Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Фізико-технічний інститут

**ЛАБОРАТОРНА РОБОТА №4**

З предмету

«Алгоритми та структури даних»

Виконав:

Студент ФІ-23 групи

Булавінцев Юрій

Київ 2024

***Варіант № 1***

**Мета роботи**: отримати навики роботи зі стеком та чергою,

реалізованими у вигляді одновимірного масиву та зв’язного лінійного

списку.

***Завдання для лабораторної роботи №4***

1. Реалізувати чергу двома способами: масивом (використовувати кільцеву чергу , див. лекцію) та зв'язним списком.

**1. Вирішення**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab4\_2\_asd

{

public class Queue<T>

{

private LinkedList<T> list;

public Queue()

{

list = new LinkedList<T>();

}

public void Enqueue(T item)

{

list.AddLast(item);

}

public T Dequeue()

{

if (list.Count == 0)

{

Console.WriteLine("Черга пуста");

}

T item = list.First.Value;

list.RemoveFirst();

return item;

}

public T Peek()

{

if (list.Count == 0)

{

Console.WriteLine("Черга пуста");

}

return list.First.Value;

}

public bool IsEmpty()

{

return list.Count == 0;

}

public void Display()

{

foreach (var item in list)

{

Console.WriteLine(item);

}

}

}

public class Program

{

static void Main(string[] args)

{

Queue<int> queue = new Queue<int>();

queue.Enqueue(1);

queue.Enqueue(2);

queue.Enqueue(4);

queue.Display();

queue.Dequeue();

queue.Display();

Console.ReadLine();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab4\_asd

{

class Queue

{

private int amount\_el;

private int[] array;

private int front;

private int rear;

static void Main(string[] args)

{

Console.WriteLine("Введіть к-ть елементів у масиві: ");

int amount\_el = Int32.Parse(Console.ReadLine());

Queue queue = new Queue(amount\_el);

queue.enquence(10);

queue.enquence(5);

queue.enquence(3);

queue.enquence(7);

queue.display();

queue.dequence();

queue.dequence();

queue.enquence(100);

queue.dequence();

queue.display();

Console.ReadLine();

}

public Queue(int amount\_el)

{

this.amount\_el = amount\_el;

array = new int[amount\_el];

front = -1;

rear = -1;

}

bool isFull()

{

if (front == 0 && rear == amount\_el - 1)

{

return true;

}

else if (front == rear + 1)

{

return true;

}

else

{

return false;

}

}

bool isEmpty()

{

if (front == -1)

{

return true;

}

else

{

return false;

}

}

public void enquence(int element)

{

if (isFull())

{

Console.WriteLine("Черга заповнена: ");

}

else if (front == -1)

{

front = 0;

rear = (rear + 1) % amount\_el;

array[rear] = element;

Console.WriteLine("Елемент додано: " + element);

}

else

{

rear = (rear + 1) % amount\_el;

array[rear] = element;

Console.WriteLine("Елемент додано: " + element);

}

}

public int dequence()

{

if (isEmpty())

{

Console.WriteLine("Черга пуста ");

return (-1);

}

else

{

int element = array[front];

if (front == rear)

{

front = -1;

rear = -1;

}

else

{

front = (front + 1) % amount\_el;

}

return element;

}

}

public void display()

{

int i;

if (isEmpty())

{

Console.WriteLine("Пуста черга ");

}

else

{

Console.WriteLine("Показник Front: " + front);

//Console.WriteLine(array[front]);

Console.WriteLine("Елементи: ");

/\*int element = dequence();

if (element != -1)

{

Console.WriteLine("Видалений елемент:" + element);

}\*/

for (i = front; i != rear; i = (i + 1) % amount\_el)

{

Console.WriteLine(array[i] + " ");

}

Console.WriteLine(array[i]);

Console.WriteLine("Показник Rear: " + rear);

}

}

}

}

Додаткове

using System;

using System.Collections.Generic;

public class Program

{

public static bool TryConvertInfixToPostfixe(string expression, out string postfixExpression)

{

postfixExpression = InfixToPostfix(expression);

return postfixExpression != null;

}

public static int Main(string[] args)

{

Console.WriteLine("Please, write your example:");

string expression = Console.ReadLine();

//string postfixExpression = InfixToPostfix(expression);

if (TryConvertInfixToPostfixe(expression, out var postfixExpression))

{

int result = Evaluate(postfixExpression);

Console.WriteLine("Result: " + result);

Console.ReadLine();

return 0;

}

else

{

Console.WriteLine("Error");

Console.ReadLine();

return 1;

}

}

public static string InfixToPostfix(string expression)

{

int length = expression.Length;

Stack<string> operation = new Stack<string>();

Queue<string> numb\_oper = new Queue<string>();

int i = 0;

int number;

char bracket = '(';

char plus = '+';

char minus = '-';

char multiply = '\*';

char bracket\_r = ')';

while (i < length)

{

string currentobject = expression[i].ToString();

if (int.TryParse(currentobject, out number))

{

numb\_oper.Enqueue(currentobject);

}

else

{

if (operation.Count == 0 || operation.Peek() == bracket.ToString())

{

operation.Push(currentobject);

}

else

{

if ((operation.Peek() == plus.ToString() || operation.Peek() == minus.ToString()) && (currentobject == multiply.ToString()))

{

operation.Push(currentobject);

}

else if (currentobject == bracket.ToString())

{

operation.Push(currentobject);

}

else if (currentobject == bracket\_r.ToString())

{

while (operation.Peek() != bracket.ToString())

{

numb\_oper.Enqueue(operation.Pop());

}

operation.Pop();

}

else

{

while (operation.Count > 0 && operation.Peek() != bracket.ToString() && GetPriority(operation.Peek()) >= GetPriority(currentobject))

{

numb\_oper.Enqueue(operation.Pop());

}

operation.Push(currentobject);

}

}

}

i++;

}

while (operation.Count > 0)

{

numb\_oper.Enqueue(operation.Pop());

}

string postfixExpression = string.Join(" ", numb\_oper);

return postfixExpression;

}

public static int Evaluate(string expression)

{

string[] tokens = expression.Split(' ');

Stack<int> stack = new Stack<int>();

foreach (string token in tokens)

{

if (int.TryParse(token, out int number))

{

stack.Push(number);

}

else

{

int operand2 = stack.Pop();

int operand1 = stack.Pop();

switch (token)

{

case "+":

stack.Push(operand1 + operand2);

break;

case "-":

stack.Push(operand1 - operand2);

break;

case "\*":

stack.Push(operand1 \* operand2);

break;

default:

throw new ArgumentException("Invalid token: " + token);

}

}

}

return stack.Pop();

}

static int GetPriority(string op)

{

switch (op)

{

case "+":

case "-":

return 1;

case "\*":

return 2;

default:

return 0;

}

}

}