Изпитна тема №2: Обработка на колекции от данни

1.

1.1. Дефинира понятието сорс-контрол система. (2 т.)

Сорс-контрол система – системи за проследяване и управление на промените в файловете управлявани от тях. Предоставят история на промените, позволяваща за възстановяване на предишни версии на даден файл. Такива са: Git, GitHub, SVN, TFS и др.

Основната полза от сорс контрол системите е поддържаната от тях информация за история на промените. Историята позволява да вземеш стара версия на файл, да сравниш различни версии на файл, да маркираш или копираш продадени версии на целия изходен код и т.н. В сорс контрол системите няма понятие за директория както във файловите системи. В центъра на сорс контрола са само файловете.

1.2. Изброява команди за работа със сорс-контрол системите. (2 т.)

Съществуват различни видове операции извършващи се със сорс контрол системите

- от клиент към сървър – добавяне на нов файл, изтриване, преименуване;
- от сървър към клиент – сваляне на последната версия на файловете, синхронизиране на директория;
- промяна на състоянието на сървъра – редактиране на файл, заключване и отключване на файлове, създаване на разклонение в развитието на историята на файловете.

Някои команди:

clone – изтегля съществуващи файлове записани на отдалечено хранилище.

push – изпраща промените към отдалеченото хранилище

pull – изтегляне на промените от сървъра и сливането им с нашите промени.

commit – изпраща промените като и също създава точка към която можем да се върнем ако е нужно на сървъра.

merge – съединява клона в който ние работим с главния клон на проект.

1.3. Различава централизирани и децентрализирани сорс-контрол системи (4 т.)

Централизираните системи за сорс-контрол използват един сървър съдържащ всички версии на кода. Всеки път когато ще се правят промени трябва да се изтегля текущата версия на целия код. Когато правим промяна тя се запазва в сървъра в основната репозитория.

Децентрализираните системи за сорс-контрол в по-голямата си част използват същите принципи като централизираният вариант, но те не зависят от състоянието на сървъра, защото всеки клиент е всъщност сървър с клонираната репозитория хостван директно от

локалната машина. Този сървър служи като един вид буфер преди промените да преминат в основната репозитория.

2. Демонстрира откриване и отстраняване на проблем в програма с помощта на дебъгера - ??? (6 т.)

3.

3.1. Описва типове данни. (2 т.)

Integer (int, byte) – цели числа

Float (float, double) – числа с десетична запетая

Character(char) – единичен символ

String(string) – символни низове

Boolean(bool) – тип позволяващ само 2 състояния: true/false; 1/

3.2. Дефинира понятието обект. (2 т.)

Обект – абстрактен тип данни от обектно-ориентираното програмиране създаден от програмиста. Може да съдържа свойства, методи, също и други обекти; Инстанция на полетата и свойствата на даден клас със стойности зададени от потребителя и/или методи чрез конструктори.

3.3. Посочва видове бройни системи. (4 т.)

Десетична – позиционна бройна система която използва 10 цифри за да съставя числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Двоична – позиционна, използва 2 цифри: 0, 1.

Троична – позиционна, използва 3 цифри: 0, 1, 2.

.

.

.

Единадесетична – позиционна, използва 10 цифри и 1 буква: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A.

.

.

.

Шестнадесетична – позиционна, използва 10 цифри и 6 буква: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

.

.

.

3.4. Преобразува числа от една бройна система в друга и изчислява изрази с тях. (8 т.)

$ADF_{(16)} = ?_{(10)}$

$10*16^2+13*16+15*1=2783_{(10)}$

$ADF_{(16)} = ?_{(2)}$

$101011011111_{(2)}$

4.

4.1. Дефинира понятието едномерен и многомерен масив (4 т.)

Масив – линейна структура от данни с определен брой елементи и тип данни описани при създаването на масива.

Масивите са основна структура от данни за всички езици за програмиране и основополагащи за реализиране на други асбтрактни структури от данни. Представляват набор от променливи, които наричаме елементи, подредени в паметта един след друг, в повечето случаи са от един и същи тип данни и могат да бъдат достъпвани чрез индекси, които не са по-големи от N - 1, където N е размера на масива. Това че са подредени един след друг помага на програмистите да обработват масивите като едно цяло.

Най-използвани от различни размерности са векторите – едномерните масиви и матриците – двумерните масиви.

Едномерен масив – съставен само от една ос на индексите.

Дължината, която се задава при инициализацията на масива, не може да се променя в хода на програмата и остава фиксирана. Това става със заделяне на памет с оператор new – масив от цели числа, например – int[] arr = new int[6]; Без да оказваме какви са стойностите на шестте елемента автоматично се инициализират със стойността по подразбиране на типа данни. Променливата arr всъщност след нейното заделяне сочи към адрес в динамичната памет [heap], където се намира нейната стойност. Елементите на масивите се съхраняват на динамичната памет. За достъп до елемент на масива се използват името на масива и [], в които се вписва индекс за нужния елемент. Например arr[3] в нашия случай без инициализация на елементи ще върне 0. Ако искаме да инициализираме стойности на масива имаме няколко възможности, но най-използваната техника е:

```
int[] arr = {
1, 2, 3, 4, 5, 6
};
```

По този начин създаване масив с шест елемента и последния индекс на масива е 5 (N-1).

Многомерен масив – осите на индексите могат да бъдат многобройни, съответно размерите на отделните оси могат да бъдат различни.

Двумерните масиви например представляват таблици или погледнато през математическа гледна точка – матрици, с редове и колони. Елементите се достъпват чрез засичане на ред с колона и съответния резултат е елемент от двумерния масив. Достъпа става чрез извикване името на двумерния масив и прилагне на индексатора arr2d[row, col]. Декларация на двумерен масив става по следния начин:

```
int[,] arr2d = {
{ 1, 2, 3 },
      { 1, 2, 3 }
};
```

Масива arr2d има 2 реда и 3 колони.

Сумирането на елементите на едномерен масив може да се случи по два начина с циклична конструкция, в която на всяко завъртане в променлива ще се добавят стойностите на всеки елемент. А по-интуитивния начин за намиране сума на дадена колекция е с включване на using

System.Linq;
и използването на метода Sum() върху колекцията.

4.2. Описва декларирането на масиви (2 т.)

В C#

<тип данни>[] <име на масива> = new <тип данни>[<големина на масива>];// едномерен масив чиито стойности не са зададени

<тип данни>[] <име на масива> = { <стойност>,<стойност> , <стойност>};//едномерен масив с 3 елемента

<тип данни>[,,…,,] <име на масива> = new <тип данни>[<големина на ос 1>,<големина на ос 2>, …, <големина на ос n>,];// многомерен масив без зададени стойности

<тип данни>[,,…,,] <име на масива> ={ { <стойност>,<стойност> , <стойност>}, <стойност>,<стойност> , <стойност>}, …, <стойност>,<стойност> , <стойност>} }//многомерен масив с 3 елемента във всяка ос

4.3. Илюстрира графично едномерни и многомерни масиви. (4 т.)

Едномерни масиви

индекс	стойност
0	13
1	15
2	64
…	…
n	5456

При извикване на елемент с индекс 2 ще бъде присвоена стойността 64

Многомерни масиви

Индекс 1	Стойност 1	Индекс 2	Стойност 2
0	254	0	345
1	78654	1	6373
…		…	
n	54	n	743

При извикване на елемент[1,0] стойностите ще бъдат 78654 и 345.

4.4. Разработва алгоритми върху масиви. (6 т.)

Алгоритъм за намиране на най-големия елемент в масив:

```
using System;

  public class Program
  {
        public static void Main()
        {
              int[] arr = {34,456,89465,132,4659,321498321,12};
              int max = arr[0];
              for(int i = 0; i < arr.Length; i++){
                    if(arr[i]>max){
                    max=arr[i];

              }

                    Console.WriteLine(max);
        }

  }
```

Очаква се да се изпише на конзолния прозорец 321498321.

5.

5.1. Дефинира понятието списък (2 т.)

Списък – линейна структура от данни с динамична дължина.

5.2. Описва декларирането на списъци (2 т.)

В C#

```
using System;
using System.Collections.Generic;

public class Example
{
      public static void Main()
      {
            List<тип данни>> listEx= new List<тип данни>> ();
      }
}
```

5.3. Дава пример за основните операции със списъци (4 т.)

ListEX.Add(<елемент>);// добавя елемент в края на списъка

listEx.Contains(<елемент>);// търси дали елемента съществува

listEx.Insert(<index>, <element>);//добавя елемент на даденото място

listEx.RemoveAt(<index>)//изтрива елемент от дадения индекс

5.4. Прави заключения за предимствата и недостатъците при употребата на списъци спрямо масиви (8 т.)

Листовете са по-лесни за сортиране, изтриване и вмъкване на данни, но са по-бавни в достъпването им и им е нужно повече място от масивите.

6.

6.1. Описва декларирането на символен низ (2 т.)

В езика C# за низ се използва типът **string** или системния тип **System.String**, а за съхранение на един символ типът ***char***.

Пример:

```
public static string GetBetween (string strSource, string strStart, string strEnd) {
if (strSource is null) {
throw new ArgumentNullException(nameof(strSource));
}
if (strSource.Contains(strStart) && strSource.Contains(strEnd)) {
int Start, End;

Start = strSource.IndexOf(strStart, 0) + strStart.Length;
End = strSource.IndexOf(strEnd, Start);

return strSource.Substring(Start, End - Start);
}

return "";
```

string name=“Yanko“;//деклариране и дефиниране на символен низ

6.2. Посочва методи за работа със символни низове от изучаван език за програмиране (2 т.)

Класа System.String предоставя набор от различни методи за работа с текстови данни. Някой от често ползваните са :

name.Length();//посочва дължината на низа

name.Split();//разделя низа на поднизове

name.Contains();//проверява дали в низа се съдържа друг подниз

6.3. Разработва програми за алгоритми за обработка на текст чрез операции за текстови низове (извличане на подниз, замяна на низ и др.) (6 т.)

```
string plc = null;

while (name.Length < 10)

        {
           plc = name += "-";


        }

Console.WriteLine(plc);

if (plc == null)

{

   throw new ArgumentNullException(nameof(plc));

}

if (plc.Contains("o-"))

{

   name =plc+"Very Co-l";

}

Console.WriteLine(name);
```

7.

7.1. Дефинира понятието речник (2 т.)

Речник(хеш-таблица) – структура от данни съставена от двойки ключ-елемент, осигуряваща достъп до елементите по ключ.. Данните се записват заедно с ключовете. Могат да бъдат сортирани или не. Бързодействието на операцията търсене по ключ е големият плюс на речника, както и добавянето и изтриването на елементи. Подобни операции има и в класа List<T>, но тук те са много по–ефективни.

За използването на речник трябва типа, който използваме за ключа да предефинира метода GetHashCode() на класа Object. Към GetHashCode() метода има няколко

изисквания:
1. Не трябва да генерира изключения.

2. Трябва да се изпълнява бързо без да губи време в сложни и големи изчисления.
3. Различните обекти да могат да връщат една и съща стойност.
4. Един и същ обект трябва да връща една и съща стойност.
5. Минимум трябва да използва едно поле на инстанция.
6. Хеш-кодът не трява да се променя, докато съществува обектът.

Друго изискване към типа на ключа е да имплементира метод IEquatable<T>.Equals() или да предефинра метода Equals() на класа Object. Equals() се използва за срявнение на хеш-кодовете на различните обекти, които трябва да връщат един и същ хеш-код.
Конструкторите на класа Dictionary<TKey, TValue> изглеждат така:
1. public Dictionary() – създава празен речник, а капацитета се определя по подразбиране.
2. public Dictionary(IDictionary<TKey, TValue> dictionary) – създава речник с елементите, зададени в параметъра dictionary.
3. public Dictionary(int capacity) – позволява да определим капацитета на речника.

Класът имплементира следните интерфейси:
IDictionary, IDictionary<TKey, TValue>,
ICollection, ICollection<KeyValuePair<TKey, TValue>, IEnumerable, IEnumerable<KeyValuePair<TKey, TValue>, ISerializiable, IDeserializationCallback.

Най – използваните методи на класа са:
Add() – добавя в речника двойка ключ-стойност, ако вече има елемент със същия ключ се хвърля изключение.
ContainsKey() – връща true, ако речника съдържа обекта key.
ContainsValue – връща true, ако речника съдържа стойността value.
Remove() – изтрива ключа key от речника, и връща true при успех и false при неналичието на такъв ключ за премахване от речника.
Keys – колекция с ключовете.
Values – колекция със стойностите.

7.2. Описва устройството на речник (хеш-таблица). (2 т.)

```
using System.Collections;

Hashtable exHash = new Hashtable();
exHash.Add(1, "one");
exHash.Add(2, "two");
exHash.Add(3, "three");
exHash.Add(4, "four");
exHash.Add(5, "five");
exHash.Add(6, "six");
```

7.3. Решава задачи с използването на подходящи методи върху речник (6 т.)

7.4. Различава ключ и стойност в речника. (4 т.)

```
Console.WriteLine("List of Keys:");

foreach(int key in exHash.Keys)
{
   Console.WriteLine(key);
}
Console.WriteLine("List of Values:");
foreach (string val in exHash.Values)
{
   Console.WriteLine(val);
}
```

8. **Анализира фрагмент/и от код и идентифицира и поправя правилно грешките в написания програмен код, така че да реши поставената задача. Допълва кода, ако и когато това е необходимо.** (16 т.)