Министерство Образования и Науки Российской Федерации  
Новосибирский Государственный Технический Университет  
Кафедра Прикладной Математики

**Лабораторная работа № 2**

**по дисциплине «Структуры данных и алгоритмы»**

Факультет: ФПМИ

Группа: ПМ-53

Студент: Тябин Егор Алексеевич

Преподаватель: Еланцева Ирина Леонидовна

### Новосибирск 2015

# Условие задачи

1.Написать операции работы с заданной структурой данных, включив их в один модуль (файл). К основным операциям (см. таблицу) добавить операцию, показывающую содержимое структуры после выполнения какого-либо действия с ней. Эту операцию реализовать на основе базовых операций:

Д) Основные операции над статической очередью б) линейным массивом с одним указателем на конец, вторая позиция очереди фиксируется.

# 2. Анализ задачи

Дано:

Структура очереди. Указатель на конец очереди(на первое свободное место) и   
линейный массив.

Последовательность символов.

Результат:

Последовательность символов.

## Метод решения

Для работы с очередью введем 5 основных подзадач.

## Выделение основных подзадач

1. Очистка очереди
2. Проверка очереди на пустоту
3. Добавление элемента в очередь
4. Взятие элемента из очереди
5. Просмотр очереди

1) Очистка очереди. Так как очередь задана линейным массивом с указателем на конец, то достаточно положить конец на начало очереди(массива), т.е. на первый элемент.

2) Проверка очереди на пустоту. Так как мы имеем указатель на конец очереди, нужно проверить, стоит ли указатель на первом элементе очереди(массива). Если да, то очередь пуста, иначе не пуста.  
3) Добавление элемента в очередь. В очередь элементы добавляются в конец. Так как мы имеем указатель на конец, то элемент добавляется на место указателя, а сам указатель передвигается на следующее место.

4) Взятие элемента из очереди. В очереди элементы берутся из начала. Берется первый элемент в очереди, потом каждый элемент передвигается на место предыдущего, как и указатель. После чего взятый элемент распечатывается.  
5) Просмотр очереди. Создается новая очередь, которая равна старой. После чего выполняется пункт 4) до тех пор, пока новая очередь не закончится.

# 3. Структуры данных, используемые для представления исходных данных и результатов задачи

## Внешнее представление входных данных

Символы, ввод которых влечет за собой выполнение определенных действий.

## Внешнее представление выходных данных

Последовательность символов, выводимых на экране в ходе работы программы.

# Внутреннее представление входных данных

Char – тип символов.

## Внутреннее представление данных

int k, showmenu = 1, exitFlag = 0, b; где k – 1 элемент очереди, берущийся в ходе выполнения подзадачи 4); showmenu – целое число которое в случае =1, выводит на экран меню вызовов операций;  
exitFlag=0 – целое число, которое в случае=1, заканчивает работу программы;  
b – целое число, которое вводится в очередь при выполнении подзадачи 3).

# 4. Укрупненный алгоритм решения задачи

# Структура программы

## Взаимосвязь функций

Pusta(queue \*q); void Ochistka(queue \*q);

int input(int x, queue \*q);

int output(queue \*q, int &num);

int show(queue q);

Void main()

Output(queue \*q, int &num);

Ochistka(queue \*q);  
input(int x, queue \*q);

Pusta(queue \*q);   
output(queue \*q, int &num);

Show(queue q)

## Составные части программы

Наименование функции:   
Pusta  
Прототип функции:  
int Pusta(queue \*q);  
Данная подпрограмма проверяет очередь на пустату, и возвращает 1 или 0, если список пуст и не пуст соответственно.

Наименование функции:  
Ochistka  
Прототип функции:  
void Ochistka(queue \*q);  
Данная подпрограмма очищает список, положив указатель на начало массива.

Наименование функции:  
input  
Прототип функции:  
int input(int x, queue \*q);   
Данная подпрограмма добавляет указанное число в конец очереди. И возвращает 1 или 0, если добавить удалось или нет соответственно.

Наименование функции:  
output  
Прототип функции:  
int output(queue \*q, int &num);  
Данная подпрограмма берет число из начала очереди(num), сдвигая другие элементы. Возвращает 1 или 0, если взять удалось или нет соответственно.

Наименование функции:  
show  
Прототип функции:  
int show(queue \*q)  
Данная подпрограмма показывает очередь, посредством применения базовых операций на новой, равной старой очереди. Возвращает 1 или 0, если очередь не пуста или пуста соответственно.

# Текст программы

/\* Файл Operations.h \*/

#ifndef OPERATIONS\_H

#define OPERATIONS\_H

#define N 5

struct queue { int end; int data[N]; };

int Pusta(queue \*q);

void Ochistka(queue \*q);

int input(int x, queue \*q);

int output(queue \*q, int &num);

int show(queue q);

/\* Файл Operations.cpp \*/

#include <stdio.h>

#include <stdlib.h>

#include "Operations.h"

int Pusta(queue \*q) {

if (q->end == 0) return 1;

else return 0;

}

void Ochistka(queue \*q){

q->end=0;

}

int input(int x, queue \*q){

if (q->end == N) return 0;

else {

q->data[q->end] = x; q->end++;

return 1;

}

}

int output(queue \*q, int &num){

int i, r, y;

if (q->end == 0) return 0;

else {

num= q->data[0];

r = q->end;

Ochistka(q);

for (i = 1; i < r; i++) y=input(q->data[i], q); return 1;

}

}

int show(queue q){

queue w;

int numr, y;

w = q;

if (Pusta(&w) == 1) return 0;

else{

while (Pusta(&w) == 0) { y=output(&w, numr); printf("%d ", numr); }

printf("\n");

return 1;

}

}

/\* Файл main.cpp \*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <locale.h>

#include <stdio.h>

#include <conio.h>

#include "Operations.h"

void main(){

setlocale(LC\_CTYPE, "Russian");

int k, showmenu = 1, exitFlag = 0, b;

char a;

queue s;

Ochistka(&s);

do {

if (showmenu){

printf("<1> - поместить элемент в очередь\n");

printf("<2> - извлечь элемент из очереди\n");

printf("<3> - очистить очередь\n");

printf("<4> - пуста ли очередь?\n");

printf("<5> - показать содержимое очереди\n");

printf("<m> - показать меню\n");

printf("<e> - завершение программы\n");

}

int repeatFlag = 0;

showmenu = 0;

do{

a = \_getch();

switch (a)

{

case '1':

printf("Введите элемент:\n");

scanf\_s("%d", &b);

if (input(b, &s) == 0) printf("Не удалось поместить элемент в очередь\n");

else printf("Элемент помещен в очередь: %d\n", b);

break;

case '2':

if (output(&s, k) == 0) printf("Не удалось взять элемент из очереди\n");

else printf("Взят элемент: '%d'\n", k);

break;

case '3': {

Ochistka(&s);

printf("Очередь очищена\n"); }

break;

case '4': {

if (Pusta(&s) == 1) printf("Очередь пуста\n");

else printf("Очередь не пуста\n");

} break;

case '5':

if (show(s) == 0) printf("Очередь пуста\n");

break;

case 'm':

showmenu = 1;

repeatFlag = 1;

break;

case 'e':

exitFlag = 1;

break;

default: printf("ОШИБКА: неверный номер команды\n");

repeatFlag = 1;

}

} while (!repeatFlag && !exitFlag);

} while (!exitFlag);

}

# 7.Тесты

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Результат | Примечания |
| 1 | 4 | Очередь пуста | Проверка на первоначальную пустоту очереди |
| 2 | 5 | Очередь пуста | Применили команду «показать очередь» |
| 3 | 2 | Не удалось взять элемент из очереди | Попробовали взять элемент из очереди |
| 4 | 1  4 | Элемент помещен в очередь: 4 | Поместили элемент в очередь |
| 5 | 4 | Очередь не пуста | Проверка на пустоту |
| 6 | 5 | 4 | Применили команду «показать очередь» |
| 7 | 3 | Очередь очищена | Применили команду «очистить очередь» |
| 8 | 5 | Очередь пуста | Применили команду «показать очередь» |
| 9 | 1  7  1  4  1  9 | Элемент помещен в очередь: … | Поместили элементы в очередь |
| 10 | 4 | Очередь не пуста | Проверка на пустоту |
| 11 | 5 | 7 4 9 | Применили команду «показать очередь» |
| 12 | 2 | Взят элемент: «7» | Извлекли элемент из очереди |
| 13 | 5 | 4 9 | Применили команду «показать очередь» |
| 14 | 1  2  1  8  1  5 | Элемент помещен в очередь: … | Поместили элементы в очередь. Сейчас очередь полна. |
| 15 | 5 | 4 9 2 8 5 | Применили команду «показать очередь» |
| 16 | 1  0 | Не удалось поместить элемент в очередь. | Попробовали поместить элемент в полную очередь. |

# 8. Результат работы программы

-Программа выдала верное решение на всех тестах и, следовательно, правильно работает.