Лабораторна робота 9:

Тема: Колекції.

Мета роботи: Класифікація колекцій. Колекції загального призначення: стек. черга, динамічний масив, хеш-таблиці

Теоретичні відомості

Колекції

У C# під колекцією розуміється деяка група об'єктів. Колекції спрощують реалізацію багатьох задач програмування, пропонуючи вже готові розв'язки для побудови структур даних. Усі колекції розроблені на основі чітко визначених інтерфейсів, тому стандартизують спосіб обробки групи об'єктів. Середовище .NET Framework підтримує три основні типи колекцій: загального призначення, спеціалізовані та орієнтовані на побітову організацію даних.

Колекції загального призначення визначені в просторі імен `System.Collection` і реалізують такі структури даних, як стеки, черги, динамічні масиви, словники (хеш-таблиці, призначені для зберігання пар ключ/значення), відсортований список для зберігання пар ключ/значення. Колекції загального призначення працюють із даними типу **Object**, тому їх можна використовувати для зберігання даних будь-якого типу.

Колекції спеціального призначення визначені в просторі імен `System.Collection.Specialized` і орієнтовані на обробку даних конкретного типу або на обробку даних унікальним способом. Наприклад, існують спеціалізовані колекції, призначені тільки для обробки рядків.

У просторі імен `System.Collection` визначена єдина колекція, *орієнтована на побітову організацію даних*, яка служить для зберігання груп бітів і підтримує такий набір операцій, який не характерний для колекцій інших типів.

Колекції загального призначення

Класи колекцій загального призначення:

Клас	Опис
`Stack`	Стек - окремий випадок односпрямованого списку, що діє за принципом: останнім прийшов - першим вийшов
`Queue`	Черга - окремий випадок односпрямованого списку, що діє за принципом:першим прийшов - першим вийшов
`ArrayList`	Динамічний масив, тобто масив який при необхідності може збільшувати свій розмір
`Hashtable`	Хеш-Таблиця для пар ключ/значення
`SortedList`	Відсортований список пар ключ/значення

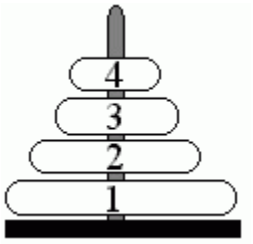
Розглянемо дані колекції більш докладно.

Зауваження. Абстрактний тип даних (АТД) *список* - це послідовність елементів $a_1, a_2, \ldots, a_n$ $(n \geq 0)$ одного типу. Кількість елементів *n* називається *довжиною списку*. Якщо $n > 0$, то a_1 називається *першим елементом списку*, а a_n - *останнім елементом списку*. У випадку $n = 0$ маємо *порожній список*, який не містить елементів. Важлива властивість списку полягає в тому, що його елементи

лінійно впорядковані відповідно до їхньої позиції в списку. Так елемент a_i *передує* a_{i+1} для $i=1, 2, \ldots n-1$ і a_i *іде* за a_{i-1} для $i=2, \ldots n$. Список називається *односпрямованим,* якщо кожний елемент списку містить посилання на наступний елемент. Якщо кожний елемент списку містить два посилання (одну на наступний елемент у списку, другу - на попередній елемент), то такий список називається *двунаправленным* (двозвязний). А якщо останній елемент зв'язати покажчиком з першим, то вийде кільцевий список.

Клас Stack

АТД стек - це окремий випадок односпрямованого списку, додавання елементів у який і вибірка елементів з якого виконуються з одного кінця, називаного вершиною стека (головою - `head`). При вибірці елемент виключається зі стека. Інші операції зі стеком не визначені. Говорять, що стек реалізує принцип обслуговування LIFO (`last in - fist out`, останнім прийшов - першим вийшов). Стек найпростіше представити собі у вигляді піраміди, на яку надягають кільця.



Отримати перше кільце можна тільки після того, як будуть зняті всі верхні кільця.

У C# реализацию АТД стек представляє клас Stack, який реалізує інтерфейси `ICollection`, `IEnumerable` і `ICloneable`. Stack - це динамічна колекція, розмір якої змінюється.

У класі Stack визначені наступні конструктори:

```
public Stack(); //створює порожній стек, початкова місткість якого рівна 10
public Stack(int capacity); // створює порожній стек, початкова місткість якого рівна capacity
public Stack(ICollection c); //створює стек, який містить елементи колекції, заданої
//параметром з, і аналогічної (аналогічної - із чим?) місткістю
```

Крім методів, певних в інтерфейсах, реалізованих класом Stack, у цьому класі визначені власні методи:

Метод	Опис
`public virtual bool Contains(Object v)`	Повертає значення `true`, якщо об'єкт `v` зберігається в стеці, а якщо ні, то повертає значення `false`.
`public virtual void Clear()`	Установлює властивість `Count` рівним нулю, тим самим очищаючи стек.
`public virtual Object Peek()`	Повертає елемент, розташований у вершині стека, але не витягаючи його зі стеку
`public virtual Object Pop()`	Повертає елемент, розташований у вершині стека, і витягає його зі стеку
`public virtual void Push(Object v)`	Поміщає об'єкт `v` у стек
`public virtual Object[] Toarray()`	Повертає масив, який містить копії елементів стеку.

Розглянемо кілька прикладів використання стека.

Приклад 1. Для заданого значення n запишемо в стек усі числа від 1 до n, а потім витягнемо зі стека:

```
using System;
using System.Collections;

namespace StackExample
{
    class Program
    {
        static void Main(string[] args)
        {

            Console.Write("n= ");
            int n = int.Parse(Console.ReadLine());
            //  System.Collections.Generic.
            Stack intstack = new Stack();
            for (int i = 1; i <= n; i++)
                intstack.Push(i);
            Console.WriteLine("Розмірність стека " + intstack.Count);

            Console.WriteLine("Верхній елемент стека = " + intstack.Peek());
            Console.WriteLine("Розмірність стека " + intstack.Count);

            Console.Write("Уміст стека = ");
            while (intstack.Count != 0)
                Console.Write("{0} ", intstack.Pop());
            Console.WriteLine("\n новая розмірність стека " + intstack.Count);
        }
    }
}
```

Приклад 2. У текстовому файлі знаходиться математичний вираз. Перевірити баланс круглих дужок у данім виразі.

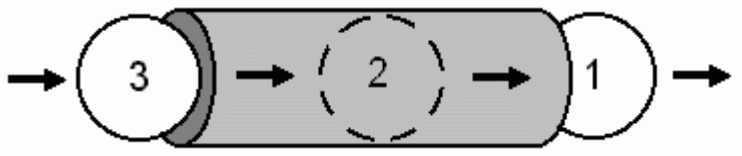
```
using System;
using System.Collections;
using System.IO;

namespace Myprogram
{
    class Program
    {
        public static void Main()
        {
            StreamReader filein = new StreamReader("t.txt");
            string line = filein.ReadToEnd();
            filein.Close();
            Stack skobki = new Stack();
            bool flag = true;
            //перевіряємо баланс дужок
            for (int i = 0; i < line.Length; i++)
            {
                //якщо поточний символ дужка, що відкривається, то поміщаємо її в стек
                if (line[i] == '(') skobki.Push(i);
                else if (line[i] == ')') //якщо поточний символ дужка, що закривається, то
                {
                   //якщо стек порожній, то для дужки, що закривається, не вистачає пари, що відкривається
                    if (skobki.Count == 0)
                    { flag = false; Console.WriteLine("Можливо в позиції " + i + "зайва ) дужка"); }
                    else skobki.Pop(); //інакше витягаємо парну дужку
                }
            }
            //якщо після перегляду рядка стек виявився порожнім, то дужки збалансовані
            if (skobki.Count == 0) { if (flag) Console.WriteLine("дужки збалансовані"); }
            else //інакше баланс дужок порушений
            {
                Console.Write("Можливо зайва ( дужка в позиції:");
                while (skobki.Count != 0)
                {
                    Console.Write("{0} ", (int)skobki.Pop());
```

```
                }
                Console.WriteLine();
            }
        }
    }
}
```

```
_________t.txt_________
(1+2)-4*(a-3)/(2-7+6)
```

Клас Queue

АТД черга - це окремий випадок односпрямованого списку, додавання елементів у який виконується в один кінець (хвіст), а вибірка проводиться з іншого кінця (голови). Інші операції із чергою не визначені. При вибірці елемент виключається із черги. Говорять, що черга реалізує принцип обслуговування FIFO (`fist in - fist out`, першим прийшов - першим вийшов). Черга найпростіше представити у вигляді вузької труби, в один кінець якої кидають м'ячі, а з іншого кінця якої вони вилітають. Зрозуміло, що м'яч, який був кинутий у трубу першим, першим і вилетить із іншого кінця.



У C# реализацию АТД очередь представляет класс `Queue`, который также как и стек реализует интерфейсы `ICollection`, `IEnumerable` и `ICloneable`. `Queue` - це динамічна колекція, розмір якої змінюється. При необхідності збільшення місткості черги відбувається з коефіцієнтом росту за замовчуванням рівним 2.0.

У класі `Queue` визначені наступні конструктори:

```
public Queue(); //створює порожню чергу, початкова місткість якої рівна 32
public Queue (int capacity); // створює порожню чергу, початкова місткість якої рівна capacity
//створює порожню чергу, початкова місткість якої рівна capacity, і коефіцієнт росту
//установлюється параметром n
public Queue (int capacity, float n);
//створює черга, яка містить елементи колекції, заданої параметром з, і аналогічної
//місткістю
public Queue (ICollection c);
```

Крім методів, певних в інтерфейсах, реалізованих класом `Queue`, у цьому класі визначені власні методи:

Метод	Опис
`public virtual bool Contains (Object v)`	Повертає значення true, якщо об'єкт v знаходиться в заданій черзі, а якщо ні, то повертає значення false
`public virtual void clear ()`	Установлює властивість `Count` рівним нулю, тим самим очищаючи чергу
`public virtual Object Dequeue ()`	Повертає об'єкт із початку заданої черги, видаляючи його із черги
`public virtual Object Peek ()`	Повертає об'єкт із початку заданої черги, не видаляючи його із черги

`public virtual void Enqueue(Object v)`	Додає об'єкт v у кінець черги
`public virtual Object [ ] ToArray ()`	Повертає масив, який містить копії елементів із заданої черги
`public virtual void Trimtosizeo`	Установлює властивість `Capacity` рівним значенню властивості `Count`

Розглянемо кілька прикладів використання черги.

Приклад 1. Для заданого значення `n` запишемо в чергу всі числа від `1` до `n`, а потім витягнемо їх із черги:

```
using System;
using System.Collections;
namespace MyQueue
{
    class Program
    {
        public static void Main()
        {
            Console.Write("n= ");
            int n = int.Parse(Console.ReadLine());
            Queue intq = new Queue();
            for (int i = 1; i <= n; i++)
                intq.Enqueue(i);
            Console.WriteLine("Розмірність черги " + intq.Count);

            Console.WriteLine("Верхній елемент черги = " + intq.Peek());
            Console.WriteLine("Розмірність черги " + intq.Count);

            Console.Write("Уміст черги = ");
            while (intq.Count != 0)
                Console.Write("{0} ", intq.Dequeue());
            Console.WriteLine("\nновая розмірність черги " + intq.Count);
        }
    }
}
```

Приклад 2. У текстовому файлі записана інформація про людей (прізвище, ім'я, по батькові, вік, вага через пробіл). Вивести на екран спочатку інформацію про людей молодше 40 років, а потім інформацію про всіх інших.

```
using System;
using System.Collections;
using System.IO;
using System.Text;

namespace MyQueue
{
    class Program
    {
        public struct One //структура для зберігання даних про одну людину
        {
            public string f;
            public string i;
            public string o;
            public int age;
            public float massa;
        }

        public static void Main()
        {
            StreamReader filein = new StreamReader("tm.txt",Encoding.UTF8);
            string line;
            Queue people = new Queue();
            One a;
            Console.WriteLine("ВІК МЕНШ 40 РОКІВ");
            while ((line = filein.ReadLine()) != null) //читаємо до кінця файлу
            {
```

```
                string[] temp = line.Split(' '); //розбиваємо рядок на складені елементи
                                                 //заповнюємо структуру
                a.f = temp[0];
                a.i = temp[1];
                a.o = temp[2];
                a.age = int.Parse(temp[3]);
                a.massa = float.Parse(temp[4]);
                // якщо вік менше 40 років, те виводимо дані на екран, інакше поміщаємо їх в
                //черга для тимчасового зберігання
                if (a.age < 40)
                    Console.WriteLine(a.f + "\t" + a.i + "\t" + a.o + "\t" + a.age + "\t" + a.massa);
                else people.Enqueue(a);
            }
            filein.Close();

            Console.WriteLine("ВІК 40 РОКІВ І БІЛЬШЕ");
            while (people.Count != 0) //витягаємо із черги дані
            {
                a = (One)people.Dequeue();
                Console.WriteLine(a.f + "\t" + a.i + "\t" + a.o + "\t" + a.age + "\t" + a.massa);
            }

        }
    }
}
```

```
_________________tm.txt_____________________
Іванов Сергій Миколайович 21 64
Петров Ігор Юрійович 45 88
Семенов Михайло Олексійович 20 70
Пиманов Олександр Дмитрович 53 101
```

Клас ArrayList

У C# стандартні масиви мають фіксовану довжину, яка не может змінюватися під час виконання я програми. Клас `ArrayList` призначений для підтримки динамічних масивів, які при необхідності можуть збільшуватися або скорочуватися.

Об'єкт класу `ArrayList` являє собою масив змінної довжини, елементами якого є об'єктні посилання. Будь-який об'єкт класу `ArrayList` створюється з деяким початковим розміром. При перевищенні цього розміру колекція автоматично подвоюється. У випадку видалення об'єктів масив можна скоротити.

Клас `ArrayList` реалізує інтерфейси `ICollection`, `IList`, `IEnumerable` і `ICloneable`. У класі `ArrayList` визначені наступні конструктори:

```
//створює порожній масив з максимальною ємністю рівної 16 елементам, при поточній розмірності 0
public ArrayList()
public ArrayList(int capacity) //створює масив із заданою ємністю capacity, при поточній розмірності 0
public ArrayList(ICollection col) //будує масив, який ініціалізується елементами з колекції col
```

Крім методів, певних в інтерфейсах, які реалізує клас ArrayList, у ньому визначені й власні методи:

Метод	Опис
`public virtual void Addrange (ICollection c)`	Додає елементи з колекції в кінець заданої колекції
`public virtual int Binarysearch (Object v)`	У заданій відсортованій колекції виконує пошук значення, заданого параметром `v`. Повертає індекс знайденого елемента. Якщо шукане значення не виявлене, повертає від'ємне значення.

`public virtual int Binarysearch (Object v, IComparer comp)`	У заданій відсортованій колекції виконує пошук значення, заданого параметром v, на основі методу порівняння об'єктів, заданого параметром comp. Повертає індекс знайденого елемента. Якщо шукане значення не виявлене, повертає від'ємне значення.
`public virtual int Binarysearch (int startIdx, int count, Object v, IComparer comp)`	У заданій відсортованій колекції виконує пошук значення, заданого параметром v, на основі методу порівняння об'єктів, заданого параметром comp. Пошук починається з елемента, індекс якого дорівнює значенню startidx, і включає count елементів. Метод повертає індекс знайденого елемента. Якщо шукане значення не виявлене, повертає від'ємне значення.
`public virtual void Copyto(Array ar, int startidx)`	Копіює вміст заданої колекції, починаючи з елемента, індекс якого дорівнює значенню startidx, у масив, заданий параметром ar. Масив приймач повинен бути одномірним і сумісним по типу з елементами колекції.
`public virtual void Copyto(int srcIdx, Array ar, int destidx, int count)`	Копіює count елементів заданої колекції, починаючи з елемента, індекс якого дорівнює значенню srcidx, у масив, заданий параметром ar, починаючи з елемента, індекс якого дорівнює значенню destidx. Масив приймач повинен бути одномірним і сумісним по типу з елементами колекції
`public virtual ArrayList GetRange(int idx, int count)`	Повертає частина заданої колекції типу ArrayList. Діапазон колекції, що вертається, починається з індексу idx і включає count елементів. об'єкт, що вертається, посилається на ті ж елементи, що й поточний об'єкт
`public static ArrayList FixedSize(ArrayList ar)`	Перетворює колекцію ar в **ArrayList**-Масив з фіксованим розміром і повертає результат
`public virtual void InsertRange(int startIdx, ICollection c)`	Вставляє елементи колекції, заданої параметром з, у поточну колекцію, починаючи з індексу, заданого параметром startidx
`public virtual int LastIndexOf(Object v)`	Повертає індекс останнього входження об'єкта **v** у заданій колекції. Якщо шуканий об'єкт не виявлений, повертає від'ємне значення
`public static ArrayList Readonly(ArrayList ar)`	Перетворює колекцію ar в ArrayList-Масив, призначений тільки для читання
`public virtual void RemoveRange(int idx, int count)`	Видаляє count елементів із заданої колекції, починаючи з елемента, індекс якого дорівнює значенню idx
`public virtual void Reverse()`	Розташовує елемент заданої колекції у зворотному порядку
`public virtual void Reverse(int startIdx, int count)`	Розташовує у зворотному порядку count елементів заданої колекції, починаючи з індексу startIdx
`public virtual void SetRange(int startIdx, ICollection c)`	Заміняє елементи заданої колекції, починаючи з індексу startIdx, елементами колекції, заданої параметром з
`public virtual void Sort()`	Сортує колекцію по зростанню

`public virtual void Sort(Icomparer comp)`	Сортує поточну колекцію на основі методу порівняння об'єктів, заданого параметром `comp`. Якщо параметр comp має нульове значення, для кожного об'єкта використовується стандартний метод порівняння
`public virtual void Sort ( int startidx, int endidx, icomparer comp)`	Сортує частина заданої колекції на основі методу порівняння об'єктів, заданого параметром comp. Сортування починається з індексу `startidx` і закінчується індексом `endidx`. Якщо параметр comp має нульове значення, для кожного об'єкта використовується стандартний метод порівняння
`public virtual Object [ ] ToArray ()`	Повертає масив, який містить копії елементів заданого об'єкта
`public virtual Array ToArray (Type type)`	Повертає масив, який містить копії елементів заданого об'єкта. Тип елементів у цьому масиві задається параметром type
`public virtual void TrimToSize()`	Установлює властивість Capacity рівним значенню властивості Count

Властивість `Capacity` дозволяє довідатися або встановити ємність заданого динамічного масиву типу `ArrayList`. Ємність являє собою кількість елементів, які можна зберегти в `ArrayList` -Масиві без його збільшення. Якщо вам заздалегідь відомо, скільки елементів повинне втримуватися в `ArrayList`-Масиві, то розмірність масиву можна встановити використовуючи властивість `Capacity`, съэкономив тим самим системні ресурси. Якщо потрібно зменшити розмір `ArrayList`-Масиву, то шляхом установки властивості `Capacity` можна зробити його меншим. Але встановлюване значення не повинне бути менше значення властивості `Count`, інакше буде сгенерировано виключення `ArgumentOutOfRangeException`. Щоб зробити ємність ArrayList-`Масиву` рівної дійсній кількості елементів, збережених у ньому в цей момент, установите властивість `Capacity` рівним властивості `Count`. Того ж ефекту можна добитися, викликавши метод `Trimtosize ()`.

Розглянемо кілька прикладів використання динамічного масиву.

```
using System;
using System.Collections;

namespace CollectionCS
{
    class Program
    {
        static void ArrayPrint(string s, ArrayList a)
        {
            Console.WriteLine(s);
            foreach (int i in a)
                Console.Write(i + " ");
            Console.WriteLine();
        }
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");

        ArrayList myarray = new ArrayList();
                Console.WriteLine("Початкова ємність масиву: " + myarray.Capacity);
                Console.WriteLine("Початкова кількість елементів: " + myarray.Count);

                Console.WriteLine("\nдобавили 5 цифр");
```

```
            for (int i = 0; i < 5; i++) myarray.Add(i);
            Console.WriteLine("Поточна ємність масиву: " + myarray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " + myarray.Count);
            ArrayPrint("Уміст масиву", myarray);

            Console.WriteLine("\noптимизируем ємність масиву");
            myarray.Capacity = myarray.Count;
            Console.WriteLine("Поточна ємність масиву: " + myarray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " + myarray.Count);
            ArrayPrint("Вміст масиву", myarray);

            Console.WriteLine("\nдобавляем елементи в масив");
            myarray.Add(10);
            myarray.Insert(1, 0);
            myarray.AddRange(myarray);
            Console.WriteLine("Поточна ємність масиву: " + myarray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " + myarray.Count);
            ArrayPrint("Уміст масиву", myarray);

            Console.WriteLine("\nудаляем елементи з масиву");
            myarray.Remove(0);
            myarray.RemoveAt(10);
            Console.WriteLine("Поточна ємність масиву: " + myarray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " + myarray.Count);
            ArrayPrint("Вміст масиву", myarray);

            Console.WriteLine("\nудаляем увесь масив");
            myarray.Clear();
            Console.WriteLine("Поточна ємність масиву: " + myarray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " + myarray.Count);
            ArrayPrint("Вміст масиву", myarray);
        }

    }
}
```

Приклад 2. У текстовому файлі записана інформація про людей (прізвище, ім'я, по батькові, вік, вага через пробіл). Вивести на екран інформацію про людей, відсортовану за віком.

```
using System;
using System.Collections;
using System.IO;
using System.Text;

namespace ProgrammSort
{
    class Program
    {
        public struct One //структура для зберігання даних про одну людину
        {
            public string f;
            public string i;
            public string o;
            public int age;
            public float massa;
        }

        public class Sortbyage : IComparer //реалізація стандартного інтерфейсу
        {
            int IComparer.Compare(Object x, Object y)  //перевизначення методу Compare
            {
                One t1 = (One)x;
                One t2 = (One)y;
                if (t1.age > t2.age) return 1;
                if (t1.age < t2.age) return -1;
                return 0;
            }
        }

        static void Arrayprint(string s, ArrayList a)
        {
```

```
            Console.WriteLine(s);
            foreach (One x in a)
                Console.WriteLine(x.f + "\t" + x.i + "\t" + x.o + "\t" + x.age + "\t" + x.massa);
        }

        static void Main(string[] args)
        {
            StreamReader filein = new StreamReader("ts.txt", Encoding.UTF8);
            string line;
            One a;
            ArrayList people = new ArrayList();
            string[] temp = new string[5];
            while ((line = filein.ReadLine()) != null) //цикл для організації обробки файлу
            {
                temp = line.Split(' ');
                a.f = temp[0];
                a.i = temp[1];
                a.o = temp[2];
                a.age = int.Parse(temp[3]);
                a.massa = float.Parse(temp[4]);
                people.Add(a);
            }
            filein.Close();

            Arrayprint("Вихідні дані: ", people);
            people.Sort(new Program.Sortbyage());  //виклик сортування
            Arrayprint("Відсортовані дані: ", people);
        }
    }
}
```

```
_____________t.txt_________________
Іванов Сергій Миколайович 21 64
Петров Ігор Юрійович 45 88
Семенов Михайло Олексійович 20 70
Пиманов Олександр Дмитрович 53 101
```

Зауваження. Зверніть увагу на те, що в даному прикладі був розроблений вкладений клас `Sortbyage`, що реалізує стандартний інтерфейс `IComparer`. У цьому класі був перевантажений метод `Compare`, що дозволяє порівнювати між собою два об'єкти типу one. Створений клас використовувався для сортування колекції за заданим критерієм (за віком).

Клас Hashtable

Клас `Hashtable` призначений для створення колекції, у якій для зберігання об'єктів використовується хеш-таблиця. У хеш-таблиці для зберігання інформації використовується механізм, іменований хешированием (hashing). Суть хеширования полягає в тому, що для визначення унікального значення, яке називається хеш-кодом, використовується інформаційний уміст відповідного йому ключа. Хеш-Код потім використовується в якості індексу, по якім у таблиці відшукуються дані, відповідні до цього ключа. Перетворення ключа в хеш-код виконується автоматично, тобто сам хеш-код ви навіть не побачите. Але перевага хеширования - у тому, що воно дозволяє скорочувати час виконання таких операцій, як пошук, зчитування й запис даних, навіть для більших обсягів інформації.

Клас `Hashtable` реалізує стандартні інтерфейси `IDictionary`, `ICollection`, `IEnumerable`, `ISerializable`, `IDeserializationCallback` і `ICloneable`. Розмір хеш-таблиці може динамічно змінюватися. Розмір таблиці збільшується тоді, коли кількість елементів перевищує значення, рівне добутку місткості таблиці і її коефіцієнта заповнення, який може ухвалювати значення на інтервалі від `0,1` до `1,0`. За замовчуванням установлений коефіцієнт рівний 1,0.

У класі `Hashtable` визначене трохи конструкторів:

```
public Hashtable()    //створює порожню хеш-таблицю
// будує хеш-таблицю, яка инициализируется елементами колекції з
```

```
public Hashtable(Idictionary c)
public Hashtable(int capacity) //створює хеш-таблицю з місткістю capacity
//створює хеш-таблицю місткістю capacity і коефіцієнтом заповнення n
public Hashtable(int capacity, float n)
```

Крім методів, певних в інтерфейсах, які реалізує клас Hashtable, у ньому визначені й власні методи:

Метод	Опис
`public virtual bool ContainsKey (Object k)`	Повертає значення true, якщо в заданої хеш-таблиці знаходиться ключ, заданий параметром `k`. А якщо ні, то повертає значення `false`
`public virtual bool ContainsValue (Object v)`	Повертає значення true, якщо в заданої хеш-таблиці знаходиться значення, задане параметром `v`. А якщо ні, то повертає значення `false`
`public virtual IDictionaryEnumerator Getenumerator()`	Повертає для заданої хеш-таблиці нумератор типу `Idictionaryenumerator`

У класі `Hashtable`, крім властивостей, певних у реалізовані їм інтерфейсах, визначено два власні public-`властивості`:

```
public virtual ICollection Keys { get; } //дозволяє одержати колекцію ключів
public virtual ICollection Values { get; } //дозволяє одержати колекцію значень
```

Для додавання елемента в хеш-таблицю необхідно викликати метод `Add()`, який ухвалює два окремі аргументи: ключ і значення. Важливо відзначити, що хеш-таблиця не гарантує збереження порядку елементів, т. до хеширование звичайно не застосовується до відсортованих таблиць.

Розглянемо приклад, який демонструє використання Hashtable колекції:

Приклад 1:розглянемо прості операції з хеш-таблицею

```
using System;
using System.Collections;

namespace MyHashtable
{
    class Program
    {
        static void printtab(string s, Hashtable a)
        {
            Console.WriteLine(s);
            ICollection key = a.Keys; //Прочитали всі ключі
            foreach (string i in key)//використання ключа для одержання значення
            {
                Console.WriteLine(i + "\t" + a[i]);
            }
            Console.WriteLine();
        }


        static void Main(string[] args)
        {
            Hashtable tab = new Hashtable();
            Console.WriteLine("Початкова кількість елементів: " + tab.Count);
            printtab("Вміст таблиці: ", tab);

            Console.WriteLine("Додали в таблицю запису");
```

```
            tab.Add("001", "ПЕРШИЙ");
            tab.Add("002", "ДРУГИЙ");
            tab.Add("003", "ТРЕТіЙ");
            tab.Add("004", "ЧЕТВЕРТИЙ");
            tab.Add("005", "П'ЯТИЙ");
            Console.WriteLine("Поточна кількість елементів: " + tab.Count);
            printtab("Вміст заповненої таблиці", tab);
            tab["005"] = "НОВИЙ П'ЯТИЙ";
            tab["001"] = "НОВИЙ ПЕРШИЙ";
            printtab("Вміст зміненої таблиці", tab);

        }
    }
}
```

Приклад 2. Розробимо найпростішу записну книжку, у яку можна додавати й видаляти телефони, а також здійснювати пошук номера телефону на прізвище й прізвища по номеру телефону.

```
using System;
using System.Collections;
using System.IO;
using System.Text;


namespace MyHashtable
{


    class Program
    {
        static void printtab(string s, Hashtable a)
        {
            Console.WriteLine(s);
            ICollection key = a.Keys; //Прочитали всі ключі
            foreach (string i in key)//використання ключа для одержання значення
            {
                Console.WriteLine(i + "\t" + a[i]);
            }
        }
        static void Main(string[] args)
        {
            StreamReader filein = new StreamReader("th.txt", Encoding.UTF8);
            string Line;
            Hashtable people = new Hashtable();
            while ((Line = filein.ReadLine()) != null) //цикл для організації обробки файлу
            {
                string[] temp = Line.Split(' ');
                people.Add(temp[0], temp[1]);
            }
            filein.Close();
            printtab("Вихідні дані: ", people);

            Console.WriteLine("Уведіть номер телефону");
            Line = Console.ReadLine();
            if (people.ContainsKey(Line)) Console.WriteLine(Line + "\t" + people[Line]);
            else
            {
                Console.WriteLine("Такого номера немає в записній книжці.\nвведите прізвище: ");
                string Line2 = Console.ReadLine();
                people.Add(Line, Line2);
            }
            printtab("Вихідні дані: ", people);

            Console.WriteLine("Уведіть прізвище для видалення");
            Line = Console.ReadLine();
            if (people.ContainsValue(Line))
            {
                ICollection key = people.Keys; //Прочитали всі ключі
                Console.WriteLine(Line);
                string del = "";
                foreach (string i in key)//використання ключа для одержання значення
```

```
                if (string.Compare((string)people[i], Line) == 0)
                {
                    del = i;
                    break;
                }

            Console.WriteLine(del + "\t" + people[del] + "- дані вилучені!!!");
            people.Remove(del);
            printtab("Змінені дані: ", people);
        }
        else Console.WriteLine("Такого абонента в записній книжці немає ");
    }


    }
}
```

```
_________t.txt_____________
12-34-56 Іванов
78-90-12 Петров
34-56-78 Семенов
90-11-12 Пиманов
```

Завдання до лабораторної роботи:

Порядок виконання роботи:

1) Клонування репозиторія C#: https://classroom.github.com/a/xj23a15K.
2) Написати C# програми згідно з варіантом завдання
3) Підготувати звіт в електронному виді надіслати мудл(https://moodle.chnu.edu.ua/course/view.php?id=3371) .

```
//  За бажанням студента для задач можна створювати консольний проект або WinForm
// Бажано для задач лаб. робіт створити окремі класи
// Виконання  виконати в стилі багатозаданості :
//   Lab9T2  lab9task2 = new Lab9T2; lab9task2.Run();
// При бажанні можна створити багатозадачний режим виконання задач.
```

Завдання 1. Варіанти задач. **Розв'язати наступні задачі з використанням класу Stack**

1.1. Дано файл, у якім записаний набір чисел. Переписати в інший файл усі числа у зворотному порядку.

1.2. Створити текстовий файл. Роздрукувати голосні букви цього файлу у зворотному порядку.

1.3. Надрукувати вміст текстового файлу `t`, виписуючи літери кожного його рядка у зворотному порядку.

1.4. Дано 2 рядки `s1` і `s2`. З кожної можна читати по одному символу. З'ясувати, чи є рядок `s2` зворотної `s1`.

1.5. Написати програму підрахунку виразу в префіксній формі.

1.6. Написати програму, яка перетворить вираз із префіксної форми в постфіксну.

1.7. Написати програму перетворення вираз із постфіксної форми в префіксну.

1.8. У текстовому файлі записана без помилок формула виду:
<формула>=<цифра>|M(<формула>,<формула>)|m(<формула>,<формула>)
<цифра>=0|1|2|3|4|5|6|7|8|9

`M` позначає обчислення максимуму, `m` – мінімуму
Обчислити значення цієї формули
Наприклад, `M(m(3,5),M(1,2))=3`

1.9. У текстовому файлі записана без помилок формула виду:
<формула>=<цифра>|p(<формула>,<формула>)|m(<формула>,<формула>)
<цифра>=0|1|2|3|4|5|6|7|8|9
m (a, b) = (a-b) mod 10,

p (a, b) = (a+b) mod 10.
Обчислити значення цієї формули.
Наприклад, `m (9, p (p (3, 5), m (3, 8))) = 6`.

1.10. Нехай символ `#` визначений у текстовому редакторі символ, що `як` стирає, Backspace, тобто рядок `abc#d##c` у дійсності є рядком `ac`. Даний текст, у якім зустрічається символ `#`. Перетворити його з урахуванням дії цього символу.

Завдання 2. Варіанти задач. **Розв'язати наступні задачі з використанням класу Queue:**

2.1. Дано текстовий файл. За один перегляд файлу надрукувати елементи файлу в наступному порядку: спочатку всі символи, відмінні від цифр, а потім усі цифри, зберігаючи вихідний порядок у кожній групі символів.

2.2. Дано файл, що містить числа. За один перегляд файлу надрукувати елементи файлу в наступному порядку: спочатку всі числа, з інтервалу `[a,b]`, потім усі числа, менші a, потім усі числа, більші b, зберігаючи вихідний порядок у кожній групі чисел.

2.3. Дано текстовий файл. За один перегляд файлу надрукувати елементи файлу в наступному порядку: спочатку всі слова, що починаються на гласну букву, потім усі слова, що починаються на згодну букву, зберігаючи вихідний порядок у кожній групі слів.

2.4. Дано файл, що містить числа. За один перегляд файлу надрукувати елементи файлу в наступному порядку: спочатку всі позитивні числа, потім усі негативні числа, зберігаючи вихідний порядок у кожній групі чисел.

2.5. Даний текстовий файл. За один перегляд файлу надрукувати елементи файлу в наступному порядку: спочатку всі слова, що починаються із прописної букви, потім усі слова, що починаються з рядкової букви, зберігаючи вихідний порядок у кожній групі слів.

2.6. Дано файл, що містить інформацію про співробітників фірми: прізвище, ім'я, по батькові, підлога, вік, розмір зарплати. За один перегляд файлу надрукувати елементи файлу в наступному порядку: спочатку всі дані про чоловіків, потім усі дані про жінок, зберігаючи вихідний порядок у кожній групі співробітників.

2.7. Дано файл, що містить інформацію про співробітників фірми: прізвище, ім'я, по батькові, підлога, вік, розмір зарплати. За один перегляд файлу надрукувати елементи файлу в наступному порядку: спочатку всі дані про співробітників, зарплата яких менше `10000`, потім дані про інших співробітників, зберігаючи вихідний порядок у кожній групі співробітників.

2.8. Дано файл, що містить інформацію про співробітників фірми: прізвище, ім'я, по батькові, підлога, вік, розмір зарплати. За один перегляд файлу надрукувати елементи файлу в наступному порядку: спочатку всі дані про співробітників молодше 30 років, потім дані про інших співробітників, зберігаючи вихідний порядок у кожній групі співробітників.

2.9. Дано файл, що містить інформацію про студентів: прізвище, ім'я, по батькові, номер групи, оцінки по трьом предметам поточної сесії. За один перегляд файлу надрукувати

елементи файлу в наступному порядку: спочатку всі дані про студентів, що успішно здали сесію, потім дані про інших студентів, зберігаючи вихідний порядок у кожній групі співробітників.

2.10. Дано файл, що містить інформацію про студентів: прізвище, ім'я, по батькові, номер групи, оцінки по трьом предметам поточної сесії. За один перегляд файлу надрукувати елементи файлу в наступному порядку: спочатку всі дані про студентів, що успішно навчаються на `4` і `5`, потім дані про інших студентів, зберігаючи вихідний порядок у кожній групі співробітників.

Завдання 3. Варіанти задач. **Розв'язати задачі із завдань 1 та 2 використовуючи клас ArrayList.**

Завдання 4. Варіанти задач. **(Для всіх.) Розв'язати задачу, використовуючи клас Hashtable:** реалізувати найпростіший каталог музичних компакт-дисків, який дозволяє:

- Додавати й видаляти диски.
- Додавати й видаляти пісні.
- Переглядати вміст цілого каталогу й кожного диска окремо.
- Здійснювати пошук усіх записів заданого виконавця по всім каталогу.