Тема 1

Изчисления, линейни, разклонени и циклични алгоритми

Денко 1

1. Дефинира понятия:
   - Програмиране
   - Език за програмиране
   - Среда за разработка IDE
   - Компилация и интерпретация
   - Приложение – създаване и зареждане на проект, стартиране на проект
2. Описва понятието променлива, сравнява типовете променливи
   - Аритметични и логически оператори, изрази
   - Разработва програми / програмни фрагменти с аритметични и логически изрази с участието на променливи и числа
3. Описва условни конструкции
   - Сравнява условен конструктор if/else и оператор за многовариантен избор switch
   - Определя кои фрагменти от код се изпълняват, колко пъти и в какъв ред
4. Циклични оператори
   - Различава операторите за цикъл
   - Определя кои фрагменти от код се изпълняват, колко пъти и в какъв ред при оператори за цикъл
5. Същност на подпрограмите(функции/методи)
   - Видове параметри
   - Видове връщана стойност
   - Пример за предимства от изпълнението на методи
6. 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.

   Задача 1. Да се състави програма, която въвежда цяло число n и последователност от n на брой цели числа, всяко на отделен ред. Програмата да намира:
   - сумата от всички числа;
   - средно-аритметично от положителните числа;
   - броя на числата, които са кратни на 3.

1. Дефинира понятия

Програмиране – дават се команди на компютъра какво да прави, като командите се подреждат една след друга образувайки компютърна програма.

Език за програмиране - **компютърните програми** представляват **поредица от команди**, които се изписват на предварително избран **език за програмиране,** който има синтаксис и семантика. Изпълнението на компютърните програми може да се реализира с компилатор или с интерпретатор.

Среда за програмиране - Средата за програмиране (Integrated Development Environment - IDE, интегрирана среда за разработка) е съвкупност от традиционни инструменти за разработване на софтуерни приложения. В средата за разработка пишем код, компилираме и изпълняваме програмите.

Средите за разработка интегрират в себе си **текстов редактор** за писане на кода, **език за програмиране**, **компилатор** или **интерпретатор** и **среда за изпълнение** за изпълнение на програмите, **дебъгер** за проследяване на програмата и търсене на грешки, **инструменти за дизайн на потребителски интерфейс** и други инструменти и добавки.

Компилация - превежда кода от програмен език на **машинен код**, като за всяка от конструкциите (командите) в кода избира подходящ, предварително подготвен фрагмент от машинен код, като междувременно проверява за грешки текста на програмата. При компилируемите езици за програмиране **компилирането на програмата се извършва задължително преди нейното изпълнение** и по време на компилация се откриват синтактичните грешки (грешно зададени команди).

Интерпретаторът е "програма за изпълняване на програми", написани на някакъв програмен език. Той изпълнява командите на програмата една след друга, като разбира не само от единични команди и поредици от команди, но и от другите езикови конструкции (проверки, повторения, функции и т.н.). Поради липса на предварителна компилация, при интерпретеруемите езици **грешките се откриват по време на изпълнение**, след като програмата започне да работи, а не предварително.

Приложение – създаване и зареждане на проект, стартиране на проект.

Създаване на конзолна програма – стартираме Visual Studio, след това създаваме конзолно приложение следвайки стъпките: [File] → [New] → [Project] → [Visual C#] → [Windows] → [Console Application]. Visual Studio създава празна C# програма. Стартиране на програма - натискаме [Ctrl + F5]. Ако няма грешки, програмата ще се изпълни. Резултатът ще се изпише на конзолата.

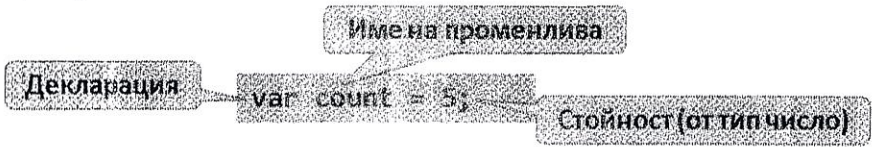
2. Описва понятието променлива, сравнява типовете променливи

Променливите са именувани области от паметта, които пазят данни от определен тип, например число или текст. Всички данни се записват в компютърната памет (RAM памет) в променливи.

Характеристики на даден тип данни:

- име
- размер
- стойност по подразбиране

Пример:



След като се декларира, че една променлива е от даден тип, за нея автоматично се определят:

- *множество от допустими стойности*
- *необходима оперативна памет*
- *приложима операция и вградени функции*

Типовете данни, които се състоят само от една компонента, се наричат скаларни (още примитивни типове данни)

Сравнение на типовете данни:

Целочислени типове

- **Приложение**

В програмирането променливите от целочислен тип се използват обикновено за броячи в цикли, селектори в оператор switch, индекси на елементи на масив, когато трябва да се опишат величини, които са точни по своята същност.

Променливи от целочислен тип се дефинират по следния начин:

```
int f, g;      // променлива от цял тип със знак, заема 32 бита от ОП
short i,j;     // променлива от цял тип със знак, заема 16 бита от ОП
long d,s;      // променлива от цял тип със знак, заема 64 бита от ОП
```

- **Операции с целочислени данни:**

Аритметични операции

Вид на операцията	**Означаване**
Събиране	+
Изваждане	-
Умножение	*
Целочислено деление	/
Остатък от деление	%

Целите данни могат да бъдат сравнявани с операциите за сравнение:

Вид на операцията	**Означаване**
Равно	==
Различно	!=
По-малко	<
По-малко или равно	<=
По-голямо	>
По-голямо или равно	>=

Присвояването в C# **също е операция.** Тя се означава чрез знака "=" от ляво на който стой променлива, а отдясно - **израз,** чиято стойност се присвоява на тази променлива.

Пример:

```
int f = 7*4;      // f е със стойност 28
int g = f/3;      // g е със стойност 9, въпреки че се получава дробно число, g ще присвои цяло число, защото е от тип int
```

Задаване на първоначална стойност на дадена променлива се нарича **инициализация** на тази променлива. Инициализацията също се извършва с **оператора за присвояване.** В много случай се налага промяна на стойността на променливата.

Например увеличаване на променлива f с 10. Присвояването f=f+10 осъществява увеличението на f с 10. В C# има кратка форма на операцията и това е операцията "+=".

Операция	Пълна	Кратка	f преди операцията	f след операцията
+=	f=f+10	f+=10	**28**	38
-=	f=f-10	f-=10	**28**	18
*=	f=f*10	f*=10	**28**	280
/=	f=f/10	f/=10	**28**	2
%=	f=f%10	f%=10	**28**	8
++	f=f+1	f++	**28**	29
--	f=f-1	f--	**28**	27

Реални типове (приближени)

Тип	Цифри след десетичната запетая	Формат
float	7	4 байта – суфикс f
double	15-16	8 байта – суфикс d

Данните от приближен тип, също както и целочислените данни могат да бъдат сравнявани с операциите за сравняване ==,!=,<,<=,>,>=, като при това винаги трябва да се има предвид **неточно представяне** на тези типове данни в ОП.

- **Вградени функции, чийто резултат е от реален тип**

Вградени функции, намиращи се в **класа Math**, които работят с данни от приближен тип.

Функция	Резултат	Пример
Abs(x)	Абсолютна стойност	Math.Abs(-23.4) -> 23.4
Pow(x,y)	x^y	Math.Pow(2,3) ->2^3->8.0
Sqrt(x)	Корен квадратен x>=0	Math.Sqrt(9) ->3.0
Sin(x)	sin x, x в радиана	Math.Sin(Math.PI/2)->1.0
Cos(x)	cos x, x в радиана	Math.Cos(Math.PI/2)->0
Ceiling(x)	Връща цяло число от тип double, което е >=x	Math.Ceiling(-23.46) ->-23.0 Math.Ceiling(23.46) ->-24.0
Floor(x)	Връща цяло число от тип double, което е <=x	Math.Floor(-23.46) ->-24.0 Math.Floor(23.46) ->-23.0

Тип с фиксирана десетична точка

Тип	Цифри след десетичната запетая	Формат
decimal	28-29	16 байта – суфикс m

- **Приложение**

Използва се удачно за парични изчисления. Константите задължително се дефинират със суфикс m.

Пример:

decimal leva = 2345.589m;

Логически(булев) тип

Величините от този тип имат две стойности: **true**(истина) и **false**(лъжа). Типът bool заема 1 байт от ОП.

Операциите за сравнение (==, !=, <, <=, >, >=) дават резултат от тип bool.

Променливи от тип bool се дефинират например така:

bool b, c=false;

- **Приложение**

Булевите данни служат главно за управление на алгоритмичния процес.

Операциите, които могат да се извършат с данни от булев тип са едноаргументни и двуаргументни.

1) Едноаргументна операция е логическо отрицание (!).
2) Двуаргументни операции:
   and(&&) – логическо умножение(конюнкция)
   or(||) – логическо събиране(дезюнкция)
   xor(^) – изключващо или(сума по модул 2)

b	c	!b	b&&c	b||c	b^c
false	false	true	false	false	false
false	true	true	false	true	true
true	false	false	false	true	true
true	true	false	true	true	false

Знаков тип

- Представлява символ.
- На всеки символ може да се съпостави целочислен код. Данни от тип char съдържат символни кодове.
- Стойност по подразбиране: '\0' ;
- В C#, символите се кодират използвайки Unicode стандарта.

Променливите от знаков тип се декларират с ключовата дума **char**.

Пример:

char cw, cz='d ';

Константите от този тип се записват заградени в апострофи (').

- **Операции с знакови данни**.

Величините от този тип могат да бъдат сравнявани посредством операциите за сравнение. Това става на базата на машинните кодове на знаковете.

Пример:

'A'< 'B' ще даде резултат true, тъй като машинния код на **'A'** е 65, а на **'B'**-66.

3. Описва условни конструкции

3.1 If конструкция

В програмирането често **проверяваме дадени условия** и извършваме различни действия, според резултата от проверката.

Условието се проверява

- Ако е вярно, се изпълнява израза или изразите.
- Ако не е вярно, изразите се пропускат.

Условието може да бъде:

- Булева променлива.
- Булев логически израз.
- Израз със сравнения.

Синтаксис:

```
if (условие)
{
<израз1>;
<израз2>;
...
}
```

Пример:

Въвеждаме оценка в конзолата и проверяваме дали тя е отлична (**≥ 5.50**).

```
var grade = double.Parse(Console.ReadLine());
if (grade >= 5.50)
{
    Console.WriteLine("Excellent!");
}
```

3.2 If-else конструкция

Условието се проверява

- Ако е вярно, се изпълнява израз или група от изрази
- Ако не е вярно, се изпълнява друга група от изрази

Синтаксис:

```
if (<условие>)
{
<израз1>;
<израз2>;
...
}
else
{
<израз3>;
}
```

Пример:

```
var grade = double.Parse(Console.ReadLine());
if (grade >= 5.50)
{
    Console.WriteLine("Excellent!");
}
else
{
    Console.WriteLine("Not excellent.");
}
```

Когато имаме само една команда в тялото на if конструкцията, можем да пропуснем къдравите скоби, обозначаващи тялото на условния оператор. Когато искаме да изпълним блок от код (група команди), къдравите скоби са задължителни. В случай че ги изпуснем, ще се изпълни само първият ред след if клаузата.

3.3 Switch конструкция

Конструкцията **switch-case** работи като поредица **if-else** блокове. Когато работата на програмата ни **зависи от стойността на една променлива можем да използваме условната конструкция switch.**

Проверява се, стойността на кой от случаите(case) съвпада със стойността на селекторната променлива.

- При съвпадение се изпълнява кодът, съдържащ се в избрания case блок.
- При ненамиране на съвпадение се изпълнява блокът default

Синтаксис:

switch (<селектор>)
{
case <една от възможните стойности на селектора>: <израз, който трябва да се изпълни, ако селектора е равен на тази си възможна стойност.>; break;
.....
default: <израз, който трябва да се изпълни, ако никой друг от case-овете не е срещнат.> break;
}

Пример:

```
int day = int.Parse(Console.ReadLine());
switch (day)
{
    case 1: Console.WriteLine("Monday"); break;
    case 2: Console.WriteLine("Tuesday"); break;
    …
    case 7: Console.WriteLine("Sunday"); break;
    default: Console.WriteLine("Error!"); break;
}
```

Вложени проверки

Доста често програмната логика налага използването на if или if-else конструкции, които се съдържат една в друга. Те биват наричани вложени if или if-else конструкции.

Синтаксис:

```
if (<условие>)
{
  if(<условие>)
  {
   <израз>;
  }
  else if(<условие>)
  {
   <израз>;
  }
  else
  {
   <израз>;
  }
}
else
{
  <израз>;
}
```

4. Циклични оператори

4.1 Цикли с условие

○ while цикъл

Синтаксис:

```
while (<условие>) {
  <израз>;
}
```

- Условието е булеви израз, който се проверява периодично.
- Условието определя, кога цикълът да се прекрати.

Пример:

```
int i = 0;
while (i < 5)
{
  Console.WriteLine(i);
  i++;
}
```

В примера кодът в цикъла ще се изпълнява отново и отново, докато променлива (i) стане по-малка от 5.

○ do ... while цикъл

Синтаксис:

```
do {
  <израз>;
} while (<условие>);
```

- Изразите в тялото на цикъла се изпълняват многократно, докато не се наруши условието.
- Изпълнява се поне веднъж.

Пример:

```
int i = 0;
do
{
  Console.WriteLine(i);
  i++;
}
while (i < 5);
```

Цикълът винаги ще се изпълнява поне веднъж, дори ако условието е невярно, тъй като кодовият блок се изпълнява преди условието да бъде тествано.

4.2 Цикли с брояч

Синтаксис:

```
for (<инициализация>; <проверка>; <актуализация>)
{
<израз>;
}
```

Състои се от:

- **Инициализация**. Представлява деклариране и инициализиране на итерираща променлива.
- **Проверка**. Представлява булев израз, който се проверява на всяка итерация.
- **Актуализация**, в която се посочва, как итериращата променлива се променя с всяка итерация.

- **Тяло на цикъла**, което съдържа групата изрази, които трябва да се изпълняват многократно.

Пример:

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine(i);
};
```

4.3 Вложени цикли

Синтаксис:

```
// външен цикъл
for (<инициализация>; <проверка>; <актуализация>) {
        // вътрешен цикъл
        for (<инициализация>; <проверка>; <актуализация>)
        {
        ...
        }
    ...
    }
}
```

„Вътрешният цикъл" ще бъде изпълнен веднъж за всяка итерация на „външния цикъл"

Пример:

```
for (int i = 1; i <= 2; ++i)
{
  Console.WriteLine("Outer: " + i);

  // Inner loop
  for (int j = 1; j <= 3; j++)
  {
    Console.WriteLine(" Inner: " + j);
  }
}
```

5. Същност на подпрограмите(функции/методи)

5.1 Метод

- Методът е малко парче код, което решава конкретен проблем.
- Методът има име и може да бъде извикан от други места в програмата.
- Може да приема параметри и да връща резултат.
- Създадени да моделират математични функции, с огромно значение в структурното и обектно програмиране.
- Синоними са (функция, процедура), в C# се наричат методи, поради обектната насоченост на езика.

Структура на метод:

```
<модификатор> <тип_данни за резултат/void> <име на метода>(<параметър1>, <параметър2>,...)
{
  [<тип_данни за резултат> <параметър за резултат>;]
  <израз>;
  <израз>;
  [return <параметър за резултат>/<израз>;]
```

}

- Ако методът трябва да връща стойност, то нейният тип се указва при декларирането.
- Ако не се изисква методът да връща стойност, се използва ключовата дума void, вместо тип данни.

5.2 Видове параметри

Параметрите са променливи.

Те могат:

• да се съдържат в дефиницията на метода;

• да се съдържат в тялото на метода;

Аргументи, наричаме стойностите на параметричните променливи. Те определят поведението на функцията.
Параметрите могат да имат стойности по подразбиране.
Параметрите могат да бъдат референции, към вече декларирани променливи (out и ref параметри).

5.3 Извикване на методи

Методите могат да се извикват поименно, от друг метод в същия клас или от друг клас, в зависимост от модификатора за достъп.

- Ако методът връща стойност, тя може да бъде присвоена на променлива от същият тип, какъвто е изходния на метода.
- Ако методът не връща стойност, той просто може да се извика поименно на един ред.

Извикване на метод не връщаш стойност:

```
static void MyMethod()
{
  Console.WriteLine("I just got executed!");
}

static void Main(string[] args)
{
  MyMethod();
}
```

Извикване на метод връщаш стойност:

```
class Program
{
  static int MyMethod(int num1, int num2)
  {
    return num1+num2;
  }

  static void Main(string[] args)
  {
    int sum = MyMethod(2,5);
    Console.WriteLine(sum);
  }
}
```