**Изпитна тема № 2: Обработка на колекции от данни**

Сорс-контрол системи – дефиниция, видове (централизирани и децентрализирани) и команди за работа. Видове типове данни. Понятие за обект. Видове бройни системи, операции и преобразуване от една бройна система в друга. Изчисления в бройни системи. Едномерни и многомерни масиви – дефиниция, деклариране, описание и илюстрация на структурата, разработка на алгоритми върху масиви. Списъци – дефиниция, деклариране, описание на структурата, основни операции, разработка на алгоритми върху списъци, изводи за предимствата и недостатъците спрямо масивите. Дебъгване и работа с дебъгер – откриване на проблеми в програми с използването на дебъгер. Символни низове – дефиниция, деклариране, методи за работа, разработка на алгоритми за обработка на текст (извличане на подниз, замяна на низ и др.). Речници (хеш-таблици) – дефиниция, деклариране, устройство, методи за работа, решаване на задачи върху речници, разлика между ключ и стойност. Определяне на реда на изпълнение на фрагментите в кода (program flow). Създаване и/или поправка/допълване на вече съществуващи компютърни програми, решаващи изчислителни и алгоритмични задачи със средствата на програмен език.

Сорс – контрол система е механизмът, по който се управлява работата по даден софтуерен проект. За да се улесни разработката на софтуер са създадени специални системи, които намаляват неудобствата при съвместна работа на много хора върху един проект. Системите за контрол на версиите обикновено използват едно централно хранилище, в което се съхраняват файловете на проекта.

**Git clone - Клониране на съществуващо Git хранилище**

**Git pull - Изтегляне и сливане на промени от отдалечено хранилище**

**Git add - Подготовка (добавяне / избор) на файлове за запис**

**Git commit - Предаване (commit) към локалното хранилище**

**Git status - Проверка на статуса (промените) в локалното хранилище**

**Git init - Създаване на ново локално хранилище (в текущата папка)**

**Git remote add - Създаване на отдалечено (+ кратко име за отдалечен Git URL)**

**Git push - Изпращане на промени (към отдалечено хранилище)**

Този подход предлага много предимства, особено спрямо локалните VCS. Например, всеки участник в проекта е запознат в доста добра степен на достоверност какво друг е правил по него.

Базовите типове данни в C# се разделят на следните видове:

-     Целочислени типове – sbyte, byte, short, ushort, int, uint, long, ulong;

-     Реални типове с плаваща запетая – float, double;

-     Реални типове с десетична точност – decimal;

-     Булев тип – bool;

-     Символен тип – char;

-     Символен низ (стринг) – string;

-     Обектен тип – object.

**Sbyte** е 8-битов целочислен тип със знак, **byte** е 8-битов беззнаков (unsigned) целочислен тип.

**Short** е 16-битов тип със знак.

**Ushort** е 16-битов беззнаков тип.

Следващият целочислен тип, който ще разгледаме, е **int**. Той е 32- битов знаков тип.

Типът **uint** е 32-битов беззнаков тип.

**Long** е 64-битов знаков тип.

Най-големият целочислен тип е типът **ulong**. Той е 64-битов беззнаков тип.

**Float** е 32-битов реален тип. Вторият реален тип с плаваща запетая в езика C# е типът **double**.

Реални типове – пример

float  floatPI = 3.14f;

Console.WriteLine(floatPI); // 3.14

double doublePI = 3.14;

Console.WriteLine(doublePI); // 3.14

double nan = Double.NaN;

Console.WriteLine(nan); // NaN

double infinity = Double.PositiveInfinity;

Console.WriteLine(infinity); // Infinity

Типът данни за реални числа с десетична точност в C# е 128-битовият тип **decimal**.

Пример, в който декларираме променлива от тип **decimal** и й присвояваме стойност:

decimal decimalPI = 3.14159265358979323846m;

Console.WriteLine(decimalPI); // 3.14159265358979323846

Булевият тип се декларира с ключовата дума **bool**. Той има две стойности, които може да приема – **true** и **false**. Стойността по подразбиране е **false**.

Символният тип представя единичен символ (16-битов). Той се декларира с ключо­вата дума **char** в езика C#. Чете се от конзолата по следния начин: char.Parse(Console.ReadLine);.

char ch=’a’;

Обект - се разбира като съвкупност от елементи и тяхното поведение. Софтуерните обекти моделират обекти от реалния свят или абстрактни концепции.

Бройни системи

* Двоична бройна система – основа – 2, цифри – 0,1

Математическото преобразуване от десетична в двоична примерно ако искаме да преобразуваме 7 в двоична ще стане каъто разделим 7:2=3 с остатък 1, след това делим 3:2=1 и остатък 1 и накрая 1:2=0 с остатък 1, събираме резултата от остатъците от долу нагоре и се получава 111.

int dec=7;

string bin=Convert.ToString(dec,2);

Console.WriteLine(“7=”+bin);

Преобразуването от двоична в десетична система става като:

111(2)=1.22+1.21+1.20=4+2+1=7(10)

Чрзе кода това става по този начин:

string bin=”111”;

int dec=Convert.ToInt32(bin,2);

Console.WriteLine(“111=”+dec);

Събиране

|  |  |  |
| --- | --- | --- |
| + | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 10 |

Умножение

|  |  |  |
| --- | --- | --- |
| \* | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |

* При **шестнайсетичните числа**  имаме за основа на бройната система числото 16.

Цифрите на бройната система са 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Преобразуването от (10) ->(16)

7(10):16=0 с остатък 7, тоест 7 от десетична е 7 в шестнайсетична.

Преобразуване с код:

int dec=7;

string hex=Convert.ToString(dec,16);

Console.WriteLine(“7=”+hex);

От (16)->(2)

A01=101000000001(2)

111001(2)=39

1А1(16) =1.162+10.161+1.160

Едномерен масив - съвкупност от еднотипни данни точно определен брой. Едномерните масиви се наричат още вектори, а двумерните – матрици.

int[] array=new int[n];

int – тип на масива

array – име на масива

new – оператор за създаване на динамичната памет

n – брой на елементите

индексиране на елементите

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 |  | ....... |  | n-1 |

Въвеждане на елементите на масив

Console.WriteLine(“n=”);

int n=int.Parse(Console.ReadLine());

int[] array=new int[n];

for(i=0;i<n;i++)

{

Console.WriteLine(“Enter: ”);

array[i]=int.Parse(Console.ReadLine());

}

Извеждане на елементите на масив

for(int i=0;i<n;i++)

{

Console.WriteLine(array[i]);

}

Сума на елементите

int S=0;

for(int=0;i<n;i++)

{

S=S+array[i];

}

Console.WriteLine(S);

Максимална стойност на елемнтите

int max=int.MinValue;

for(int i=0;i<n;i++)

{

{ max=array[i];

}

}

Console.WriteLine(“Max=”+ max);

Илюстрация на многомерен масив

0 1 2 m-1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 00 | 01 | 02 | 0m-1 |
| 1 | 10 | 11 | 12 | 1m-1 |
| n-1 | n-10 | n-11 | n-12 | n1 m-1 |

n –реда

m- колони

[nxm] – матрица на

i – променлива обхождаща редове

j – променлива обхождаща колони

Въвеждане на елементите на многомерен масив

int n=int.Parse(Console.ReadLine());

int n=int.Parse(Console.ReadLine());

Int[,] array=new int [n,m];

for(int i=0;i<n;i++)

{

for(int j=0;j<m;j++)

{

array[I,j]=int.Parse(Console.ReadLine());

}

}

Извеждане на елементите на многомерен масив

for(int i=0;i<n;i++)

{

for(int j=0;j<m;j++)

{

Console.Write(array[i,j] + ” ”);

}

Console.WriteLine();

}

**Списъци – елементи от един и същ тип**

**List<T>** - списък на елементи от какъв да е тип T. **List<T>** съдържа списък от елементи (като масив, но разширяващ се).

* **Add(element)** **–** добавя елемент към **List<T>**
* **Count –** връща броя на елементите в **List<T>**
* **Remove(element)** **–** премахва първото срещане на елемент (връща **true** / **false**)
* **RemoveAt(index) –** премахва елемент по неговия **индекс**
* **Insert(index, element)** – вмъква елемент на зададената позиция
* **Contains(element)** **–** определя дали елемента се съдържа в списъка
* **Sort() –** сортира във възходящ ред
* **Reverse() –** обръща списъка наобратно

**Въвеждане на списъци от конзолата:**

Първо, въвеждаме от конзолата дължината на списъка:

* **int n = int.Parse(Console.ReadLine());**

После, създаваме списък с размер **n** и въвеждаме елементи:

**List<int> list = new List<int>();**

**for (int i = 0; i < n; i++)**

**{**

**list.Add(int.Parse(Console.ReadLine()));**

**}**

**Изпечатване на елементите на списък:**

**Console.Write(string.Join(" ", list));**

**Предимства:**

**В списък винаги може да добавим елемент, да премахнем елемент**

**Недостатъци:**

**Че не знаем предварително бройката**

**Процесът на дебъгване на програма включва:**

Откриване на грешка

Откриване на редовете в кода, които я предизвикват

Коригиране на грешката в кода

Проверка дали грешката е отстранена и дали междувременно не са добавени нови грешки

Това е многократен и продължителен процес

**Дебъгера ни предлага:**

Стопери (breakpoints)

Възможност да следим изпълнението на кода

Средство за наблюдение на променливите по време на изпълнението на програмата

**Използване на дебъгера във Visual Studio:**

Стартиране без дебъгер: [Ctrl+F5]

Активиране на стопер: [F9]

Стартиране с дебъгер: [F5]

Проследяване на кода: [F10] / [F11]

Използване на Locals / Watches

Условни стопери

Дебъг режим след изключение

**Символни низове – представляват поредица от символи (текстове). Декларират се с ключовата дума string. Символните низове са immutable (read-only) поредици от символи. Достъпни по индекс (read-only).**

**Въвеждане на string от конзолата:**

Console.WriteLine("Въведи name");

string name = Console.ReadLine();

Console.WriteLine("name");

**Извличане на подниз:**

str.Substring(int startIndex, int length)

**Смяна на подниз:**

str.Replace(match, replacement)– замества всички съвпадения

В програмирането абстрактната структура данни "речник" представлява съвкупност от наредени двойки (ключ, стойност), заедно с дефинирани операции за достъп до стойностите по ключ.

Dictionary<string, string> phonebook = new Dictionary<string, string>();

|  |  |
| --- | --- |
| Key | Value |
| Ilker | Man |
| Apostolova | Woman |

**Нечетни срещания - Пример**

string input = Console.ReadLine().ToLower();

string[] words = input.Split(' ');

Dictionary<string, int> counts = new Dictionary<string, int>();

foreach (var word in words)

if (counts.ContainsKey(word))

counts[word]++;

else counts[word] = 1;

var results = new List<string>();

foreach (var pair in counts)

Console.WriteLine(string.Join(", ", results));

Ключа стои отляво в случая стрингова променлива, а стойността целочислена променлива.

**Въвеждане на речник**

Dictionary<string, string> phonebook = new Dictionary<string, string>();

phonebook["JohnSmith"]="+1-555-8976";  
phonebook["Lisa Smith"] = "+1-555-1234";

phonebook["Sam Doe"] = "+1-555-5030";

phonebook["Ivan"] = "+359-899-555-592";

phonebook["Ivan"] = "+359-2-981-9819"; // Заменяне

phonebook.Remove("John Smith");

foreach (var pair in phonebook)

Console.WriteLine("{0} --> {1}",

pair.Key, pair.Value);

**Методи за работа**

* **ContainsKey()** – проверяваме дали даден ключ съществува в речника (бърза операция)
* **ContainsValue()** – проверяваме дали дадена стойност съществува в речника (бавна операция)
* **TryGetValue() –** проверяваме дали даден ключ съществува в речника и отпечатва стойността му

Dictionary<string, string> phonebook – декларация

Dictionary<string, string> phonebook = new Dictionary<string, string>(); - дефиниция