**Тема 1**

**Изчисления, линейни,**

**разклонени и циклични алгоритми**

**1.**

Да "програмираме" означава да даваме команди на

компютъра какво да прави

▪ Командите се подреждат една след друга

▪ Така те образуват "компютърна програма"

▪ Компютърната програма е поредица от команди

▪ Програмите се пишат на език за програмиране

▪ Например C#, Java, JavaScript, Python, PHP, C, C++, …

▪ Използва се среда за програмиране (например Visual Studio)

За да програмирате, ви трябва среда за разработка

▪ Integrated Development Environment (IDE)

▪ За C# →

▪ Visual Studio за Windows

▪ MonoDevelop за Linux / Max OS X

▪ за Java → IntelliJ IDEA

▪ за PHP → PHP Storm

Алтернативна среда за разработка (online)

▪ C# – .NET Fiddle - https://dotnetfiddle.net/

▪ Java – https://www.compilejava.net/

▪ JavaScript – може директно в конзолата на браузър

**Компилаторът:**  е компютърна програма, която превежда (компилира) даден компютърен изходен код в семантично отговарящ код на език от (обикновено) по-ниско ниво. Целевият език може да бъде машинен език или асемблерен език за конкретен процесор или процесорна фамилия, както и междинен език за конкретна виртуална машина (например байткод за виртуална машина на Java) или друг език от високо ниво. Когато се компилира до машинен език, крайният продукт е изпълнима програма или обектен код.

**Интерпретатор:** Най -общо **интерпретаторът** представлява програма, която изпълнява друга програма. Интерпретаторъ извършва последователен анализ на командите от изходния код(написата вече програма, която искаме компютърът да изпълни), непосредствено ги превежда на машинен език и изпълнява т.е. не превежда целият изходен код само веднъж както прави компилатора, а го прави всеки път когато се извиква програмата. Основният недостатък на интерпретаторите е, че когато дадена програма се интерпретира обикновено работи по-бавно от колкото би работила ако е била компилирана.

**2.**

Променливата е контейнер на информация, който може да променя стойността си. Тя осигурява възможност за:

- запазване на информация;

- извличане на запазената информация;

- модифициране на запазената информация.

Програмирането на C# е свързано с постоянно използване на променливи, в които се съхраняват и обработват данни.

**Съответствие на типовете в C# и в .NET Framework**

Примитивните типове данни в C# имат директно съответствие с типове от общата система от типове (CTS) от .NET Framework. Например типът int в C# съответства на типа System.Int32 от CTS и на типа Integer в езика VB.NET, a типът long в C# съответства на типа System.Int64 от CTS и на типа Long в езика VB.NET. Благодарение на общата система на типовете (CTS) в .NET Framework има съвместимост между различните езици за програмиране (като например C#, Managed C++, VB.NET и F#). По същата причина типовете int, Int32 и System.Int32 в C# са всъщност различни псевдоними за един и същ тип данни – 32 битово цяло число със знак

**Категории оператори**

Следва списък с операторите, разделени по категории:

|  |  |
| --- | --- |
| **Категория** | **Оператори** |
| аритметични | **-, +, \*, /, %, ++, --** |
| логически | **&&, ||, !, ^** |
| побитови | **&, |, ^, ~, <<, >>** |
| за сравнение | **==, !=, >, <, >=, <=** |
| за присвояване | **=, +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>=** |
| съединяване на символни низове | **+** |
| за работа с типове | **(type), as, is, typeof, sizeof** |
| други | **., new, (), [], ?:, ??** |
|  |  |

**Оператори според броя аргументи**

Операторите могат да се разделят на типове според броя на аргументите, които приемат:

|  |  |
| --- | --- |
| **Тип оператор** | **Брой на аргументите (операндите)** |
| едноаргументни (unary) | приема един аргумент |
| двуаргументни (binary) | приема два аргумента |
| триаргументни (ternary) | приема три аргумента |

Всички двуаргументни оператори в C# са ляво-асоциативни, т.е. изра­зите, в които участват се изчисляват от ляво на дясно, освен операторите за присвояване на стойности. Всички оператори за присвояване на стойности и условните оператори **?:** и **??** са дясно-асоциативни (изчисляват се от дясно на ляво). Едноаргументните оператори нямат асоциативност.

Някой оператори в C# извършват различни операции, когато се приложат върху различен тип данни. Пример за това е операторът +. Когато се използва върху числени типове данни (**int**, **long**, **float** и др.), операторът извършва операцията математическо събиране. Когато обаче използваме оператора върху символни низове, той слепва съдържанието на двете про­менливи / литерали и връща новополучения низ.

**Оператори – пример**

Ето един пример за използване на оператори:

|  |
| --- |
| int a = 7 + 9;  Console.WriteLine(a); // 16    string firstName = "Dilyan";  string lastName = "Dimitrov";    // Do not forget the interval between them  string fullName = firstName + " " + lastName;  Console.WriteLine(fullName); // Dilyan Dimitrov |

Примерът показва как при използването на оператора **+** върху числа той връща числова стойност, а при използването му върху низове връща низ.

**3.**

Условната конструкция switch (множествено разклонение в някои [езици за програмиране](https://bg.wikipedia.org/wiki/%D0%95%D0%B7%D0%B8%D0%BA_%D0%B7%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D0%BD%D0%B5)) се използва за избор измежду списък с възможности. Конструкцията сравнява дадена стойност с определени [константи](https://bg.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D0%B0) и въз основа на съвпадението с някоя от тях, предприема дадено действие.

**switch** (селектор)

{

**case** стойност-1: код за изпълнение; **break**;

**case** стойност-2: код за изпълнение; **break**;

**case** стойност-3: код за изпълнение; **break**;

**case** стойност-4: код за изпълнение; **break**;

*// …*

**default**: код за изпълнение; **break**;

}

Конструкцията switch-case избира измежду части от [програмен код](https://bg.wikipedia.org/wiki/%D0%98%D0%B7%D1%85%D0%BE%D0%B4%D0%B5%D0%BD_%D0%BA%D0%BE%D0%B4) на базата на изчислената стойност на определен [израз](https://bg.wikipedia.org/wiki/%D0%98%D0%B7%D1%80%D0%B0%D0%B7_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)#%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0). Този израз най-често е целочислен, но може да бъде и от тип [string](https://bg.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7) или [char](https://bg.wikipedia.org/wiki/Char). Стой­ността на селектора трябва задължително да бъде изчис­лена преди да се сравнява със стойностите вътре в switch конструк­цията. Етикетите (case) не трябва да имат една и съща стойност. При намиране на съвпадение на селектора с някоя от case стойностите, switch-case конструкцията изпълнява кода след съответния case. При липса на съвпадение, се изпълнява default конструкцията, когато такава съществува. Всеки case етикет, както и етикетът по подразбиране (default), трябва да завършват с ключовата дума break, която приключва работата на switch-case конструкцията, след като е намерено съвпадение и е изпълнен съответния код.

В [C Sharp](https://bg.wikipedia.org/wiki/C_Sharp) имаме възможността да използваме множество етикети, когато те трябва да изпълняват един и същи код. При този начин на записване, когато намерим съвпадение, тъй като след съответния case етикет липсва код за изпълнение и break оператор, ще се изпълни следващия срещнат код. Ако такъв липсва ще се изпълни default конструкцията.

int number = 6;

**switch** (number)

{

**case** 1:

**case** 4:

**case** 6:

**case** 8:

**case** 10:

Console.WriteLine("Числото не е просто!"); **break**;

**case** 2:

**case** 3:

**case** 5:

**case** 7:

Console.WriteLine("Числото е просто!"); **break**;

**default**:

Console.WriteLine("Не знам какво е това число!"); **break**;

}

**Нека разгледаме следния пример, за да покажем в действие как работи if-else конструкцията:**

|  |
| --- |
| static void Main()  {        int x = 2;        if (x > 3)        {              Console.WriteLine(**"x е по-голямо от 3"**);        }        else        {              Console.WriteLine(**"x не е по-голямо от 3"**);        }  } |

Програмният код може да бъде интерпретиран по следния начин: ако x>3, то резултатът на изхода е: "**x е по-голямо от 3**", иначе (**else**) резултатът е: "**x не е по-голямо от 3**". В случая, понеже x=2, след изчислението на булевия израз ще бъде изпълнен операторът от **else**-конструкцията. Резултатът от примера е:

|  |
| --- |
| **x не е по-голямо от 3** |

**Как работи switch-case конструкцията?**

Конструкцията **switch-case** избира измежду части от програмен код на базата на изчислената стойност на определен израз (най-често цело­числен). Форматът на конструкцията за избор на вариант е следният:

|  |
| --- |
| switch (селектор)  {            case целочислена-стойност-1: конструкция; break;            case целочислена-стойност-2: конструкция; break;            case целочислена-стойност-3: конструкция; break;            case целочислена-стойност-4: конструкция; break;            // …            default: конструкция; break;  } |

Селекторът е израз, връщащ като резултат някаква стойност, която може да бъде сравнявана, например число или **string**. Операторът **switch** сравнява резултата от селек­тора с всяка една стойност от изброените в тялото на switch конструкцията в **case** етикетите. Ако се открие съвпа­дение с някой **case** етикет, се изпълнява съответната конструкция (проста или съставна). Ако не се открие съвпадение, се изпълнява **default** конструкцията (когато има такава). Стой­ността на селектора трябва задължително да бъде изчис­лена преди да се сравнява със стойностите вътре в **switch** конструк­цията. Етикетите не трябва да имат една и съща стойност.

Както се вижда, че в горната дефиниция всеки **case** завършва с оператора **break**, което води до преход към края на тялото на **switch** конструкцията. C# компилаторът задължително изисква да се пише **break** в края на всяка **case**-секция, която съдържа някакъв код. Ако след дадена **case**-конструк­ция липсва програмен код,**break** може да бъде пропуснат и тогава изпълнението преминава към следващата **case**-конструкция и т.н. до срещането на оператор **break**. След **default** конструкцията,**break** е задължителен.

Не е задължително **default** конструкцията да е на последно място, но е препоръчително да се постави накрая, а не в средата на **switch** конструкцията.

**Правила за израза в switch**

Конструкцията **switch** е един ясен начин за имплементиране на избор между множество варианти (тоест, избор между няколко различни пътища за изпълнение на програмния код). Тя изисква селектор, който се изчислява до някаква конкретна стойност. Типът на селектора може да бъде цяло число, **string** или **enum**. Ако искаме да използваме, например, низ или число с плаваща запетая като селектор, това няма да работи в **switch** конструкция. За нецелочислени типове данни трябва да използваме пос­ледователност от **if** конструкции.

**4.**

Извеждане на масив на конзолата:

За извеждане на елементите на масив може да се ползва цикъл „for“

Разделяне на елементите с интервал или нов ред

**5.**

**Какво е "метод"?**

До момента установихме, че при **писане** на код на програма, която решава дадена задача, ни **улеснява** това, че **разделяме** задачата на **части**. Всяка част отговаря за **дадено действие** и по този начин не само ни е **по-лесно** да решим задачата, но и значително се подобрява както **четимостта** на кода, така и проследяването за грешки.

Всяко едно парче код, което изпълнява дадена функционалност и което сме отделили логически, може да изземе функционалността на метода. Точно това представляват **методите – парчета код, които са именувани** от нас по определен начин и които могат да бъдат **извикани** толкова пъти, колкото имаме нужда.

Един метод може да бъде извикан толкова пъти, колкото ние преценим, че ни е нужно за решаване на даден проблем. Това ни **спестява** повторението на един и същи код няколко пъти, както и **намалява** възможността да пропуснем грешка при евентуална корекция на въпросния код.

**Вложени методи (локални функции)**

Graphical user interface, text

Description automatically generated

**Какво е локална функция?**

Виждаме, че в този код, в главния метод **Main()** има **друг** деклариран метод **Result()**. Такъв **вложен** метод се нарича **локална** функция и е нововъведение в C# 7. Локалните функции могат да се декларират във всеки един друг метод. Когато C# компилаторът компилира такива функции, те биват превърнати в private методи. Тъй като разликата между **public** и **private** методи се изучава на по-късен етап, за момента ще отбележим, че **private** методите могат да се използват само в класа, в който са декларирани. Програмите, които пишем на това ниво, използват само един клас, затова и приемаме, че можем да използваме вложените методи без каквито и да било притеснения

**Защо да използваме методи?** ▪ Програмирането става по-обозримо ▪ Разделяме големите задачи на малки части ▪ По-оптимална организация на програмата ▪ Подобрява се четимостта на кода ▪ Улеснява разбирането на кода ▪ Избягват се повторенията в кода ▪ Улеснява поддръжката на кода ▪ Повторно използване на код ▪ Използваме методите няколко пъти