Практическая работа 32
Разработка модулей для работы со стеком и хеш-таблицами

Цель занятия: Получить практические навыки разработки модулей для работы со стеком и хеш-таблицами

Перечень оборудования и программного обеспечения

Персональный компьютер
Microsoft Office (Word, Visio)
Microsoft Visual Studio

Краткие теоретические сведения

1 Коллекция Stack и ее применение

Стек (Stack) - это класс коллекции, организованный по принципу LIFO (последним вошел -первым вышел). Классическая аналогия - стопка тарелок. Верхняя тарелка всегда будет использована первой, в то время как для изъятия тарелок из середины, необходимо сначала снять верхние.

Добавление элемента, называемое также проталкиванием (push), возможно только в вершину стека (добавленный элемент становится первым сверху). Удаление элемента, называемое также выталкиванием (pop), тоже возможно только из вершины стека, при этом второй сверху элемент становится верхним.

Стеки широко применяются в вычислительной технике. Например, для отслеживания точек возврата из подпрограмм используется стек вызовов, который является неотъемлемой частью архитектуры большинства современных процессоров. Языки программирования высокого уровня также используют стек вызовов для передачи параметров при вызове процедур.

1.1Создание объекта класса Stack

Создания объекта класса Stack выполняется практически так же, как создаётся объект любого другого класса. Рассмотрим основные варианты создания объектов класса Stack:

1) создаём объект x типа Stack, который не содержит ни одного элемента: Stack x = new Stack();

Это наиболее распространённый метод создания объекта типа Stack.

2) создаём объект x типа Stack, который рассчитан на хранение заданного количества элементов (n):

```
int n = 20;
Stack x = new Stack(n);
```

Здесь под стек будет выделено памяти не менее, чем под n элементов.

3) создаём объект x типа Stack, который заполняется элементами коллекции Col.

Stack x = new Stack (Col);

В этом случае создается новый экземпляр класса Stack, содержащий элементы, скопированные из указанной коллекции, и обладающий начальной емкостью, равной количеству скопированных элементов.

1.2 Добавление новых элементов в стек

Для добавление новых элементов в стек имеется метод Push(). Добавление всегда делается в начало стека.

x.Push(100);

Здесь в начало (верхушку) стека x добавляется число 100.

1.3 Удаление данных из стека

Для удаления данных также имеется несколько вариантов:

1) Удаление всех элементов стека (очистка стека):

x.Clear();

После выполнения такого оператора стек будет пустой. Сам объект x не удаляется!

2) Удаление следующего элемента стека для последующей обработки :

Object o = x.Pop();

3) Иногда необходимо получить объект без удаления его из стека (например для контроля). В таком случае используется метод Peek().

Object o = x.Peek();

1.4 Определение размера стека

Имеется специальное свойство Count — определяет текущее количество элементов в стеке.

int c = x.Count;

Емкость всегда больше или равна значению свойства Count. Если значение Count превысит емкость при добавлении элементов, емкость автоматически увеличивается посредством перераспределения внутреннего массива перед копированием старых элементов и добавлением новых.

1.5 Проверка наличия элемента в стеке

Для проверки наличия объекта в очереди используется метод Contains(). Данный метод определяет равенство с помощью вызова метода Object.Equals().

С помощью этого метода выполняется линейный поиск; поэтому данный метод является операцией O(n), где n— значение свойства Count.

1.6 Преобразование в массив

Для преобразования стека в массив существует 2 метода: CopyTo() и ToArray().

1) В случае использования метода CopyTo() элементы коллекции стека копируюстя в существующий одномерный массив Array, начиная с указанного значения индекса массива.

Существующие по указанным индексам в массиве элементы при этом переопределяются.

```
Stack myStack = new Stack();
Array myTargetArray=new Array(15);
myStack.CopyTo( myTargetArray, 6 );
```

2) В случае использования метода ToArray() элементы коллекции стека копируются в новый массив.

```
Stack myStack = new Stack();
Object[] myStandardArray = myStack.ToArray();
```

2 Коллекция HashTable и ее применение

Структура HashTable

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Класс HashTable предоставляет коллекцию пар "ключ-значение", которые упорядочены по хэш-коду ключа.

2.1 Свойства хеш-таблицы

Каждый элемент в паре "ключ-значение" хранится в объекте DictionaryEntry. Ключ не может быть равным null, а значение — может.

Важное свойство хеш-таблиц состоит в том, что, при некоторых разумных допущениях, все три операции (поиск, вставка, удаление элементов) в среднем выполняются за время 0(1). При добавлении элемента в коллекцию Hashtable он помещается в определенный сегмент в зависимости от хэш-кода ключа. В дальнейшем поиск ключа осуществляется только в определенном сегменте с использованием хэш-кода ключа. Таким образом, в значительной степени уменьшается количество операций сравнения ключей, которое требуется для нахождения элемента.

Но при этом не гарантируется, что время выполнения отдельной операции мало. Это связано с тем, что при достижении некоторого значения коэффициента заполнения необходимо осуществлять перестройку индекса хеш-таблицы: увеличить значение размера массива и заново добавить в пустую хеш-таблицу все пары.

Показатель загрузки коллекции Hashtable определяет максимальное отношение количества элементов к количеству сегментов. Снижение показателя загрузки уменьшает среднее время поиска за счет увеличения объема используемой памяти. Значение показателя загрузки по умолчанию, равное 1.0, обычно обеспечивает наилучшее соотношение между объемом памяти и временем поиска. При создании коллекции Hashtable может быть задан другой показатель загрузки.

Каждый ключевой объект в коллекции Hashtable должен иметь свою собственную хэш-функцию, доступ к которой может быть получен при вызове метода GetHash.

2.2 Методы класса HashTable

Инициализирует новый пустой экземпляр класса Hashtable с заданными по Hashtable() умолчанию начальной емкостью, показателем загрузки, поставщиком хэш-кода и объектом сравнения.

Hashtable(IDictionary) - инициализирует новый пустой экземпляр класса Hashtable посредством копирования элементов из указанного словаря в новый объект Hashtable. У нового объекта Hashtable исходная емкость равна числу копируемых элементов, и он обладает заданными по умолчанию показателем загрузки, поставщиком хэш-кода и объектом сравнения.

Hashtable(Int32) - инициализирует новый пустой экземпляр класса Hashtable с указанной исходной емкостью и заданными по умолчанию показателем загрузки, поставщиком хэш-кода и объектом сравнения.

Add - добавляет элемент с указанными ключом и значением в коллекциюHashtable.

Clear - удаляет все элементы из коллекции Hashtable.

Contains, ContainsKey - определяет, содержит ли коллекция Hashtable указанный ключ.

ContainsValue - определяет, содержит ли коллекция Hashtable указанное значение.

GetHash - возвращает хэш-код указанного ключа.

KeyEquals - сравнивает указанный объект класса Object с указанным ключом, который содержится в коллекции Hashtable.

Remove - удаляет элемент с указанным ключом из коллекции Hashtable.

2.3 Использование класса HashTable

В операторе foreach языка C# необходима информация о типе каждого элемента коллекции. Так как каждый элемент коллекции Hashtable представляет собой пару "ключ-значение", тип элемента не является типом ключа или типом значения. Вместо этого в качестве типа элемента используется DictionaryEntry.

Например:

```
Hashtable myHashtable = new Hashtable();
//первый элемент - ключ, второй - значение
myHashtable.Add(1, "Book 1");
myHashtable.Add(2, "Book 2");
myHashtable.Add(3, "Book 3");
myHashtable.Add(4, "Book 4");
//При повторной попытке добавить существующий ключ будет сгенерировано исключение
foreach (DictionaryEntry de in myHashtable) {
```

```
Console.WriteLine("Key = {0}, Value = {1}", de.Key, de.Value);
}
if (MyHashTable.ContainsKey(2)) myHashTable.Remove(2);
```

2.4 Свойства класса HashTable

Класс имеет множество свойств, наиболее часто используемые из которых:

Count - Получает число пар "ключ-значение" в коллекции Hashtable.

Keys - Получает коллекцию ICollection, содержащую ключи из коллекции Hashtable.

Values - Получает коллекцию ICollection, содержащую значения из коллекции Hashtable.

Свойство Item можно также использовать для добавления новых элементов посредством задания значения ключа, которого не существует в коллекции Hashtable.

Использование Item.

Например

myHashTable[5] = "Book 5".

Однако, если указанный ключ уже существует в коллекции Hashtable, задание свойства Item перезаписывает прежнее значение. Напротив, метод Add не изменяет существующих элементов.

Пример

Создайте проект приложения Windows Forms. В обозревателе решений щелкните правой кнопкой мыши имя проекта, укажите пункт Добавить и выберите класс, чтобы добавить модуль класса. По умолчанию в проект добавляется Class1.

Замените код в модуле класса Class1 на следующий код:

```
using System.Collections;

public static  class Person
{
   public static Hashtable GetHashtable()
   {
     // Create and return new Hashtable.
     Hashtable Mytable = new Hashtable();
     Mytable.Add("2674367", "Директор");
     Mytable.Add("2674360", "Секретарь");
     Mytable.Add("2674536", "Бухгалтер");
     return Mytable;
   }
}
```

После объявления класса Form1 добавьте код:

```
public  Hashtable Mytable = Person.GetHashtable();
```

Поместите на форму элемент управления Button и измените свойство Text на Выдача элементов, щелкните кнопку, чтобы открыть ее окно кода и вставьте следующий код в событие Button1_Click:

```
using System.Collections;

if (Mytable.Count == 0)
    MessageBox.Show("The hashtable is empty");
  else
  {

    IDictionaryEnumerator Enumerator;
    Enumerator = Mytable.GetEnumerator();

    while (Enumerator.MoveNext())
    {
      MessageBox.Show(Enumerator.Value.ToString());
    }
  }
```

Поместите на форму элемент управления Button и измените свойство Text на Количество элементов, щелкните кнопку, чтобы открыть ее окно кода и вставьте следующий код в событие Button2_Click:

```
MessageBox.Show(Convert.ToString(Mytable.Count));
```

Поместите на форму элемент управления Button и измените свойство Text на Удаление элементов, щелкните кнопку, чтобы открыть ее окно кода и вставьте следующий код в событие Button3_Click:

```
Mytable.Remove("2674536");
  foreach (DictionaryEntry pers in Mytable)
  {
    MessageBox.Show(Mytable[pers.Key].ToString());
  }
```

Поместите на форму элемент управления Button и измените свойство Text на Добавление элементов, щелкните кнопку, чтобы открыть ее окно кода и вставьте следующий код в событие Button4_Click:

```
Mytable.Add("2674536","Заместитель");
```

Задания

1 Изучить теоретические сведения и задание к работе

2 В соответствии с вариантом задания составить отлаженную программу.

Порядок выполнения работы

…

Содержание отчета

1 Название работы
2 Цель работы
3 Технические средства обучения
4 Задания (условия задач)
5 Порядок выполнения работы
6 Вывод

Варианты заданий

Создайте класс и управляющую форму для хеш-таблицы по заданию, указанному в практической работе 31

Используемая литература

1. Гниденко, И. Г. Технология разработки программного обеспечения: учеб. пособие для СПО / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. — М.: Издательство Юрайт, 2017.
2. Шарп Джон Ш26 Microsoft Visual C#. Подробное руководство. 8-е изд. — СПб.: Питер, 2017.
3. Васильев А.Н. Программирование на C# для начинающих. Основные сведения. – Москва: Эксмо, 2018.
4. Васильев А.Н. Программирование на C# для начинающих. Особенности языка. – Москва: Эксмо, 2019.
5. http://msdn.microsoft.com/ru-ru/library/67ef8sbd.aspx.