**Лабораторная работа №5**

**Технологии программирования**

**«Разработка консольных приложений на языке C#»**

Солонин Егор

А-14-19

Вариант 14

**Задание:**

1. Составить на языке C# описания классов для указанных объектов.
2. В среде Visual Studio разработать консольную программу, иллюстрирующую использование объектов заданных классов.

Индивидуальный вариант (№14): Очередь на основе массива (включение, исключение элемента)

**Описание работы программы:**

При открытии приложения пользователь имеет возможность выбрать один из пунктов меню:

1. Инициализация очереди
2. Вывод очереди на экран
3. Добавление элемента в конец очереди
4. Удаление элемента из начала очереди
5. Выйти из программы

Класс Queue имеет:

* \_array (закрытое поле) – массив, содержащий значения элементов очереди
* \_head (закрытое поле) – индекс в массиве \_array, указывающий на начало очереди
* \_tail (закрытое поле) – индекс в массиве \_array, указывающий на конец очереди
* Конструктор, который инициализирует новый массив \_array длины 5.
* Count (свойство) – позволяет получить или установить размер очереди
* ToArray (свойство) – позволяет получить очередь, записанную в массив
* Pop(метод) – сдвигает индекс начала очереди на 1 элемент, тем самым исключая его из очереди. Возвращает удаленный элемент.
* Push(метод) – сдвигает индекс конца очереди на 1 элемент, записывает новый элемент в конец очереди.
* SetCapacity (закрытый метод) – позволяет увеличить емкость массива, в котором хранится очередь.

При инициализации очереди пользователю предлагается вводить строковые значения. Ключевым словом прекращения ввода является “stop”. Алгоритм инициализации полностью основан на алгоритме добавления элемента в конец очереди.

**Алгоритмы операций на псевдокоде:**

Удаление(pop):

1. Если (очередь пуста) – выход. Сообщить пользователю, что очередь пуста.
2. Получить удаляемый элемент из массива по индексу начала очереди. Записать его в переменную deletedItem.
3. Записать в элемент в начале очереди пустую строку.
4. Изменить индекс начала очереди
5. Уменьшить размер очереди на 1
6. Вернуть удаленный элемент, хранящийся в переменной deletedItem

Добавление(push):

1. Если емкость массива равна количеству элементов очереди – удвоить емкость.
2. Записать по индексу, указывающий на конец очереди в массив элемент item, который хотим добавить
3. Изменить индекс конца очереди
4. Увеличить размер очереди на 1

**Тесты работы программы:**

|  |  |
| --- | --- |
| Инициализация очереди |  |
| Добавление элемента “sixth\_pushed” |  |
| Вывод очереди на экран. Добавленный элемент – в конце очереди |  |
| Удаление элемента. Последующий вывод на экран. |  |
| Попытка удалить элемент из пустой очереди |  |

**Листинг программы:**

using System;

using System.Diagnostics.Tracing;

using System.Threading;

namespace Labs

{

public class Queue

{

private string[] \_array;

private int \_head;

private int \_tail;

public Queue()

{

\_array = new string[5];

}

public int Count { get; private set; }

public string[] ToArray

{

get

{

var destinationArray = new string[Count];

Array.Copy(\_array, \_head, destinationArray, 0, Count);

return destinationArray;

}

}

public string Pop()

{

if (Count == 0)

return "\*\*FIFO IS EMPTY\*\*";

var deletedItem = \_array[\_head];

\_array[\_head] = "";

\_head = (\_head + 1) % \_array.Length;

Count--;

return deletedItem;

}

public void Push(string item)

{

if (Count == \_array.Length)

SetCapacity(\_array.Length \* 2);

\_array[\_tail] = item;

\_tail = (\_tail + 1) % \_array.Length;

Count++;

}

private void SetCapacity(int capacity)

{

var destinationArray = new string[capacity];

if (Count > 0)

{

if (\_head < \_tail)

{

Array.Copy(\_array, \_head, destinationArray, 0, Count);

}

else

{

Array.Copy(\_array, \_head, destinationArray, 0, \_array.Length - \_head);

Array.Copy(\_array, 0, destinationArray, \_array.Length - \_head, \_tail);

}

}

\_array = destinationArray;

\_head = 0;

\_tail = Count == capacity ? 0 : Count;

}

}

public class Menu

{

public void ShowMenu()

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("\n\*\*MENU\*\*\n");

Console.WriteLine("1. init new FIFO");

Console.WriteLine("2. print FIFO");

Console.WriteLine("3. push item");

Console.WriteLine("4. pop item");

Console.WriteLine("5. exit\n");

Console.ResetColor();

}

public void Init(Queue fifo)

{

Console.ForegroundColor = ConsoleColor.DarkGreen;

Console.WriteLine("\n\*\*CREATING NEW FIFO\*\*\n");

Console.WriteLine("Input items. To finish - input keyword stop \n");

while (true)

{

var item = Console.ReadLine();

if (item == "stop")

break;

fifo.Push(item);

}

Console.WriteLine("\nNew FIFO has been successfully created\n");

Thread.Sleep(3000);

Console.ResetColor();

}

public void Print(Queue fifo)

{

Console.ForegroundColor = ConsoleColor.DarkGreen;

Console.WriteLine("\n\*\*PRINT FIFO\*\*");

var size = fifo.Count;

if (size == 0)

{

Console.WriteLine("\n\*---EMPTY---\*");

Thread.Sleep(3000);

Console.ResetColor();

return;

}

var queue = fifo.ToArray;

Console.WriteLine("\n\*---HEAD---\*");

foreach (var item in queue)

Console.WriteLine(item);

Console.WriteLine("\*---TAIL---\*\n");

Thread.Sleep(3000);

Console.ResetColor();

}

public void Push(Queue fifo)

{

Console.ForegroundColor = ConsoleColor.DarkGreen;

Console.WriteLine("\n\*\*PUSH ITEM\*\*\n");

Console.WriteLine("Input item you wanna push\n");

var item = Console.ReadLine();

fifo.Push(item);

Console.WriteLine("\n\*\*SUCCESS\*\*\n");

Thread.Sleep(3000);

Console.ResetColor();

}

public void Pop(Queue fifo)

{

Console.ForegroundColor = ConsoleColor.DarkGreen;

var deletedItem = fifo.Pop();

Console.WriteLine(deletedItem == "\*\*FIFO IS EMPTY\*\*"

? deletedItem

: $"\n\*\*ITEM '{deletedItem}' HAS BEEN POPPED\*\*\n");

Thread.Sleep(3000);

Console.ResetColor();

}

}

internal class Program

{

private static void Main()

{

var fifo = new Queue();

var running = true;

while (running)

{

var menu = new Menu();

menu.ShowMenu();

var option = int.Parse(Console.ReadLine());

switch (option)

{

case 1:

menu.Init(fifo);

break;

case 2:

menu.Print(fifo);

break;

case 3:

menu.Push(fifo);

break;

case 4:

menu.Pop(fifo);

break;

case 5:

running = false;

break;

default:

Console.WriteLine("ERROR - INCORRECT OPTION");

break;

}

}

}

}

}