Практическое занятие 15
«Динамические структуры данных. Очередь»

Краткие теоретические сведения

Очередь – структура данных типа «список», позволяющая добавлять элементы лишь в конец списка, и извлекать их из его начала. Она функционирует по принципу FIFO (First In, First Out — «первым пришёл — первым вышел»), для которого характерно, что все элементы a_1, a_2, …, a_{n-1}, a_n, добавленные раньше элемента a_{n+1}, должны быть удалены прежде, чем будет удален элемент a_{n+1}. Также очередь может быть определена как частный случай односвязного списка, который обслуживает элементы в порядке их поступления. Как и в «живой» очереди, здесь первым будет обслужен тот, кто пришел первым (рис. 1).

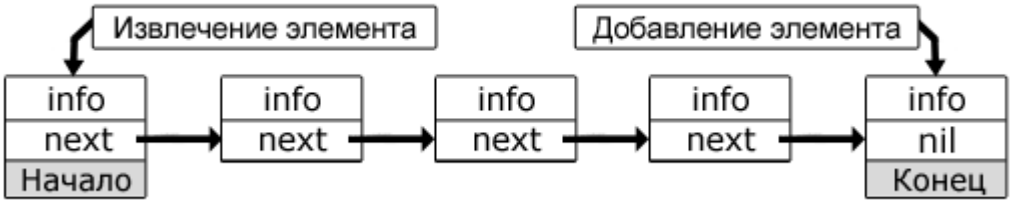

Рисунок 1. Представление очереди в памяти

Стандартный набор операций, выполняемых над очередями:

- добавление элемента;
- удаление элемента;
- чтение первого элемента.

Операции добавления или удаления элемента должны быть применены так, как это регламентировано в определении этой структуры данных, т. е. добавление – в конец, удаление – из начала.

Выделяют два способа программной реализации очереди. Первый из них основан на базе массива, а второй на базе указателей (связного списка). Первый способ – статический, т. к. очередь представляется в виде простого статического массива, второй – динамический.

Реализация очереди с помощью массива (статический способ)

Данный способ позволяет организовать и впоследствии обрабатывать очередь, имеющую фиксированный размер. Определим список операций, который будет использоваться как при реализации статической очереди, так и динамической:

- Creation(Q) – создание очереди Q;
- Full(Q) – проверка очереди Q на пустоту;
- Add(Q) – добавление элемента в очередь Q (его значение задается из функции);

- Delete(Q) – удаление элемента из очереди Q;
- Top(Q) – вывод начального элемента очереди Q;
- Size(Q) – размер очереди Q.

В программе каждая из этих операций оформлена в виде отдельной функции. Помимо того, потребуется описать массив данных **data[N]**, по сути, являющийся хранилищем данных вместимостью N, а также указатель на конец очереди (на ту позицию, в которую будет добавлен очередной элемент) – **last**. Изначально last равен 0.

```
#include "pch.h"
#include <iostream>
using namespace std;
const int N = 4; //размер очереди
struct Queue
{
    int data[N]; //массив данных
    int last; //указатель на начало
};
void Creation(Queue *Q) //создание очереди
{
    Q->last = 0;
}
bool Full(Queue *Q) //проверка очереди на пустоту
{
    if (Q->last == 0) return true;
    else return false;
}
void Add(Queue *Q) //добавление элемента
{
    if (Q->last == N)
    {
        cout << "\nОчередь заполнена\n\n"; return;
    }
    int value;
    cout << "\nЗначение > "; cin >> value;
    Q->data[Q->last++] = value;
    cout << endl << "Элемент добавлен в очередь\n\n";
}
void Delete(Queue *Q) //удаление элемента
{
    for (int i = 0; i < Q->last && i < N; i++) //смещение элементов
        Q->data[i] = Q->data[i + 1]; Q->last--;
}
int Top(Queue *Q) //вывод начального элемента
{
    return Q->data[0];
}
int Size(Queue *Q) //размер очереди
{
    return Q->last;
}
void main() //главная функция
{
    setlocale(LC_ALL, "Rus");
    Queue Q;
    Creation(&Q);
    char number;
    do
    {
        cout << "1. Добавить элемент" << endl;
        cout << "2. Удалить элемент" << endl;
        cout << "3. Вывести верхний элемент" << endl;
        cout << "4. Узнать размер очереди" << endl;
        cout << "0. Выйти\n\n";
        cout << "Номер команды > "; cin >> number;
```

```
        switch (number)
        {
        case '1': Add(&Q);
            break;
            //--------------------------------------------
        case '2':
            if (Full(&Q)) cout << endl << "Очередь пуста\n\n";
            else
            {
                Delete(&Q);
                cout << endl << "Элемент удален из очереди\n\n";
            } break;
            //--------------------------------------------
        case '3':
            if (Full(&Q)) cout << endl << "Очередь пуста\n\n";
            else cout << "\nНачальный элемент: " << Top(&Q) << "\n\n";
            break;
            //--------------------------------------------
        case '4':
            if (Full(&Q)) cout << endl << "Очередь пуста\n\n";
            else cout << "\nРазмер очереди: " << Size(&Q) << "\n\n";
            break;
            //--------------------------------------------
        case '0': break;
        default: cout << endl << "Команда не определена\n\n";
            break;
        }
    } while (number != '0');
    system("pause");
}
```

В функции main, сразу после запуска программы, создается переменная Q структурного типа Queue, адрес которой будет посылаться в функцию (в зависимости от выбора операции) как фактический параметр. Функция Creation создает очередь, обнуляя указатель на последний элемент. Далее выполняется оператор цикла do…while (цикл с постусловием), выход из которого осуществляется только в том случае, если пользователь ввел 0 в качестве номера команды. В остальных случаях вызывается подпрограмма, соответствующая команде, либо выводится сообщение о том, что команда не определена.

Из всех подпрограмм особого внимания заслуживает функция Delete. Удаление элемента из очереди осуществляется путем сдвига всех элементов в начало, т. е. значения элементов переписываются: в data[0] записывается значение элемента data[1], в data[1] – data[2] и т.д.; указатель конца смещается на позицию назад. Получается, что эта операция требует линейного времени O(n), где n – размер очереди, в то время как остальные операции выполняются за константное время. Данная проблема поддается решению.

Реализация очереди с помощью циклического массива (статический способ)

Вместо «мигрирующей» очереди, наиболее приемлемо реализовать очередь на базе циклического массива. Здесь напрашивается аналогия с «живой» очередью: если в первом случае покупатели подходили к продавцу, то теперь продавец будет подходить к покупателям. В приведенной реализации очередь считалась заполненной тогда, когда указатель last находился над последней ячейкой, т. е. на расстоянии N элементов от начала. В циклическом варианте расширяется интерпретация определения позиции last относительно начала очереди.

Пусть на начало указывает переменная first. Представим массив в виде круга – замкнутой структуры. После последнего элемента идет первый, и поэтому можно говорить, что очередь заполнила весь массив, тогда когда ячейки с указателями last и first находятся радом, а именно за last следует first. Теперь удаление элемента из очереди осуществляется простым смещением указателя first на одну позицию вправо (по часовой); чтобы добавить элемент нужно записать его значение в ячейку last массива data и сместить указатель last на одну позицию правее. Чтобы не выйти за границы массива воспользуемся следующей формулой:

(A mod N) + 1

Здесь A – один из указателей, N – размер массива, а mod – операция взятия остатка от деления.

В циклической реализации, как и прежде, очередь не содержит элементов тогда, когда first и last указывают на одну и ту же ячейку. Но в таком случае возникает одно небольшое отличие этой реализации от предшествующей. Рассмотрим случай заполнения очереди, основанной на базе массива, размер которого 5:

Элементы	first	last
—	1	1
1	1	2
1, 2	1	3
1, 2, 3	1	4
1, 2, 3, 4	1	5

В левом столбце записаны произвольные значения элементов, а в двух других значения указателей при соответствующем состоянии очереди. Необходимо заметить, что в массив размером 5 удалось поместить только 4 элемента. Все дело в том, что еще один элемент требует смещения указателя last на позицию 1. Тогда last=first. Но именно эта ситуация является необходимым и достаточным условием отсутствия в очереди элементов. Следовательно, мы не можем хранить в массиве больше N-1 элементов.

```cpp
#include "pch.h"
#include <iostream>
using namespace std;
const int N = 6; //размер очереди
struct Queue
{
    int data[N]; //массив данных
    int first; //указатель на начало
    int last; //указатель на конец
};
void Creation(Queue *Q) //создание очереди
{
    Q->first = Q->last = 1;
}
bool Full(Queue *Q) //проверка очереди на пустоту
{
    if (Q->last == Q->first) return true;
    else return false;
}
void Add(Queue *Q) //добавление элемента
{
    int value;
    cout << "\nЗначение > "; cin >> value;
    if ((Q->last % (N - 1)) + 1 == Q->first)
        cout << "\nОчередь заполнена\n\n"; else {
        Q->data[Q->last] = value;
        Q->last = (Q->last % (N - 1)) + 1;
        cout << endl << "Элемент добавлен в очередь\n\n";
    }
```

```
} void Delete(Queue *Q) //удаление элемента
{ Q->first=(Q->first%(N-1))+1;
cout << endl << "Элемент удален из очереди\n\n"; }
int Top(Queue *Q) //вывод начального элемента
{ return Q->data[Q->first]; }
int Size(Queue *Q) //размер очереди
{
    if (Q->first > Q->last) return (N - 1) - (Q->first - Q->last);
    else return Q->last - Q->first;
}
void main() //главная функция
{
    setlocale(LC_ALL, "Rus");
    Queue Q;
    Creation(&Q);
    char number;
    do
    {
        cout << "1. Добавить элемент" << endl;
        cout << "2. Удалить элемент" << endl;
        cout << "3. Вывести верхний элемент" << endl;
        cout << "4. Узнать размер очереди" << endl;
        cout << "0. Выйти\n\n";
        cout << "Номер команды > "; cin >> number;
       switch (number)
       {
       case '1': Add(&Q);
           break;
           //-------------------------------------------------
       case '2':
           if (Full(&Q)) cout << endl << "Очередь пуста\n\n";
           else Delete(&Q);
           break;
           //-------------------------------------------------
       case '3':
           if (Full(&Q)) cout << endl << "Очередь пуста\n\n";
           else cout << "\nНачальный элемент: " << Top(&Q) << "\n\n";
           break;
           //-------------------------------------------------
       case '4':
           if (Full(&Q)) cout << endl << "Очередь пуста\n\n";
           else cout << "\nРазмер очереди: " << Size(&Q) << "\n\n";
           break;
           //-------------------------------------------------
       case '0': break;
       default: cout << endl << "Команда не определена\n\n";
           break;
       }
    } while (number != '0');
    system("pause");
}
```

Таким образом, циклический вариант позволяет оптимизировать операцию Delete, которая прежде требовала линейного времени, а теперь выполняется за константное, независимо от длины очереди. Тем не менее, реализация очереди на базе массива имеет один существенный недостаток:

размер очереди статичен, поскольку зависит от размера массива. Реализация очереди на базе связного списка позволит обойти эту проблему.

Реализация очереди с помощью указателей (динамический способ)

Данный способ предполагает работу с динамической памятью. Для представления очереди используется односвязный список, в конец которого помещаются новые элементы, а старые извлекаются, соответственно, из начала списка. Здесь каждый узел списка имеет два поля: информационное и связующее:

```
struct Node
{
	int data;
	Node *next;
};
```

Также понадобится определить указатели на начало и конец очереди:

```
#include "pch.h"
#include <iostream>
using namespace std;
struct Node //описание узла списка
{
    int data; //информационное поле
    Node *next; //указатель на следующий элемент
};
struct Queue //описание очереди
{
    int size; //счетчик размера очереди
    Node *first; //указатель на начало очереди
    Node *last; //указатель на конец очереди
};
void Creation(Queue *Q) //создание очереди
{
    Q->first = new Node;
    Q->first->next = NULL;
    Q->last = Q->first;
    Q->size = 0;
}
bool Full(Queue *Q) //проверка очереди на пустоту
{
    if (Q->first == Q->last) return true;
    else return false;
}
int Top(Queue *Q) //вывод начального элемента
{
    return Q->first->next->data;
}
```

```
void Add(Queue *Q) //добавление элемента
{
    int value;
    cout << "\nЗначение > "; cin >> value;
    Q->last->next = new Node;
    Q->last = Q->last->next;
    Q->last->data = value; //добавление элемента в конец
    Q->last->next = NULL; //обнуление указателя на следующий элемент
    Q->size++;
    cout << "\nЭлемент добавлен\n\n";
}
void Delete(Queue *Q) //удаление элемента
{
    Q->first = Q->first->next; //смещение указателя
    Q->size--;
    cout << "\nЭлемент удален\n\n";
}
int Size(Queue *Q) //размер очереди
{
    return Q->size;
}
void main() //главная функция
{
    setlocale(LC_ALL, "Rus");
    Queue Q;
    Creation(&Q);
    char number;
    do
    {
        cout << "1. Добавить элемент" << endl;
        cout << "2. Удалить элемент" << endl;
        cout << "3. Вывести верхний элемент" << endl;
        cout << "4. Узнать размер очереди" << endl;
        cout << "0. Выйти\n\n";
        cout << "Номер команды > "; cin >> number;
        switch (number)
        {
        case '1': Add(&Q);
            break;
            //-------------------------------------------------
        case '2':
            if (Full(&Q)) cout << endl << "Очередь пуста\n\n";
            else Delete(&Q);
            break;
            //-------------------------------------------------
        case '3':
            if (Full(&Q)) cout << endl << "Очередь пуста\n\n";
            else { cout << "\nНачальный элемент: " << Top(&Q) << "\n\n"; }
            break;
            //-------------------------------------------------
        case '4':
            if (Full(&Q)) cout << endl << "Очередь пуста\n\n";
            else cout << "\nРазмер очереди: " << Size(&Q) << "\n\n";
            break;
            //-------------------------------------------------
        case '0': break;
        default: cout << endl << "Команда не определена\n\n";
            break;
```

```
        }
    } while (number != '0');
    system("pause");
}
```

Как отмечалось, этот способ позволяет не заботиться о месте, отводимом под рассматриваемую структуру данных – ее объем ограничен лишь памятью компьютера. Тем не менее, к числу недостатков данной реализации, в сравнении с предыдущей, можно отнести увеличение времени обработки и количества памяти.

Индивидуальные задания

Создать однонаправленную очередь с числами в диапазоне от –50 *до* +50. *После создания очереди выполнить индивидуальное задание. В конце работы все очереди должны быть удалены.*

1. Удалить из очереди все четные числа.
2. Удалить из очереди все отрицательные числа.
3. Поменять местами крайние элементы очереди.
4. Поменять местами минимальный и максимальный элементы очереди.
5. Удалить из очереди каждый второй элемент.
6. Удалить из очереди все элементы, расположенные до минимального элемента очереди.
7. Поменять местами наибольший среди отрицательных и наименьший среди положительных элементов очереди.
8. Поместить максимальный элемент очереди на первую позицию.
9. Поменять местами минимальный и первый элементы очереди.
10. Поменять местами первый и последний элементы очереди.
11. Удалить первый и последний элементы очереди.
12. Удалить из очереди все элементы, расположенные между минимальным и максимальным элементами очереди.
13. Удалить из очереди все элементы, стоящие после максимального элемента.
14. Найти среднее значение всех элементов очереди и удалить все элементы, которые меньше среднего значения.
15. Найти среднее значение всех элементов очереди. Поместить ближайший к среднему значению элемент очереди на первую позицию.