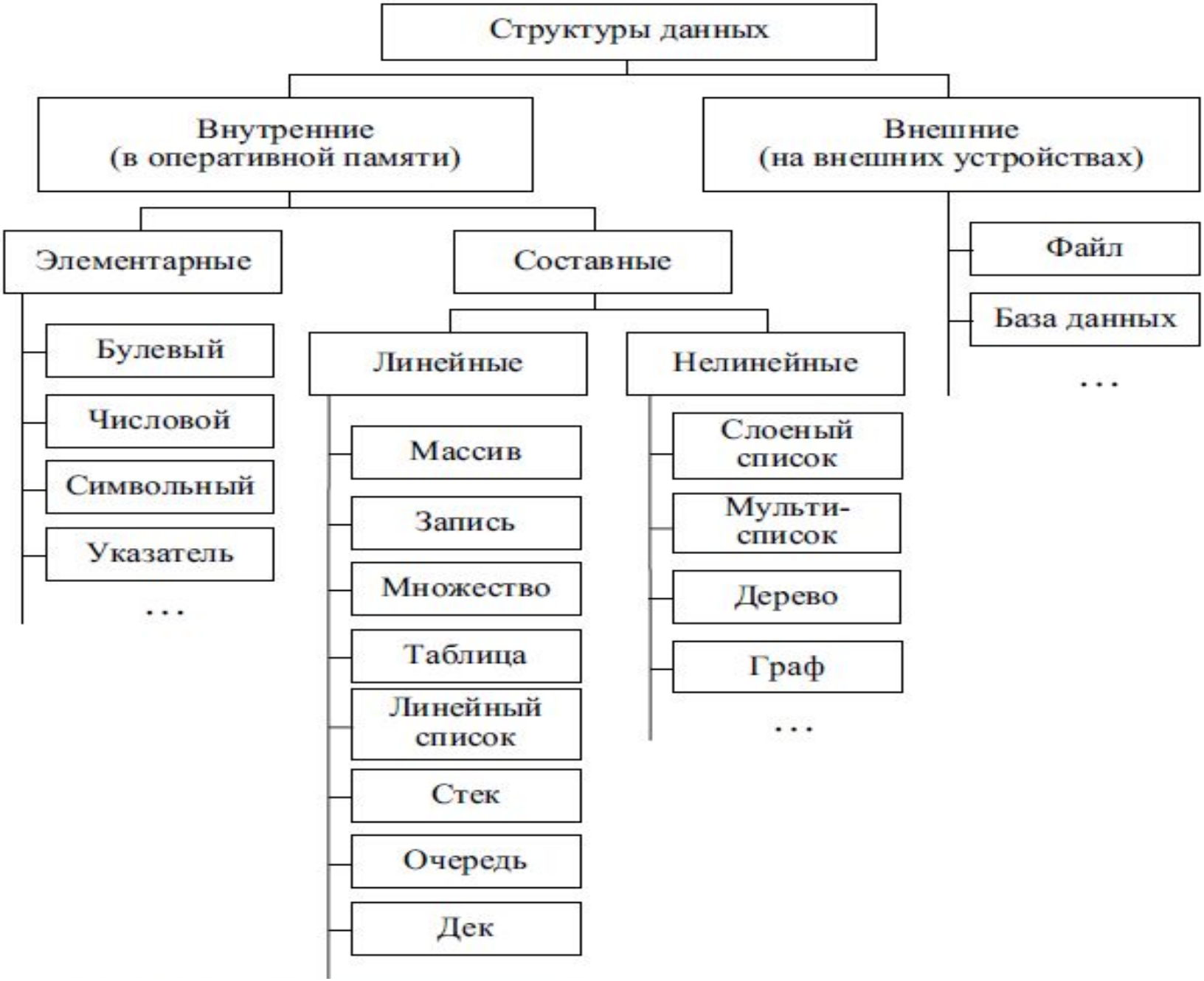
Алгоритмы и структуры данных

Лекция 2. Линейные структуры данных

Преподаватель: Тазиева Рамиля Фаридовна


Структуры данных
Внутренние (в оперативной памяти)
Внешние (на внешних устройствах)
Элементарные
Составные
Файл
База данных
...
Булевый
Числовой
Символьный
Указатель
...
Линейные
Нелинейные
Массив
Запись
Множество
Таблица
Линейный список
Стек
Очередь
Дек
Слоеный список
Мульти-список
Дерево
Граф
...

Линейные структуры данных

Массив - это поименованная совокупность однотипных элементов, упорядоченных по индексам, определяющих положение элементов в массиве.

Строка - это последовательность символов.

Структуры (запись)– это агрегат, составляющие которого (поля и функции) могут иметь имя и могут быть различного типа.

Множество (enum или перечисление) - это совокупность каких-либо однородных элементов, объединенных общим признаком и представляемых как единое целое.

Таблица – представляет сбой одномерный массив, элементами которого являются записи.

Ключ таблицы – поле, значение которого может быть использовано, для однозначной идентификации каждой записи в таблице.

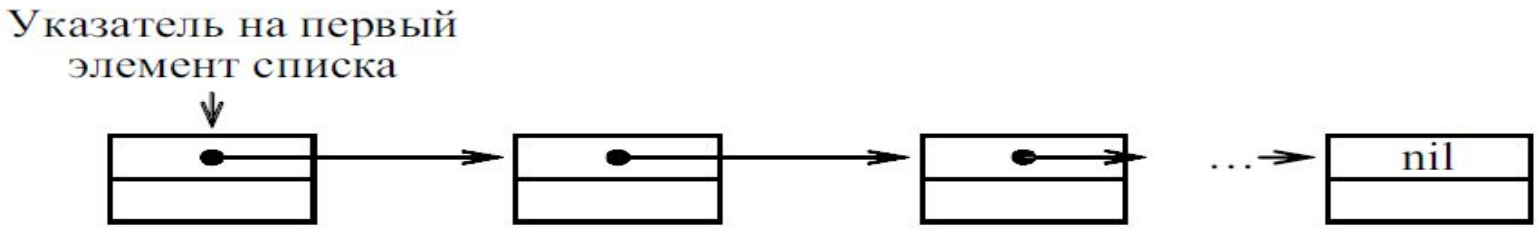
```
enum Operation
{   Add = 1,
    Subtract,
    Multiply,
    Divide
}
```

```
class Program
{    static void Main(string[]
args)   {
        Operation op;
        op = Operation.Add;
        Console.WriteLine(op);
// Add
  } }
```

СПИСОК

Список- это структура данных, представляющая собой логически связанную последовательность элементов.

Линейный однонаправленный список- любой элемент хранит собственно данные, а также ссылку указывающую на следующий элемент в списке или является пустым у последнего элемента.



Основные операции:

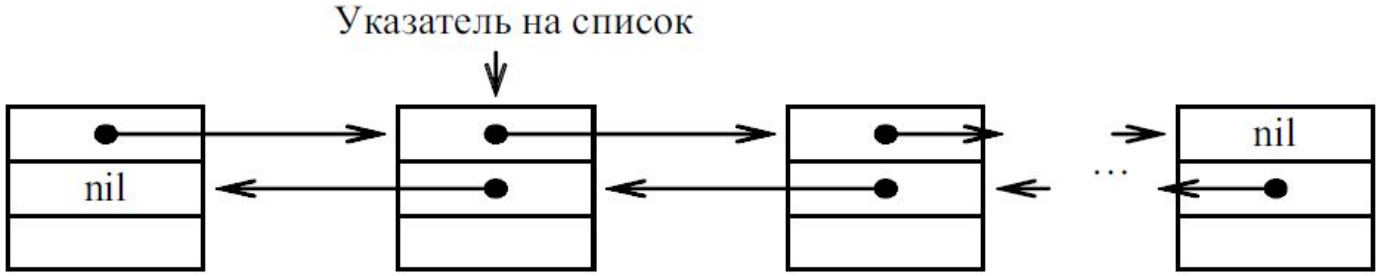
- ✔ Вставка элемента
- ✔ Просмотр
- ✔ Поиск
- ✔ Удаление элемента.

Примечание:
При выполнении операций с линейным однонаправленным списком необходимо обеспечить позиционирование какого либо указателя на первый элемент. В противном случае часть или весь список будет недоступен.

Линейный однонаправленный список имеет только один указатель на элемент, это позволяет минимизировать расход памяти на организацию такого списка.
Переход элементов только в одном направлении увеличивает время затрачиваемое на его обработку, т.к. обработка предыдущего элемента требует просмотра сначала списка.

Линейный двунаправленный список

Линейный двунаправленный список- любой элемент хранит собственно данные, а также две ссылки указывающую на предыдущий и следующий элемент в списке или является пустым у первого и последнего элемента соответственно.



Основные операции:

- ✔ Вставка элемента
- ✔ Просмотр
- ✔ Поиск
- ✔ Удаление элемента.

Примечание:
Нет необходимости обеспечивать позиционирование на первый элемент, т.к. благодаря двум указателям можно получить доступ к любому элементу, осуществляя переходы в прямом и обратном направлении.

Наличие двух указателей, позволяет ускорить операции связанные с передвижением по списку. Однако элементы списка за счет дополнительного поля занимают больший объем памяти. Кроме того операции вставки и удаления усложняются за счет необходимости манипулирования большим числом указателей.

Класс LinkedList<T>

Каждый узел представляет объект класса LinkedListNode<T>. Этот класс имеет следующие свойства:

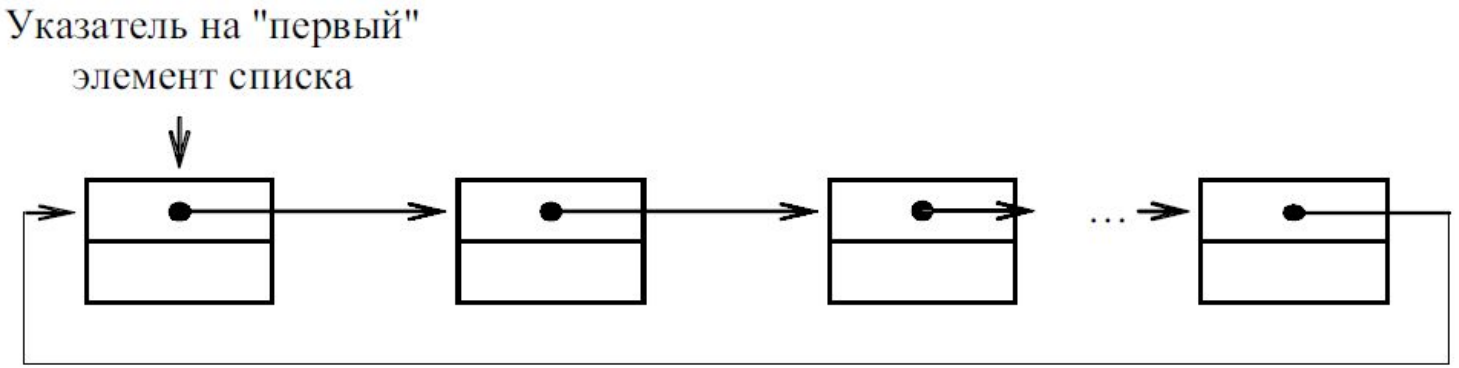
- **Value**: само значение узла, представленное типом T
- **Next**: ссылка на следующий элемент типа LinkedListNode<T> в списке. Если следующий элемент отсутствует, то имеет значение null.
- **Previous**: ссылка на предыдущий элемент типа LinkedListNode<T> в списке. Если предыдущий элемент отсутствует, то имеет значение null.

Методы класса LinkedList<T>:

AddAfter(LinkedListNode<T> node, LinkedListNode<T> newNode): вставляет узел newNode в список после узла node.
AddAfter(LinkedListNode<T> node, T value): вставляет в список новый узел со значением value после узла node.
AddBefore(LinkedListNode<T> node, LinkedListNode<T> newNode): вставляет в список узел newNode перед узлом node.
AddBefore(LinkedListNode<T> node, T value): вставляет в список новый узел со значением value перед узлом node.
AddFirst(LinkedListNode<T> node): вставляет новый узел в начало списка
AddFirst(T value): вставляет новый узел со значением value в начало списка
AddLast(LinkedListNode<T> node): вставляет новый узел в конец списка
AddLast(T value): вставляет новый узел со значением value в конец списка
RemoveFirst(): удаляет первый узел из списка. После этого новым первым узлом становится узел, следующий за удаленным
RemoveLast(): удаляет последний узел из списка

Циклический однонаправленный список

Циклический однонаправленный список – последний элемент содержит указатель, связывающий его с первым элементом.



Основные операции:

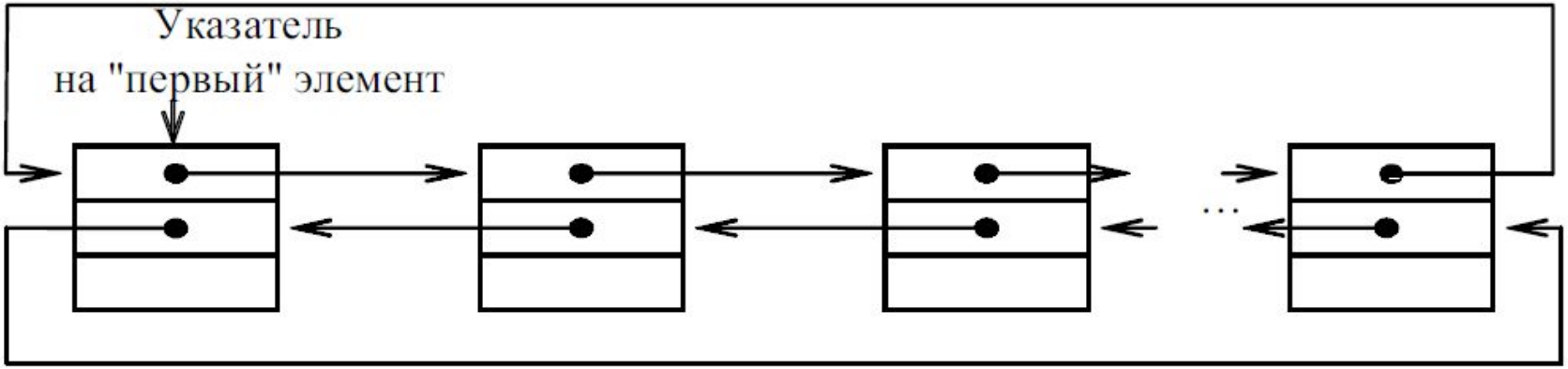
- ✔ Вставка элемента
- ✔ Просмотр
- ✔ Поиск
- ✔ Удаление элемента.

Примечание:
В случае удаления первого элемента, следует указатель переместить на следующий элемент.

Циклический однонаправленный список имеет только один указатель на элемент, это позволяет минимизировать расход памяти на организацию такого списка.
Переход элементов только в одном направлении увеличивает время затрачиваемое на его обработку.

Циклический двунаправленный список

Циклический двунаправленный список – имеет два указателя, один из которых указывает на следующий элемент в списке, а второй указывает на предыдущий элемент.



Основные операции:

- ✔ Вставка элемента
- ✔ Просмотр
- ✔ Поиск
- ✔ Удаление элемента.

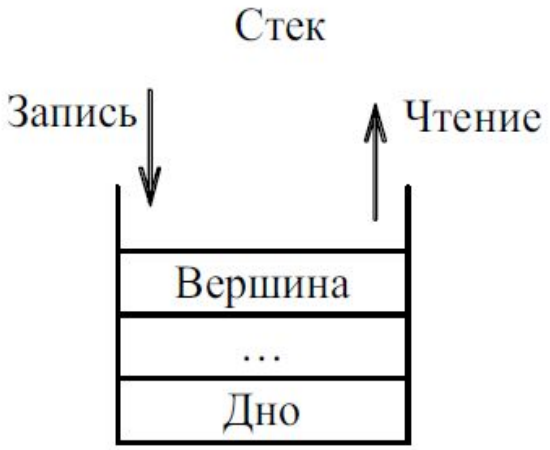
Примечание:
При неправильном переопределении указателей возможен разрыв списка ли потеря указателя на первый элемент, что приводит к потере доступа к части или всему списку.

Использование двух указателей, позволяет ускорить операции связанные с передвижением по списку. Однако элементы списка за счет дополнительного поля занимают больший объем памяти. Кроме того операции вставки и удаления осуществляются проще, чем в линейном двунаправленном списке, но сложнее, чем циклическом однонаправленном списке.

Стек. Принцип LIFO

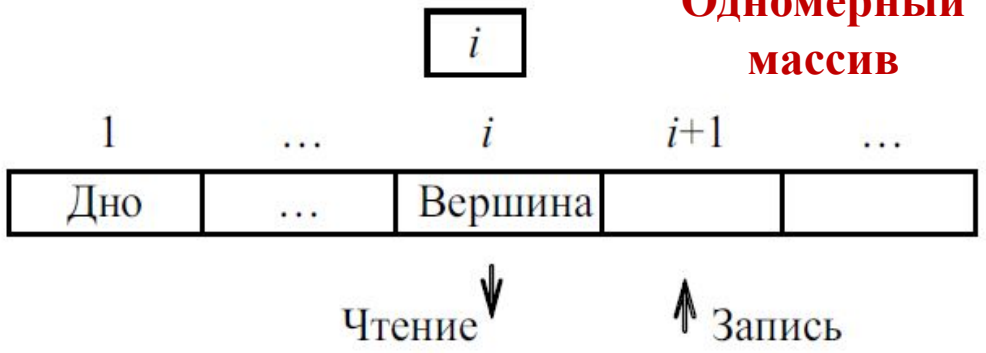
Стек – это структура данных, в которой новый элемент всегда записывается в ее начало (вершину) и очередной читаемый элемент также всегда выбирается из ее начала.

«последний пришел – первый вышел»



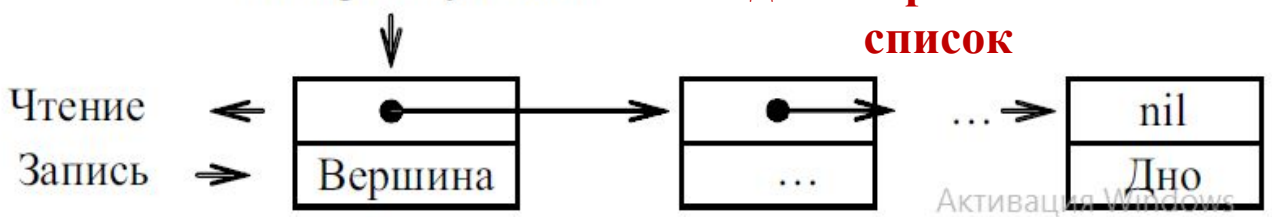
Статическая реализация

Одномерный массив



Динамическая реализация

Линейный однонаправленный список



Основные операции:

- ✔ Записать (Push)
- ✔ Прочитать (Pop)
- ✔ Очистить
- ✔ Проверка пустоты стека.

Активация Windows

Стек Stack<T>

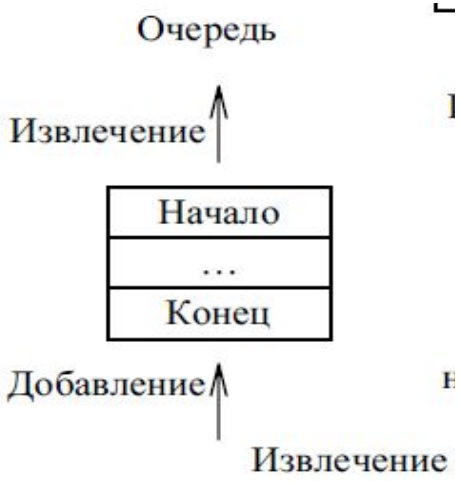
Методы класса Stack<T>:
Push: добавляет элемент в стек на первое место.
Pop: извлекает и возвращает первый элемент из стека.
Peek: просто возвращает первый элемент из стека без его удаления.

```
using System;
using System.Collections.Generic;
    class Program
    {
        static void Main(string[] args)
        {
            Stack<int> numbers = new Stack<int>();

            numbers.Push(3); // в стеке 3
            numbers.Push(5); // в стеке 5, 3
            numbers.Push(8); // в стеке 8, 5, 3

            // так как вверху стека будет находиться число 8, то оно и извлекается
            int stackElement = numbers.Pop(); // в стеке 5, 3
            Console.WriteLine(stackElement);
}}
```

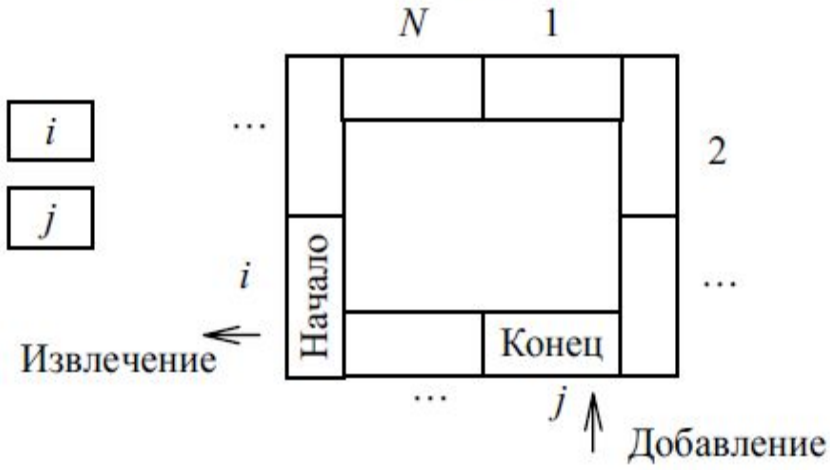
Очередь. FIFO

Очередь– это структура данных, представляющая собой последовательность элементов, образованных в порядке их поступления. Каждый элемент размещается в конце очереди; элемент, стоящий вначале очереди, выбирается из нее первым.

«первый пришел – первый вышел»

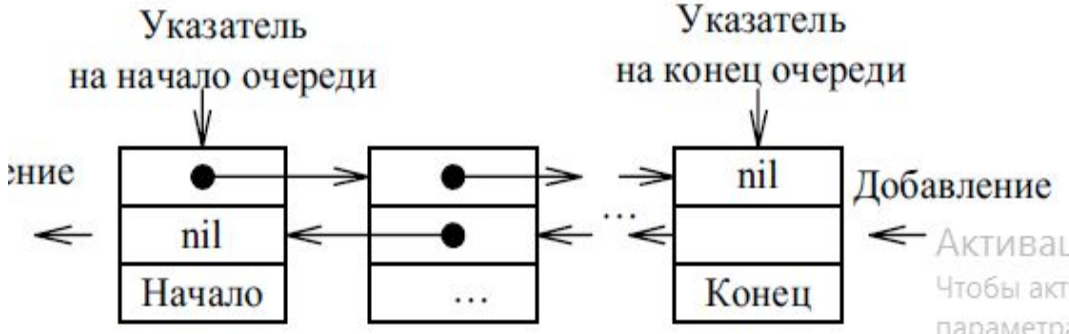


Статическая реализация



Динамическая реализация

Линейный двунаправленный список



Основные операции:

- ✔ Добавление.
- ✔ Извлечение.
- ✔ Очистка.
- ✔ Проверка пустоты очереди.

Очередь Queue<T>

Методы класса Queue<T> :

- **Dequeue**: извлекает и возвращает первый элемент очереди
- **Enqueue**: добавляет элемент в конец очереди
- **Peek**: просто возвращает первый элемент из начала очереди без его удаления

```
using System;
using System.Collections.Generic;
    class Program
    {
        static void Main(string[] args)
        {
            Queue<int> numbers = new Queue<int>();

            numbers.Enqueue(3); // очередь 3
            numbers.Enqueue(5); // очередь 3, 5
            numbers.Enqueue(8); // очередь 3, 5, 8

            // получаем первый элемент очереди
            int queueElement = numbers.Dequeue(); //теперь очередь
5, 8
            Console.WriteLine(queueElement);}}
```

Дек

Дек – это структура данных, представляющая собой последовательность элементов, в которую можно добавлять и удалять элементы с двух сторон.

Основные операции:

- Добавление в начало и конец.
- Извлечение в начало и конец.
- Очистка.
- Проверка пустоты дека.