**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 |  | Азаревич А.Д. |
| 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*.

Классы:

1. Binomial – класс, являющий собой очередь.

Переменные класса:

1. double \*store - массив, являющийся кольцевым буфером (т.е. сама очередь).
2. int begin – метка начала очереди.
3. int end – метка конца очереди.
4. int size – размер очереди.
5. bool full – флаг заполненности очереди (true – заполнено, false – есть свободные метста).

Функции класса:

1. Queue(int n = 100);

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

1. ~Queue();

Деструктор класса. Освобождает выделенную очередь память.

1. bool push(type item);

Вставляет в очередь новый элемент;

Входные параметры:

Значение item, которое надо вставить.

Возвращаемое значение:

True, если элемент вставлен, false, если вся очередь была заполнена;

Постусловие:

Если под новый элемент нет места устанавливает флаг full в true.

1. bool pop(type &item);

Извлекает из очереди первый элемент.

Входные параметры:

Переменная item, в которую поместится извлечённое значение

Выходные параметры:

Переменная item, в которую поместится извлечённое значение

Возвращаемое значение:

True, если элемент извлечён, false, если очередь была пуста.

Побочные действия:

Устанавливает full == false.

Функции:

* + - 1. bool F\_read(istream &in, Queue &lessA, Queue &greaterB, Queue &A\_B, type a, type b)

Считывает файл F.txt, заполняя 3 очереди.

Входные параметры:

Поток вывода из файла, три очереди, значения a и b;

Выходные параметры:

3 заполненные определённым образом очереди;

Возвращаемое значение:

False если в одной из очередей закончилось место, иначе true.

2. void F\_write(Queue &lessA, Queue &greaterB, Queue &A\_B)

Выводит сначала числа из очереди с числами меньшими a, потом из очереди с промежутком [a;b], в конце из очереди с числами большими b.

Входные параметры:

3 очереди;

Выходные параметры:

Новая последовательность чисел.

**Тестирование.**

В файле F.txt:

**11;2.7;4;10;14.303;4;-3;0;3;7.453;14;5;5.02**

|  |  |  |
| --- | --- | --- |
| A | B | результат |
| 0 | 3 | -3; 2.7; 0; 3; 11; 4; 10; 14.303; 4; 7.453; 14; 5; 5.02 |
| -5 | 5 | 2.7; 4; 4; -3; 0; 3; 5; 11; 10; 14.303; 7.453; 14; 5.02 |
| 5 | 3 | Ошибка: а должно быть меньше b |
| 7 | 10 | 2.7; 4; 4; -3; 0; 3; 5; 5.02; 10; 7.453; 11; 14.303; 14 |

**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

#include <iostream>

typedef double type;

class Queue

{

public:

Queue(int n = 100);

~Queue();

bool push(type item);//Поместить в очередь

bool pop(type &item);//Изъять из очереди

private:

type \*store;//Очередь

int begin;//Метка начала очереди

int end;//Метка конца очереди

int size;//Размер очереди

bool full;//Флаг "отсутствия свободных мест"

};

Queue.cpp

#include "Queue.h"

using namespace std ;

extern void GLOBAL\_ERRORS(int num);

Queue::Queue(int n)

{

store = (type \*)malloc(n\*sizeof(type) );

if ( !store )

GLOBAL\_ERRORS(1);

begin = end = 0;

size = n;

full = false;

}

Queue::~Queue()

{

free(store);

}

bool Queue::push(type item)

{

if ( full )

{

//cerr << "Предупреждение: очередь заполнена\n";

return false;

}

store[end] = item;

end = (end +1)%size;

if (begin == end) full = true;

return true;

}

bool Queue::pop(type &item)

{

if ( (begin == end) && !full)

{

//cerr << "Предупреждение: очередь пуста\n";

return false;

}

item = store[begin];

begin = (begin +1)%size;

full = false;

return true;

}

main.cpp

//

// Лабораторная работа №3

// main.cpp

// Азаревич артём, группа 6383

// 16.09.2017

//

// Программа для вывода содержимого файла F в определённом порядке

#include <fstream>

#include "Queue.h"

using namespace std ;

void GLOBAL\_ERRORS(int num);

bool F\_read(istream &in, Queue &lessA, Queue &greaterB, Queue &A\_B, type a, type b);

void F\_write(Queue &lessA, Queue &greaterB, Queue &A\_B);

void main()

{

setlocale(0, "");

Queue lessA;

Queue greaterB;

Queue A\_B;

ifstream fin("F.txt");

if (!fin)

GLOBAL\_ERRORS(0);

type a, b;

cout <<"Введите а: ";

cin >> a;

cout <<"Введите b: ";

cin >> b;

if (a >= b)

GLOBAL\_ERRORS(2);

if ( !F\_read(fin, lessA, greaterB, A\_B, a, b) )

cerr << "Предупреждение: одна из очередей переполнилась\n";

F\_write(lessA, greaterB, A\_B);

char c;

cin >> c;

}

void GLOBAL\_ERRORS(int num)

{

switch(num)

{

case 0: cerr << "Ошибка: файл с исходными данными не найден"; break;

case 1: cerr << "Ошибка: Память не выделена"; break;

case 2: cerr << "Ошибка: а должно быть меньше b"; break;

case 3: cerr << "Ошибка: в файле должны быть ТОЛЬКО числа\n";

break;

default: return;

}

exit(1);

}

bool F\_read(istream &in, Queue &lessA, Queue &greaterB, Queue &A\_B, type a, type b)

{

type item;

cout << "Содержимое файла:\n";

while(!in.eof())

{

in >> item;

if (!in.good() && !in.eof())

GLOBAL\_ERRORS(3);

cout << item <<ends;

if ( item < a )

if( !lessA.push(item) ) return false;

else

if ( item > b )

if( !greaterB.push(item) ) return false;

else

if ( !A\_B.push(item) ) return false;

}

return true;

}

void F\_write(Queue &lessA, Queue &greaterB, Queue &A\_B)

{

type item;

cout <<"\nПосле преобразования:\n";

while( lessA.pop(item) )

cout << item << ends;

while( A\_B.pop(item) )

cout << item << ends;

while( greaterB.pop(item) )

cout << item << ends;

}

system("pause");

}

bool found(const base &x, const list y)

{

if( isAtom(y) )

if (y->node.atom == x) return true;

else return false;

if( found(x, y->node.pair.head) ) return true;

if ( !y->node.pair.tail ) return false;

if( found(x, y->node.pair.tail) ) return true;

return false;

}