Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Чувашский государственный университет им. И.Н.Ульянова».
Кафедра вычислительной техники.
Предмет: Объектно-ориентированное программирование

# Лабораторная работа №1.
# Стеки. Очереди.

Вариант 5.

Выполнил: Васильев Егор Юрьевич
студент группы ИВТ-41-20
Проверил: Леонид. А.

Чебоксары, 2021.

**Цель работы:** ознакомление со способами реализации стеков и очередей и выполняемых над ними операций; получение практических навыков программирования задач с использованием стеков и очередей.

## Информация в соответствии с подготовкой к работе.

**п. 1 и п. 2** Стек с использованием последовательного распределения элементов

```
private List<Type> _stack = new List<Type>();
private int _top;

Ссылок: 1
public bool Empty => _top == -1;
Ссылок: 1
public Type Peek {
    get {
        if (_top < 0)
        {
            throw new InvalidOperationException("Stack is empty");
        }

        return _stack[_top];
    }
}
```

```
public void Push(Type element)
{
    _stack.Add(element);
    _top++;
}
```

```
public Type Pop()
{
    if (_top < 0)
    {
        throw new InvalidOperationException("Stack is empty");
    }

    var temp = _stack[_top];
    _stack.RemoveAt(_top);
    _top--;
    return temp;
}
```

```
public bool Contains(Type element)
{
    return _stack.Contains(element);
}
```

```
public void Clear()
{
    _stack.Clear();
    _top = -1;
}
```

Очередь с использованием последовательного распределения элементов

```
private List<Type> _queue = new List<Type>();
private int _top;
private int _front;

Ссылок: 2
public bool Empty => _top == -1;
Ссылок: 2
public Type Peek
{
    get
    {
        if (_top < 0)
        {
            throw new InvalidOperationException("Queue is empty");
        }

        return _queue[_front];
    }
}
```

```
public void Enqueue(Type element)
{
    _queue.Add(element);
    _top++;
}
```

```
public Type Dequeue()
{
    if (_top < 0)
    {
        throw new IndexOutOfRangeException("Queue is empty");
    }

    var temp = _queue[_front];
    _queue.RemoveAt(_front);
    _top--;
    return temp;
}
```

```
public bool Contains(Type element)
{
    return _queue.Contains(element);
}
```

```
public void Clear()
{
    _queue.Clear();
    _top = -1;
}
```

Стек с использованием связанного распределения элементов.

```
private Node _head;
private Node _peak;

Ссылок: 1
public bool Empty => _head == null;
Ссылок: 1
public Type Peek => (Type)_peak.data;

Ссылок: 3
public void Push(Type data)
{
    var element = new Node();
    element.data = data;

    if (_head == null)
    {
        _head = element;
        _peak = element;
    }
    else
    {
        _peak.next = element;
        _peak = element;
    }
}
```

```
public Type Pop()
{
    if (_head  == null&& _peak == null)
    {
        throw new InvalidOperationException("Stack is empty");
    }

    Node element = _head;

    if (_head == _peak)
    {
        _head = null;
        _peak = null;
        return (Type)element.data;
    }

    while (element != null)
    {
        if (element.next == _peak)
        {
            var temp = (Type)element.next.data;
            _peak = element;
            _peak.next = null;
            return temp;
        }
        element = element.next;
    }

    return (Type)element.data;
}
```

```
public bool Contains(Type element)
{
    var node = _head;

    while (node != null)
    {
        if (Compare(node.data, element) == 0)
        {
            return true;
        }
        node = node.next;
    }

    return false;
}
public void Clear()
{
    var node = _head;

    while (node.next != null)
    {
        Pop();
    }

    Pop();
}
```

Очередь с использованием связанного распределения элементов.

```
private Node _front;
private Node _peak;

Ссылок: 0
public bool Empty => _front == null;
Ссылок: 0
public Type Front => (Type)_front.data;

Ссылок: 0
public void Enqueue(Type data)
{
    var element = new Node();
    element.data = data;

    if (_front == null)
    {
        _front = element;
        _peak = element;
    }
    else
    {
        _peak.next = element;
        _peak = element;
    }
}
```

```
public Type Dequeue()
{
    if (_front == null && _peak == null)
    {
        throw new InvalidOperationException("Queue is empty");
    }

    Node element = _front;

    if (_front == _peak)
    {
        _front = null;
        _peak = null;
        return (Type)element.data;
    }

    var temp = _front;
    _front = _front.next;

    return (Type)temp.data;
}
```

```
public bool Contains(Type element)
{
    var node = _front;

    while (node != null)
    {
        if (Compare(node.data, element) == 0)
        {
            return true;
        }
        node = node.next;
    }

    return false;
}
```

```
public void Clear()
{
    var node = _front;

    while (node != null)
    {
        Dequeue();
        node = node.next;
    }

}
```

3. Метод поддержания в одном линейном массиве двух стеков.

```
private List<Type> _array = new List<Type>();
private int _firstTop;
private int _secondTop;

private readonly int _firstCapacity;
private readonly int _secondCapacity;
private readonly int _capacity;

Ссылок: 0
public int FirstTop => _firstTop;
Ссылок: 0
public int SecondTop => _secondTop;
```

```
public void FirstPush(Type element)
{
    if (_firstTop > _secondCapacity + 1)
    {
        throw new StackOverflowException("First stack is full");
    }

    _array[++_firstTop] = element;
}
```

```
public void SecondPush(Type element)
{
    if (_secondTop < _firstCapacity)
    {
        throw new StackOverflowException("Second stack is full");
    }

    _array[--_secondTop] = element;
}
```

```
public Type FirstPop()
{
    if (_firstTop < 0)
    {
        throw new InvalidOperationException("First stack is empty");
    }
    var temp = _array[_firstTop];
    _array[_firstTop--] = default(Type);
    return temp;
}
```

```
public Type SecondPop()
{
    if (_secondTop >= _capacity)
    {
        throw new InvalidOperationException("First stack is empty");
    }

    var temp = _array[_secondTop];
    _array[_secondTop++] = default(Type);
    return temp;
}
```

5. Стек на базе двух очередей.

```
private Queue<Type> _queue1 = new Queue<Type>();
private Queue<Type> _queue2 = new Queue<Type>();
private int _stackSize;

Ссылок: 2
public bool FirstEmpty => _queue1.Count == 0;
Ссылок: 0
public bool SecondEmpty => _queue2.Count == 0;
Ссылок: 0
public int Count => _stackSize;
```

```
public void Push(Type value)
{
    _stackSize++;
    _queue2.Enqueue(value);

    while (!FirstEmpty)
    {
        _queue2.Enqueue(_queue1.Dequeue());
    }

    var queue = _queue1;

    _queue1 = _queue2;
    _queue2 = queue;
}
```

```
public Type Pop()
{
    if (FirstEmpty) return default(Type);
    _stackSize--;
    return _queue1.Dequeue();
}
```

## Индивидуальное задание. (Java)

Преобразование префиксной формы записи выражения в постфиксную.

(код плохой, писал давно)

```java
public class ConvertByStack {
    String strPrefix;
    Stack stack = new Stack();

    public ConvertByStack(String strPrefix) {
        this.strPrefix = strPrefix;
    }

    public String toPostfix() {
        char[] prefix = strPrefix.toCharArray();
        String temp = "";

        for (int i = strPrefix.length() - 1; i >= 0; i--) {
            if (prefix[i] >= 97 && prefix[i] <= 122) {
                stack.push(prefix[i]);
            } else if (prefix[i] >= 42 && prefix[i] <= 47 && prefix[i] != 44 && prefix[i] != 46){
                for (int j = 0; j < 2; j++) {
                    temp += stack.pop();
                }
                temp += prefix[i];
                stack.push(temp);
                System.out.println(stack);
                temp = "";
            } else {
                System.out.println("Incorrect expression");
            }
        }
        return stack.toString();
    }
}
```

Результат выполнения программы.
Для выражения *-*-+ab*cde-fg+hi

```
[hi+]
[hi+, fg-]
[hi+, fg-, e, cd*]
[hi+, fg-, e, cd*, ab+]
[hi+, fg-, e, ab+cd*-]
[hi+, fg-, ab+cd*-e*]
[hi+, ab+cd*-e*fg--]
[ab+cd*-e*fg--hi+*]
```

**Вывод:** ознакомился со способами реализации стеков, очередей и выполняемыми над ними операциями. Получил практические навыки программирования задач с использованием стеков и очередей.