МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ижевский государственный технический университет имени М. Т. Калашникова»

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение»

Алгоритмы и структуры данных

Лабораторная работа №2

«Списки, стеки, очереди»

Вариант № 4

Выполнил:

Студент группы Б03-191-2з

Закиров Айдар Ринатович

Проверил:

Старший преподаватель

[Еланцев Михаил Олегович](https://istu.ru/staff/1092)

Ижевск 2020

**Задача:**

Реализовать две структуры данных: однонаправленный список и очередь на любом языке программирования.

Список должен содержать следующие операции:

1. Вставка элемента в конец (целочисленное значение) Вставка должна корректно обрабатывать случай, когда список пуст (указатель на первый узел пуст)
2. Вставка элемента перед заданным индексом
3. Получение значения элемента по индексу
4. Удаление элемента по значению В случае, если элемент с ключом не найден, функция должна вернуть false, иначе - true
5. Печать всех элементов списка
6. Освобождение памяти от структуры данных

Очередь должна содержать следующие операции:

1. Вставка элемента в конец (целочисленное значение) Вставка должна корректно обрабатывать случай, когда список пуст (указатель на первый узел пуст)
2. Вставка элемента перед заданным индексом
3. Получение значения элемента по индексу
4. Удаление элемента по значению В случае, если элемент с ключом не найден, функция должна вернуть false, иначе - true
5. Печать всех элементов списка
6. Освобождение памяти от структуры данных

В качестве ответа приложите файл с исходным текстом программы, контрольный пример и его текстовое описание.

1. **Однонаправленный список.**

**Решение:**

Для реализации структуры данных в виде однонаправленного списка была составлена программа на языке программирования C#:

**Исходный код программы:**

using System;

using System.Collections.Generic;

namespace UnidirectionalList

{

class Program

{

List<int> numList = new List<int>();

//добавление элемента

void AddToList(int num)

{

numList.Add(num);

}

//удаление элемента

void RemoveToList(int num)

{

int indexNum = numList.IndexOf(num);

if (indexNum < 0) Console.WriteLine("Элемент с таким значением не существует");

else numList.RemoveAt(indexNum);

}

//вставка элемента на позицию перед указанным индексом

void InsertBeforeId(int index, int num)

{

index--;

numList.Insert(index, num);

}

//получение значения элемента по индексу

int GetItemById (int index)

{

Console.WriteLine(numList[index]);

return numList[index];

}

//Печать всех элементов

void PrintAllElements()

{

foreach(int elem in numList)

{

Console.Write($"{elem}, ");

}

Console.WriteLine("");

}

// команды вызова методов

void Comands(string comand)

{

switch(comand)

{

case "Add":

Console.Write("Введите число которое нужно добавить: ");

int num = Convert.ToInt32(Console.ReadLine());

AddToList(num);

break;

case "Reemove":

Console.Write("Введите число которое нужно удалить: ");

int elem = Convert.ToInt32(Console.ReadLine());

RemoveToList(elem);

break;

case "InsertBefore":

Console.Write("Введите индекс перед которым нужно вставить число: ");

int index = Convert.ToInt32(Console.ReadLine());

Console.Write("Введите число: ");

int number = Convert.ToInt32(Console.ReadLine());

InsertBeforeId(index, number);

break;

case "Print":

PrintAllElements();

break;

case "GetToIndex":

Console.Write("Введите индекс получаемого элемента: ");

int indexElem = Convert.ToInt32(Console.ReadLine());

GetItemById(indexElem);

break;

case "ClearMemory":

Console.WriteLine(GC.GetTotalMemory(false));

GC.Collect(2, GCCollectionMode.Forced);

Console.WriteLine(GC.GetTotalMemory(true));

break;

default:

Console.Write("Введена несуществующая команда, попробуйте ещё раз!");

Help();

break;

}

}

void Help()

{

Console.WriteLine("Для того, что бы добавить элемент в список введите комманду \"Add\" \n" +

"Для того, что бы удалить элемент в списке введите комманду \"Reemove\"\n" +

"Для того, что бы вставить элемент перед нужным индексом введите комманду \"InsertBefore\"\n" +

"Для того, что бы получить элемент по индексу введите комманду \"GetToIndex\"\n" +

"Для того, что бы вывести на экран все элементы списка введите комманду \"Print\"\n" +

"Для очистки памяти введите команду \"ClearMemory\"\n");

}

static void Main(string[] args)

{

Program a = new Program();

a.Help();

string str;

do

{

str = Console.ReadLine();

a.Comands(str);

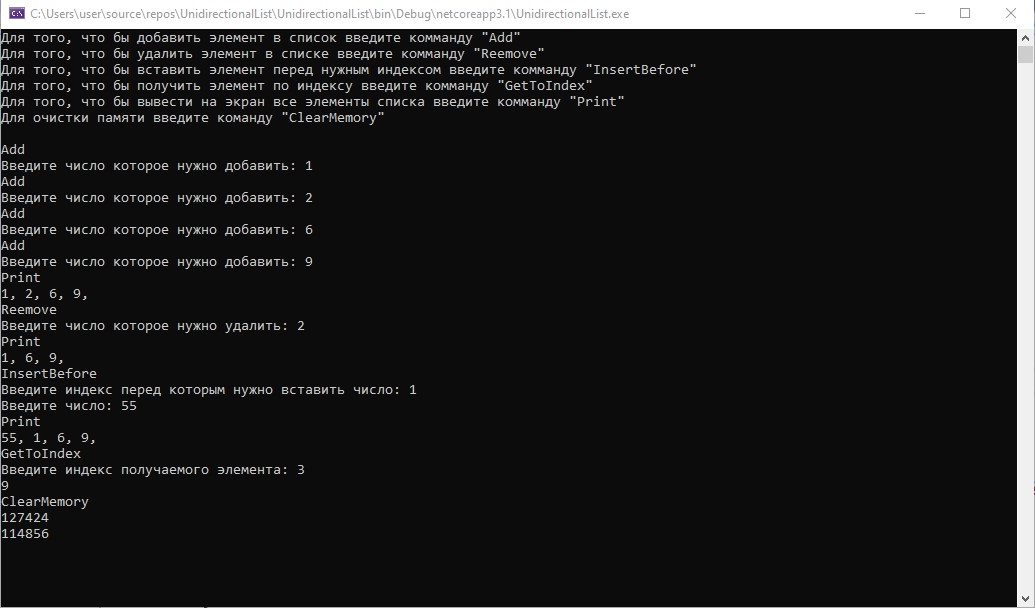
} while (str != null);

}

}

}

**Вывод программы:**



После запуска программа выводит список команд для проведения нужных операций со структурой данных.

Вводим целочисленные элементы 1, 2, 6, 9 в структуру используя команду «Add» которая запускает метод AddToList, осуществляющий добавление элемента в конец структуры.

Проверяем наличие элементов в списке вводя команду «Print», данная команда запускает метод PrintAllElements, который выводит в консоль все элементы нашей структуры.

С помощью команды «Reemove» вызываем метод ReemoveToList, который осуществляет поиск и удаление элемента с указанным нами значением из списка, в данном случае это элемент со значением 2. Затем проверяем наш список для того что бы убедится, что элемент удалён, получаем элементы 1, 6, 9.

Введя команду «InsertBefore» запускаем метод InsertBeforeId, вставляющий новый элемент перед указанным нами индексом элемента, в данном случае мы вставляем элемент со значением 55 перед элементом с индексом 1. Проверяем корректность выполнения команды используя команду «Print», получаем вывод элементов 55, 1, 6, 9.

Используя команду «GetToIndex» вызываем метод GetItemById. Данный метод выводит в консоль значение элемента, который мы ввели. Введя индекс 3, мы получили элемент со значение 9.

Команда «ClearMemory» запускает методы для сборщика мусора. Используя эти методы мы получаем вывод в консоль затраченную память до и после очистки.

1. **Сортировка слиянием.**

**Решение:**

Для реализации структуры данных в виде очереди была составлена программа на языке программирования C#:

**Исходный код программы:**

using System;

using System.Collections.Generic;

namespace Queue

{

class Program

{

static void AddToQueue(Queue<int> queue, int num)

{

queue.Enqueue(num);

}

static void ReemoveToQueue(Queue<int> queue)

{

if (queue.Count == 0) Console.WriteLine("В структуре нет элементов");

else

{

int num = queue.Dequeue();

Console.WriteLine($"Элемент со значением {num} удален из структуры");

}

}

static Queue<int> CloneQueue(Queue<int> queue)

{

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

return cloneNumbers;

}

static void PrintQueue(Queue<int> queue)

{

foreach (int elem in queue)

{

Console.Write($"{elem} ");

}

Console.WriteLine();

}

//метод очистки памяти (хотя и без него сборщик бы справился вполне себе:-) )

static void FreeingUpMemory()

{

Console.WriteLine(GC.GetTotalMemory(false));

GC.Collect();

Console.WriteLine(GC.GetTotalMemory(true));

}

static void Main(string[] args)

{

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

AddToQueue(numbers, 5);

AddToQueue(numbers, 7);

AddToQueue(numbers, 9);

PrintQueue(numbers);

ReemoveToQueue(numbers);

PrintQueue(numbers);

Queue<int> copyQueue = CloneQueue(numbers);

PrintQueue(copyQueue);

AddToQueue(copyQueue, 18);

PrintQueue(numbers);

PrintQueue(copyQueue);

numbers = null;

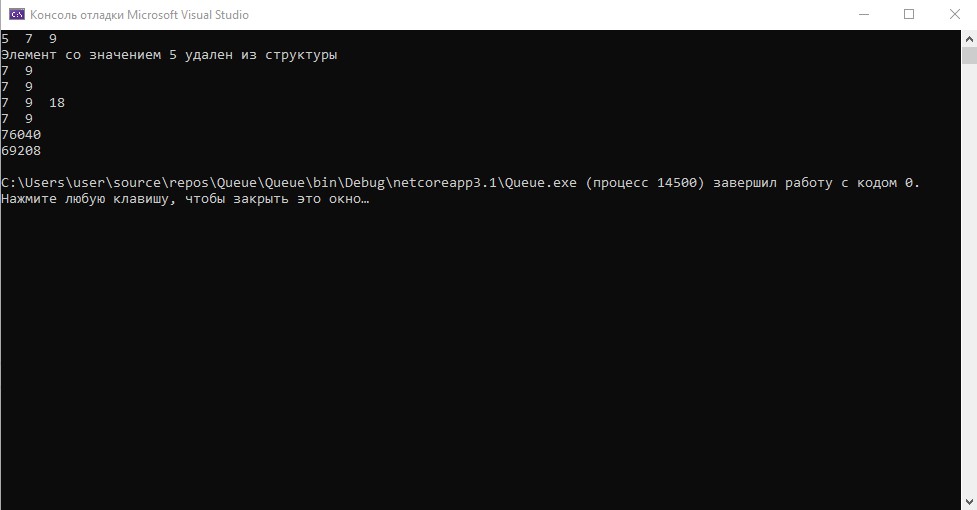
FreeingUpMemory();

}

}

}

**Вывод программы:**



Инициальзируем новую очередь в методе Main.

Вводим целочисленные элементы 5, 7, 9 в структуру используя метод AddToQueue. Передовая параметрами очередь в которую хотим добавить новый элемент и значение элемента.

Используя метод PrintQueue выводим все элементы на экран консоли.

Применив метод ReemoveToQueue, и передав в параметром очередь из которой нужно удалить элемент, мы удаляем первый элемент очереди, в нашем случае это элемент со значением 5. Выводим элементы на экран консоли. Элемент со значением 5 отсутствует.

Далее создаём независимую копию имеющейся очереди. Выводим на экран консоли все ее элементы, затем вводим новый элемент со значением 18 в новую очередь и сверяем очередь 1 с очередью 2.

Удаляем память выделенную на очередь 1. Вызываем метод «FreeingUpMemory» запускающий методы для сборщика мусора. Получаем вывод на экран консоли затраченную память до и после очистки.

**Вывод:**

В данной лабораторной работе были изучены различные структуры данных. Реализованы согласно варианту такие структуры как однонаправленный список и очередь.