>

#include <stdio.h>

class Queue

{

public:

Queue();

void put(double); // Функция добавления элемента в очередь

double take(); // Функция получения элемента из очереди

bool isEmpty(); // Функция проверки на пустоту очереди

bool isFull(); // Функция проверки на наличие свободного места в очереди

private:

static const int maxSize=512; // Максимальное количество элементов в очереди

double arr[maxSize]; // Массив, в котором хранятся элементы очереди

int start, size; // Индекс первого элемента и их кол-во

};

Queue.cpp

#include "queue.hpp"

Queue::Queue() : size(0), start(0) // Инициализация переменных

{}

void Queue::put(double a) // Определение функции добавления элемента a в очередь

{

if (!isFull()) // Если в очереди есть место

{

arr[(start+size)%maxSize]=a; // То добавляем элемент после последнего(с номером (start+size)%maxSize)

++size;

} else // Иначе ошибка переполнения

{

printf("Queue overflow.\n");

exit(1);

}

}

double Queue::take() // Определение функции получения элемента из очереди

{

if (!isEmpty()) // Если в очереди есть элементы

{

double b = arr[start]; // Сохраняем первый элемент

start=(++start%maxSize); // Меняем индекс первого элемента на следующий (если текущий элемент был 511 - то следующим будет 0)

--size;

return b;

} else // Иначе ошибка

{

printf("Can't take an element from empty queue.\n");

exit(1);

}

}

bool Queue::isEmpty()

{

return (size==0);

}

bool Queue::isFull()

{

return (size==maxSize);

}

main.cpp

#include <stdio.h>

#include <stdlib.h>

#include "queue.hpp"

void f1(Queue in); // Функция для выполнения первой части задания

void f2(Queue in); // Второй

void f3(Queue in); // Третьей

double a,b;

int main(int argc, char const \*argv[])

{

FILE\* f = fopen("/Users/denislubchyk/Documents/studies/lab3/lab\_3.1/lab\_3.1/input.txt", "r"); // Открытие файла

if(!f)

{

printf("Can't open file \"input.txt\"\n");

exit(1);

}

printf("Input a and b: \n");

scanf("%lf %lf", &a, &b);

if (a>=b)

{

printf("Wrong input.\n");

exit(1);

}

double d;

Queue q;

while (!feof(f)) // Чтение из файла

{

if (fscanf(f, "%lf", &d)>0) // Если данные прочитались верно

{

q.put(d);

}

}

f1(q);

f2(q);

f3(q);

fclose(f);

return 0;

}

void f1(Queue in)

{

double d;

while(!in.isEmpty())

{

d = in.take();

if (d<a) printf("%lf ", d);

}

}

void f2(Queue in)

{

double d;

while(!in.isEmpty())

{

d = in.take();

if (d>=a && d<=b) printf("%lf ", d);

}

}

void f3(Queue in)

{

double d;

while(!in.isEmpty())

{

d = in.take();

if (d>b) printf("%lf ", d);

}

}