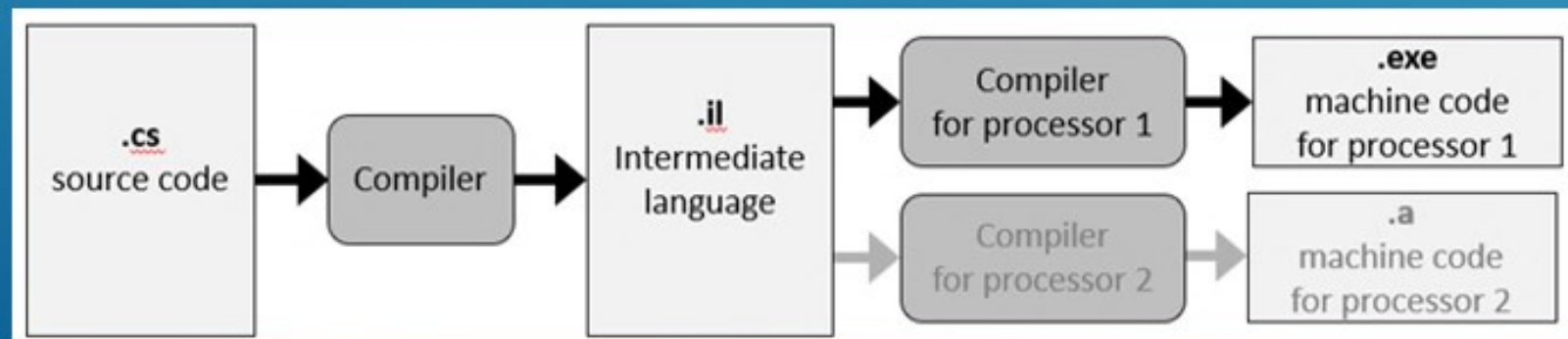
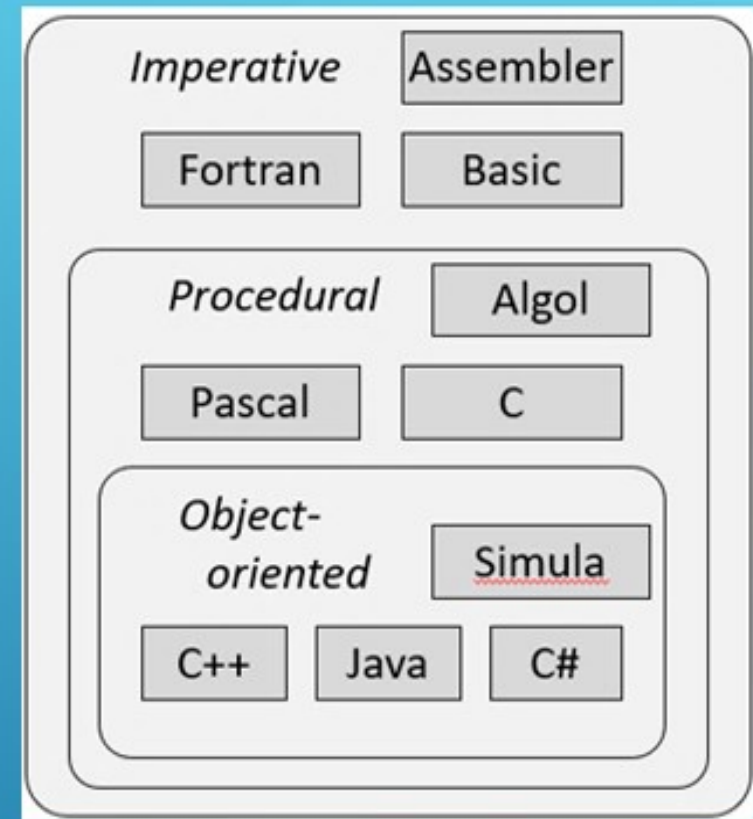


# MICROSOFT C#

## Fundamentos

# INTRODUÇÃO

- C# é uma linguagem de programação da .NET Framework:
  - Multiparadigma;
  - Fortemente tipada;
  - *Case sensitive*;
  - O código fonte é inicialmente compilado para um linguagem intermédia e só depois compilado para o ambiente onde é executado.



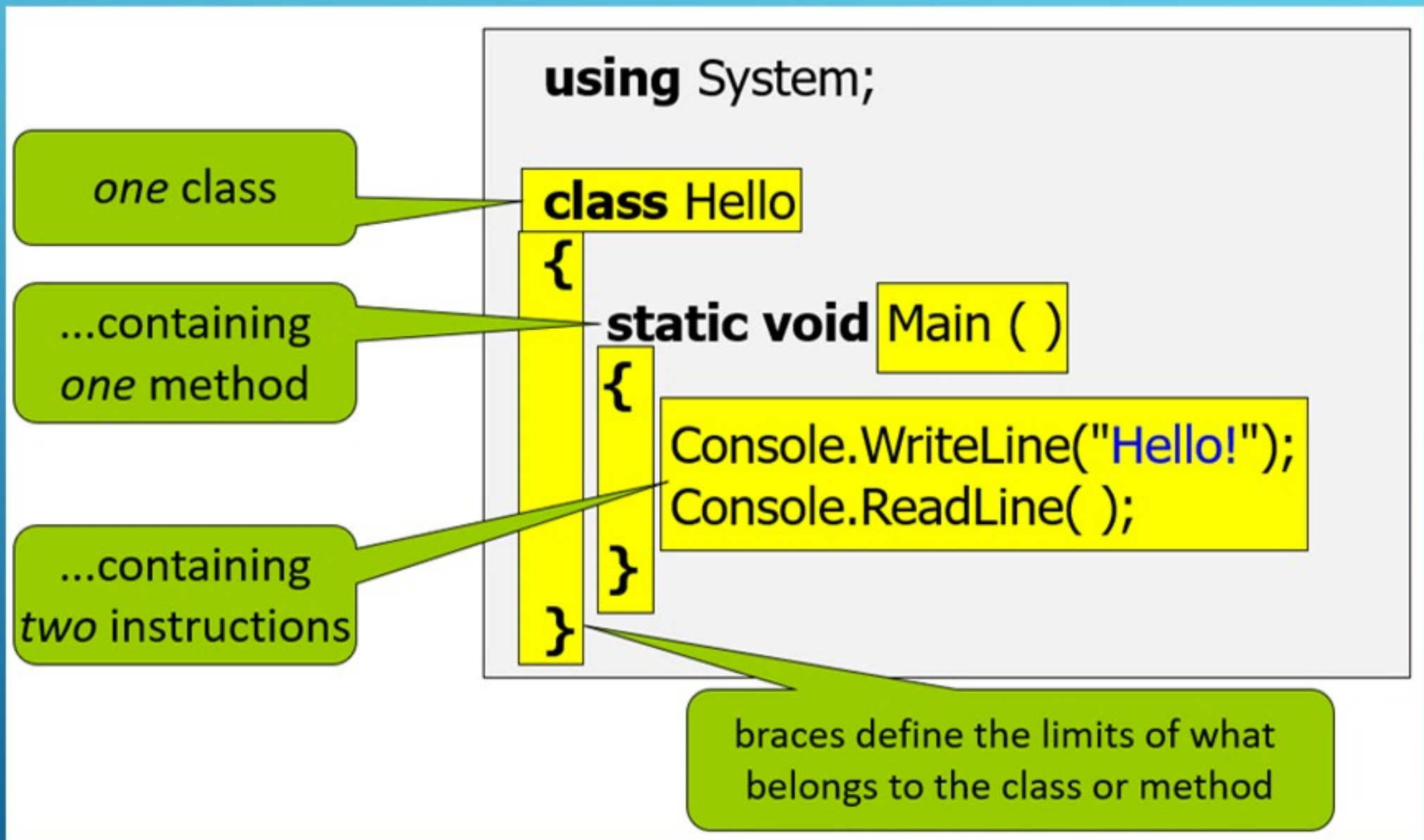
# INTRODUÇÃO

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/>

- Semelhanças com o C, C++ e java;

in	int	interface	internal				
is	lock	long	namespace	add	alias	ascending	
new	null	object	operator	async	await	by	
out	override	params	private	descending	dynamic	equals	
protected	public	readonly	ref	from	get	global	
return	sbyte	sealed	short	group	into	join	
sizeof	stackalloc	static	string	let	nameof	on	
struct	switch	this	throw	orderby	partial (type)	partial (method)	
true	try	typeof	uint	remove	select	set	
ulong	unchecked	unsafe	ushort	value	var	when (filter condition)	
using	using static	virtual	void				
volatile	while	abstract	as	base	bool	where (query clause)	yield
		break	byte	case	catch		
		char	checked	class	const		
		continue	decimal	default	delegate		
		do	double	else	enum		
		event	explicit	extern	false		
		finally	fixed	float	for		
		foreach	goto	if	implicit		

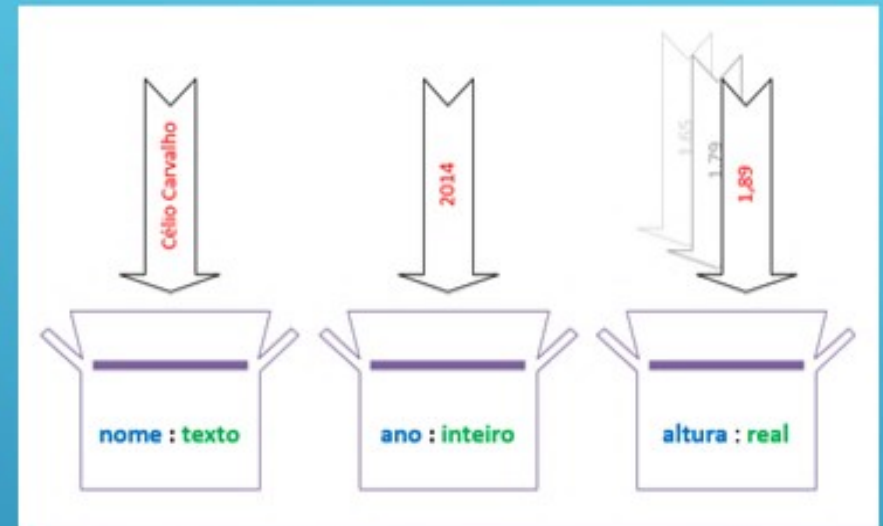
# EXEMPLO DE UM PROGRAMA





# VARIÁVEIS

- Uma variável pode ser entendida como uma caixa (**lilás**) que consegue guardar um valor (**vermelho**) e cujo rótulo é constituído por um nome (**azul**) (que identifica univocamente o conteúdo ou seja o valor), e um tipo (**verde**) (que especifica o tipo de dados que consegue guardar (numérico, texto, etc.)).



- O conteúdo dessa caixa, ou seja o valor da variável (**vermelho**), pode ser alterado sempre que necessário. No entanto, o novo valor tem que ser sempre do tipo de dados (**verde**), inscrito inicialmente no rótulo da caixa (variável) e, quando é alterado, o valor anterior é perdido. Ao contrário do valor (**vermelho**), o nome (**azul**) e o tipo de dados (**verde**) mantêm-se fixos desde a criação da variável (**lilás**), até ao momento que deixa de fazer falta e é destruída (descartada da memória).

# VARIÁVEIS

- Declarar uma variável, é o mesmo que dizer ao computador que deve alocar (reservar) espaço em memória para guardar valores;
- O nome da variável deve ser sugestivo, ou seja, escolhido de forma a representar intuitivamente o seu conteúdo;
- Não pode ser nenhuma das palavras reservadas da linguagem e o caracter espaço é proibido;
- O primeiro caracter deve ser uma letra ou um *underscore*;
- No C# o tamanho máximo do identificador da variável é de 132 caracteres;
- Em C# devem ser respeitadas as convenções assumidas pela comunidade de programadores – *Common Language Specification* (CLS).



# CONSTANTES

- Regras similares às variáveis mas o conteúdo é atribuído no momento de compilação e não pode ser alterado durante o tempo de execução;
- São habitualmente utilizadas quando há um valor utilizado repetidamente no código mas que não é alterado em execução;
- Uma constante **const** tem que ser inicializada no momento de declaração;
- Em C# existe também a hipótese de definir uma variável **readonly**, similar à constante, mas a sua atribuição pode ser efetuada em *runtime*;

# TIPOS DE DADOS

<https://www.tutorialsteacher.com/csharp/csharp-data-types>

Reserved Word	.NET Type	Type	Size (bits)	Range (values)
byte	Byte	Unsigned integer	8	0 to 255
sbyte	SByte	Signed integer	8	-128 to 127
short	Int16	Signed integer	16	-32,768 to 32,767
ushort	UInt16	Unsigned integer	16	0 to 65,535
int	Int32	Signed integer	32	-2,147,483,648 to 2,147,483,647
uint	UInt32	Unsigned integer	32	0 to 4294967295
long	Int64	Signed integer	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	UInt64	Unsigned integer	64	0 to 18,446,744,073,709,551,615
float	Single	Single-precision floating point type	32	-3.402823e38 to 3.402823e38
double	Double	Double-precision floating point type	64	-1.79769313486232e308 to 1.79769313486232e308
decimal	Decimal	Precise fractional or integral type that can represent decimal numbers with 29 significant digits	128	(+ or -)1.0 x 10e-28 to 7.9 x 10e28
char	Char	A single Unicode character	16	Unicode symbols used in text
bool	Boolean	Logical Boolean type	8	True or False
object	Object	Base type of all other types		
string	String	A sequence of characters		
DateTime	DateTime	Represents date and time		0:00:00am 1/1/01 to 11:59:59pm 12/31/9999








# CONVERSÕES IMPLÍCITAS DE DADOS

<https://www.tutorialsteacher.com/csharp/csharp-data-types>

Implicit Conversion From	To
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, or decimal
ushort	int, uint, long, ulong, float, double, or decimal
int	long, float, double, or decimal.
uint	long, ulong, float, double, or decimal
long	float, double, or decimal
ulong	float, double, or decimal
char	ushort, int, uint, long, ulong, float, double, or decimal
float	Double

# OPERADORES MATEMÁTICOS

Operator	Description	Example (x=100, y=50)
	Used to add two values or operands	$x+y=150$
	Used to subtract second value from the expression	$x-y=-50$
	Used to Multiply two operands or values	$x*y=5000$
	Used to divide numerator by de-numerator	$x/y=2$
	Used to get the remainder after an integer division	$x\%y=0$

<http://www.tutorialslader.com/csharp/c-sharp-operators.htm>

# OPERADORES RELACIONAIS

Operator	Description	Example (x=100, y=50)
<code>==</code>	Used to check if two operands are equal or not. It returns true in case if both operands are equal else it returns false.	x==y returns false
<code>!=</code>	It works opposite of == operator. Returns true if two values are not same. It returns false if two values are same.	x!=y returns true
<code>&gt;</code>	Used to check if the left value is greater than the value of right side. It returns true if the left side value is greater.	x>y returns true
<code>&lt;</code>	Used to check if the left value is smaller than the value of right side. It returns true if the left side value is smaller.	x>y returns false
<code>&gt;=</code>	Used to check if the left value is either greater or equal to the value of right side. It returns true if any conditions match.	x>=y returns true
<code>&lt;=</code>	Used to check if the left value is smaller or equal to the value of right side. It returns true if any condition matches.	x<=y returns false

<http://www.tutorialslader.com/csharp/c-sharp-operators.htm>



# OPERADORES LÓGICOS

Operator	Description	Example (a=true, b=false)
&&	The AND operator returns true if both operands have TRUE value.	a&&b returns false
	The OR operator returns true if any operator from both has a TRUE value.	a  b returns true
!	NOT operator is used to reverse the operand's value. It converts TRUE into FALSE and vice versa.	!(a&&b) returns true

<http://www.tutorialslader.com/csharp/c-sharp-operators.htm>

ALGORITMOS	C, C#	NOTAS
E	& &&	<p><b>Exemplo:</b> ( A &gt; B ) &amp; ( C &lt; D ) Se ambos os operandos forem verdadeiros (1), ou seja, se A for maior que B e C menor que D, a expressão no seu todo é avaliada como verdadeira.</p> <p>Quando usado o operador lógico &amp;, ambos os operandos são sempre avaliados.</p> <p>Se for usado o &amp;&amp; e se o primeiro operando resultar em falso (0), o segundo já não é avaliado já que, qualquer que seja o seu resultado, a expressão completa será avaliada sempre como falsa (0).</p>
OU	 	<p><b>Exemplo:</b> ( A &gt; B )    ( C &lt; D ) Basta que um dos operandos seja verdadeiro (1) para que a expressão no seu todo seja avaliada como verdadeira.</p> <p>Quando usado o operador lógico  , ambos os operados são sempre avaliados.</p> <p>Quando usado o operador   , se o primeiro operando for verdadeiro (1), o segundo não será avaliado porque, qualquer que seja o seu valor, a expressão avaliará sempre como verdadeiro (1).</p>
NÃO	!	<p><b>Exemplo:</b> ~ ( A &gt; B ) Se A maior que B então o resultado é falso (0) já que A &gt; B é verdadeiro e a negação de verdadeiro é falso. Qualquer outro caso é verdadeiro (1).</p>

# OPERADORES DE ATRIBUIÇÃO

Operator	Description	Example If (a=100, b=50)
=	= Operator works from Right to left. It assigns values from right side operand to left side operand.	a = b Results a =50
+=	+= Operator is called add AND operator. First, it sums both operands(left and right) value and then it assigns the resultant value to the left operand.	a += b works same as a = a + b which Results a=150
-=	-= named as Subtract AND assignment operator. First, it subtracts the right operand's value from the left operand value and then it assigns the resultant value to the left operand.	a -= b works same as a = a - b which Results a = 50
*=	*= named as Multiply AND assignment operator. First, it multiplies both operands(left and right) value and then it assigns the resultant value to the left operand.	a *= b works same as a = a * b which Results a = 5000
/=	/= named as divide AND assignment operator. First, it divides left operand value from the right operand value and then it assigns the resultant value to the left operand.	a /= b works same as a = a / b which Results a = 2
%=	%= named as modules AND assignment operator. First, it divides left operand value from the right operand value and then it assigns modulus value to the left operand.	a %= b works same as a = a % b which Results a=0

Operator	Description	Example If (a=100, b=50)
<<=	<<= named as left shift AND assignment operator. First, it left shifts, left side operand value to that number of times specified on the right side, and then it assigns the resultant value to the left operand.	a <<= 2 works same as a = a << 2 which Results a = 400
>>=	>>= named as right shift AND assignment operator. First it right shifts, left side operand value to that number of times specified on the right side, and then it assigns the resultant value to the left operand.	a >>= 2 works same as a = a >> 2 which Results a = 25
&=	&= named as AND assignment operator. It performs a bitwise logical AND operation then it assigns the resultant value to the left operand.	a &= b works same as a = a & b which Results a = 32
^=	^= named as exclusive OR assignment operator. It performs a bitwise exclusive-OR operation and then it assigns the resultant value to the left operand.	a ^= b works same as a = a ^ b which Results a = 86
=	= named as OR assignment operator. It performs a bitwise OR operation and then it assigns the resultant value to the left operand.	a  = b works same as a = a   b which Results a = 118

<http://www.tutorialslader.com/csharp/c-sharp-operators.htm>



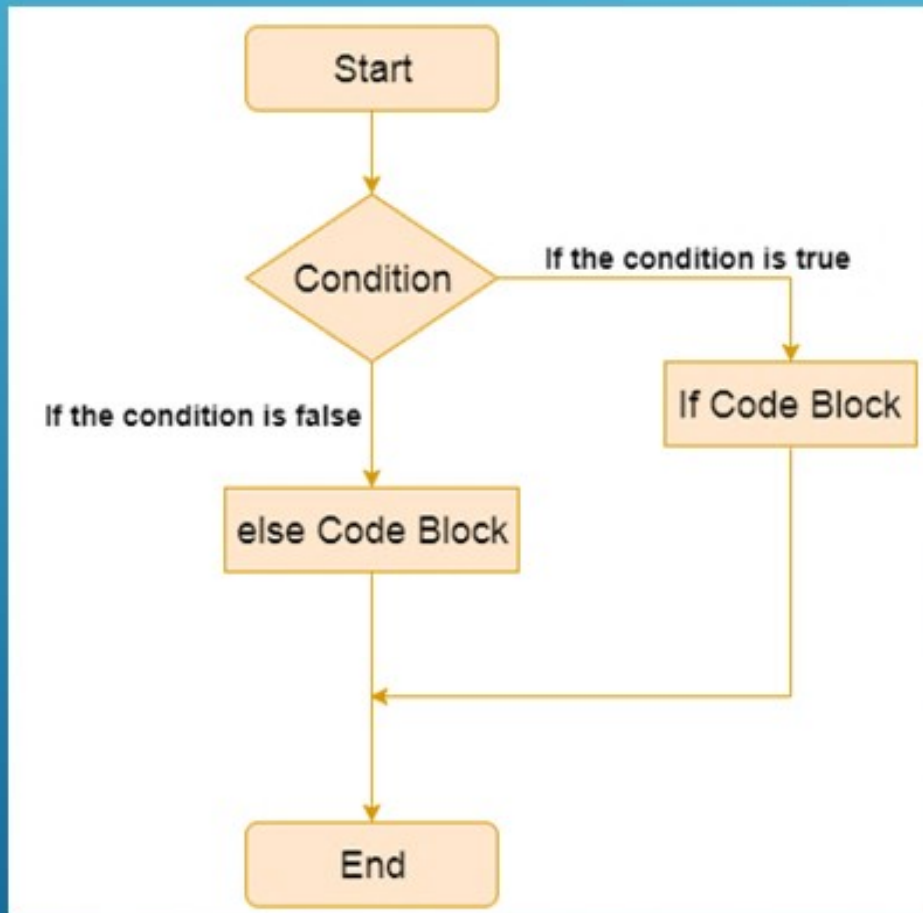
# INSTRUÇÕES DE DECISÃO

- As instruções de decisão (também conhecidas por instruções de seleção) permitem a escolha ou opção entre vários caminhos possíveis, sendo que a decisão acerca do caminho escolhido, depende totalmente do valor dos dados que serviram de base à decisão e do critério usado para a sua avaliação;
- Analise-se a seguinte frase: “se ganhar o 1º prémio do euromilhões, compro uma casa para mim e ofereço outra ao professor; se ganhar o 2º prémio então, em vez das casas compro dois carros, um para mim e outro para ele; senão pago-lhe um jantar...”.
- Fique claro a existência de 3 caminhos distintos na frase acima:
  - Caso ganhe o 1º prémio – se ganhar o 1º prémio...
  - Caso ganhe o 2º prémio – se ganhar o 2º prémio...
  - Nos restantes casos – senão (ou seja nos restantes casos... outros prémios) ...



# INSTRUÇÕES DE DECISÃO

- `if(...){...} else {...}`



```
static void Main()
{
    int a = 50, b = 20, c = 30;

    if (a > b)
    {
        if (a > c)
        {
            Console.WriteLine("a is greater than b and c");
        }
        else
        {
            Console.WriteLine("a is greater than b but less than c");
        }
    }
    else
    {
        if (a > c)
        {
            Console.WriteLine("a is less than b but greater than c");
        }
        else
        {
            Console.WriteLine("a is less than b and c");
        }
    }
}
```

# INSTRUÇÕES DE DECISÃO . EXEMPLO

Na Unidade Curricular de português foi definida a nota mínima de 8,5 valores para o primeiro teste e de 9 valores para o segundo. Foi também decidido que a nota final à unidade seria a média aritmética entre as notas dos dois testes.

A aprovação de um aluno é apenas possível se a nota final (média entre os dois testes) for igual ou superior a 10 valores. Se o aluno tiver uma nota final igual ou superior a 17 valores terá que fazer defesa oral. Nestes casos, a nota final do aluno será sempre inserida diretamente pelo professor. Caso não compareça à defesa, o professor atribuirá a nota final de 17 valores.

Desenvolva um algoritmo para resolver o problema e sugira uma implementação.

```
// definir variáveis
float nota1, nota2, notaFinal;
string nome;

// limpar consola
Console.Clear();

// recolher nome do aluno
Console.WriteLine("Insira o nome do aluno: ");
nome = Console.ReadLine();

// recolher nota primeiro teste
Console.WriteLine("");
Console.WriteLine("Insira a nota do primeiro teste: ");
nota1 = float.Parse(Console.ReadLine());

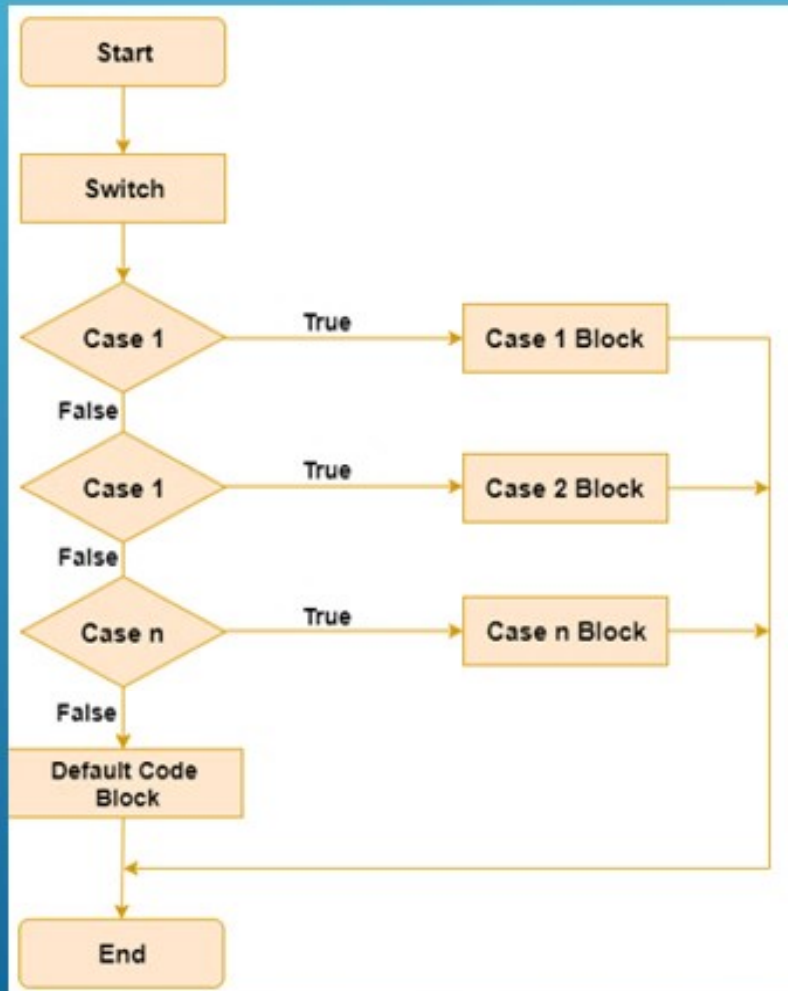
// recolher nota segundo teste
Console.WriteLine("");
Console.WriteLine("Insira a nota do segundo teste: ");
nota2 = float.Parse(Console.ReadLine());

// decidir aprovação do aluno
if (nota1 >= 8.5 && nota2 >= 9)
{
    // calcular notaFinal
    notaFinal = (nota1 + nota2) / 2;

    // verificar nota final
    if (notaFinal >= 10 && notaFinal < 17)
    {
        Console.WriteLine("");
        Console.WriteLine("O aluno {0} está aprovado com a nota: {1}.", nome, notaFinal);
    }
    else if (notaFinal >= 17)
    {
        Console.WriteLine("");
        Console.WriteLine("Introduza a nota final do aluno (defesa oral) {0}: ", nome);
        notaFinal = float.Parse(Console.ReadLine());
        Console.WriteLine("");
        Console.WriteLine("A classificação final do aluno {0}, é: {1}.", nome, notaFinal);
    }
    else
    {
        // reprovado
        Console.WriteLine("");
        Console.WriteLine("O aluno {0} está reprovado com a nota {1}!", nome, notaFinal);
    }
}
else
{
    // reprovado
    Console.WriteLine("");
    Console.WriteLine("O aluno {0} está reprovado!", nome);
}
```

# INSTRUÇÕES DE DECISÃO

- `switch(...){ case 1: {...} case 2: {...} case n: {...} default: {...} }`



```
static void Main()
{
    int a = 2;

    switch (a)
    {
        case 0:
            Console.WriteLine("The value of a is zero");
            break;
        case 1:
            Console.WriteLine("The value of a is one");
            break;
        case 2:
            Console.WriteLine("The value of a is two");
            break;
        case 3:
            Console.WriteLine("The value of a is three");
            break;
        case 4:
            Console.WriteLine("The value of a is four");
            break;
        case 5:
            Console.WriteLine("The value of a is five");
            break;
        default:
            Console.WriteLine("The value is not defined in the switch statement");
            break;
    }
}
```



# INSTRUÇÕES DE DECISÃO . EXEMPLO

Um clube de futebol pretende um programa que lhe facilite o processo de classificação de atletas em categorias. A tabela apresentada abaixo, define as categorias existentes e também os intervalos de idade que especificam a categoria para cada atleta.

Com o objetivo de promover as inscrições na categoria Juvenil, o clube oferece a taxa de inscrição aos atletas com 11 e 12 anos. Certifique-se que o utilizador é lembrado desse facto na sua implementação.

De	ATÉ	Categoria
11	15	Juvenil
16	20	Júnior
21	25	Profissional

```
// variáveis
string nome;
int idade;

// limpar consola
Console.Clear();

// solicitar e recolher dados do atleta
Console.Write("Insira o nome do atleta: ");
nome = Console.ReadLine();
Console.Write("Insira a idade do atleta: ");
idade = int.Parse(Console.ReadLine());

// decidir categoria e informar o utilizador
switch (idade)
{
    case 11:
        Console.WriteLine("LEMBRETE: inscrição gratuita (atletas de 11 e 12 anos!)");
        Console.WriteLine("O atleta {0} pertence à categoria {1}", nome, "juvenil");
        break;

    case 12:
        Console.WriteLine("LEMBRETE: inscrição gratuita (atletas de 11 e 12 anos!)");
        Console.WriteLine("O atleta {0} pertence à categoria {1}", nome, "juvenil");
        break;

    case 13:
    case 14:
    case 15:
        Console.WriteLine("O atleta {0} pertence à categoria {1}", nome, "juvenil");
        break;

    case 16:
    case 17:
    case 18:
    case 19:
    case 20:
        Console.WriteLine("O atleta {0} pertence à categoria {1}", nome, "junior");
        break;

    case 21: case 22: case 23: case 24: case 25:
        Console.WriteLine("O atleta {0} pertence à categoria {1}", nome, "profissional");
        break;

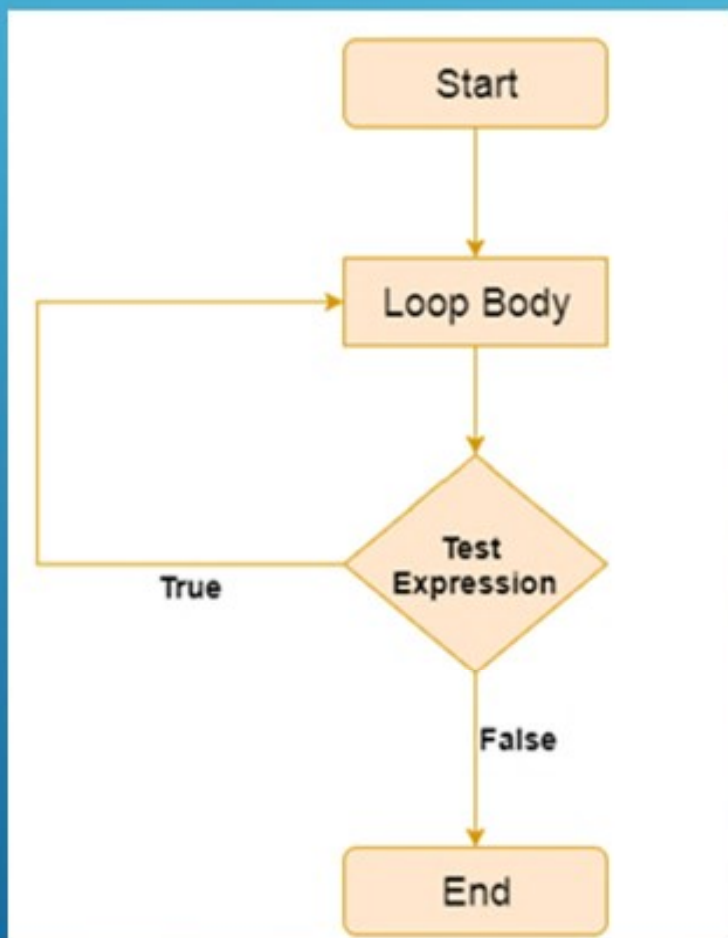
    default:
        Console.WriteLine("Categoria não prevista!!!");
        break;
}
```

# INSTRUÇÕES DE REPETIÇÃO

- As instruções de repetição, vulgarmente conhecidas por ciclos, permitem a execução repetitiva de uma sequência de instruções;
- Um dos aspetos mais importantes a ter em conta na definição de um ciclo é a condição de paragem (também chamada de sentinela);
- Caso seja mal planeada, pode conduzir o fluxo de execução a um ciclo infinito, dando a sensação ao utilizador que a aplicação deixou de responder;

# INSTRUÇÕES DE REPETIÇÃO

- `do {...} while (...);`



```
static void Main()
{
    int x = 0;
    do
    {
        Console.WriteLine("The value of x is = {0}", x);
        x++;
    } while (x < 10);
}
```

```
The value of x is = 0
The value of x is = 1
The value of x is = 2
The value of x is = 3
The value of x is = 4
The value of x is = 5
The value of x is = 6
The value of x is = 7
The value of x is = 8
The value of x is = 9
```



# INSTRUÇÕES DE REPETIÇÃO . EXEMPLO

Um centro comercial pretende adequar um pouco mais a decoração do seu espaço de lazer e de alimentação à faixa etária dos seus visitantes.

Para isso, no próximo fim-de-semana, irá recolher informação das pessoas que lá entram, com o objetivo de determinar:

- O número total de visitantes femininos e masculinos
- O número de visitantes com idade compreendida entre os 10 e 16 anos e os 16 e 24 anos.

```
// declarar variaveis
char genero;
int idade, contadorFemininos = 0, contadorMasculinis = 0,
    contadorIdade10Ate16 = 0, contadorIdade16Ate24 = 0;

do
{
    // limpar consola
    Console.Clear();

    // informar utilizador de como terminar aplicação
    Console.WriteLine("=== REGISTO DE NOVO VISITANTE =====(terminar com Z)");
    Console.WriteLine("");

    // recolher género
    Console.Write("        -> Género (M/F): ");
    genero = Console.ReadKey().KeyChar;

    // tratar somente se M ou F
    if (genero == 'M' || genero == 'm' || genero == 'F' || genero == 'f')
    {
        // recolher idade
        Console.WriteLine("");
        Console.Write("        -> Idade (0:99): ");
        idade = int.Parse(Console.ReadLine());

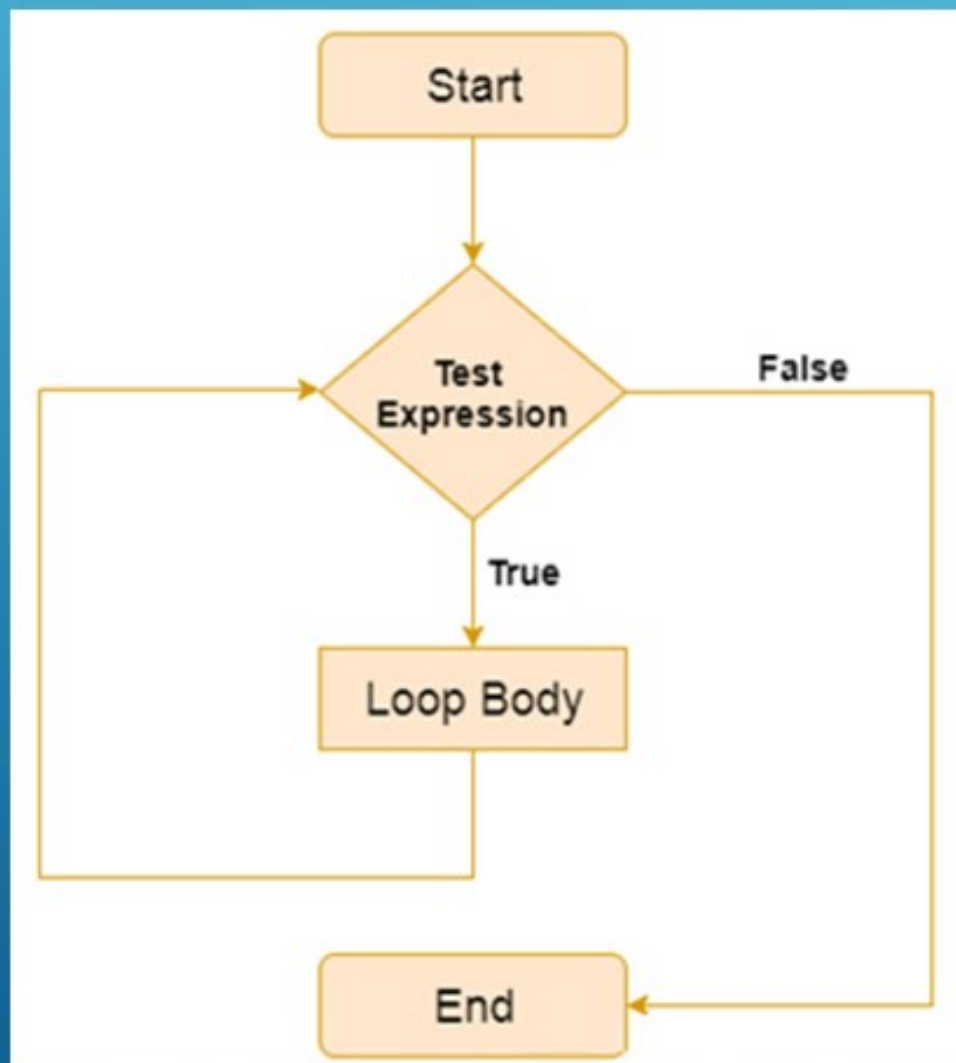
        // processar género
        if (genero == 'M' || genero == 'm')
            contadorMasculinis++;
        else
            contadorFemininos++;

        // processar idade
        if (idade >= 10 && idade < 16)
            contadorIdade10Ate16++;
        else if (idade >= 16 && idade < 24)
            contadorIdade16Ate24++;
    }
} while (genero != 'Z' && genero != 'z');

// mostrar resultados
Console.WriteLine("");
Console.WriteLine("");
Console.WriteLine("===== RESULTADOS ===== by Célio Carvalho =");
Console.WriteLine("Visitantes = femininos + masculinos = {0} + {1} = {2}",
    contadorFemininos.ToString(),
    contadorMasculinis.ToString(),
    (contadorFemininos + contadorMasculinis).ToString());
Console.WriteLine("Idade [10:16[ = {0} *** Idade [16:24[ = {1}",
    contadorIdade10Ate16.ToString(),
    contadorIdade16Ate24.ToString());
```

# INSTRUÇÕES DE REPETIÇÃO

- while (...) {...}



```
static void Main()
{
    int i = 0;
    while (i < 10)
    {
        Console.WriteLine("Value of i is : {0}", i);
        i++;
    }
}
```

```
Value of i is : 0
Value of i is : 1
Value of i is : 2
Value of i is : 3
Value of i is : 4
Value of i is : 5
Value of i is : 6
Value of i is : 7
Value of i is : 8
Value of i is : 9
```



# INSTRUÇÕES DE REPETIÇÃO . EXEMPLO

A Associação de estudantes de Escola Secundária "Só Crânios" está a organizar um concurso de matemática com o objetivo de encontrar os 3 melhores alunos à disciplina.

A equipa organizadora do concurso necessita de uma aplicação que faça as inscrições de alunos, através da recolha do nome e da nota de 11º ano a matemática.

O número de alunos inscritos no concurso nunca poderá ser inferior a 5 e superior a 10. No entanto, as inscrições terminam se a média das notas dos alunos inscritos atingir os 18 valores, desde que cumprido o preceito do mínimo de 5 alunos inscritos.

Caso um candidato tenha nota inferior a 14 valores, a sua inscrição é automaticamente recusada.

```
// declarar variáveis necessárias
string nome, lista = "";
double nota, soma = 0;
int numeroInscritos = 0;

// (ou) while (numeroInscritos < 5 || (numeroInscritos < 10 && (soma / numeroInscritos) < 18))
while (numeroInscritos != 10 && (numeroInscritos < 5 || (soma / numeroInscritos) < 18))
{
    // limpar a consola
    Console.Clear();

    /*
     * NOTA: ? é uma instrução if simplificada (também conhecido por iif)
     * (...a) ? (...b) : (...c), em que:
     *   -> (...a) é a condição a ser avaliada
     *   -> (...b) instrução executada se o resultado da avaliação é verdadeiro
     *   -> (...c) instrução executada se o resultado da avaliação é falso
     */

    // solicitar e recolher informação ao utilizador
    Console.WriteLine("= NOVO CANDIDATO ===== (inscritos: {0}), (média atual: {1})",
        numeroInscritos.ToString(),
        numeroInscritos > 0 ? (soma / numeroInscritos).ToString() : "0" // (instrução if simplificada - iif)
    );
    Console.WriteLine("");
    Console.Write(" > NOME: ");
    nome = Console.ReadLine();
    Console.Write(" > NOTA DO 11º ANO: ");
    nota = double.Parse(Console.ReadLine());

    // avaliar cumprimento das regras
    if (nota < 14)
    {
        // se nota inferior a 14 rejeita candidatura
        Console.WriteLine("");
        Console.WriteLine("\t\tINSCRIÇÃO RECUSADA (nota inferior a 14 valores)");
        Console.ReadKey();
        continue; // salta para próxima iteração
    }

    // incrementar o contador do numero de inscritos
    numeroInscritos++;

    // atualizar valor
    soma += nota;

    /*
     * NOTAS:
     *   -> (...).ToString(00), força representação do numero até às dezenas
     *   -> \t e \n, caracteres especiais que significam tabulação e nova linha (respetivamente)
     */

    // guardar informação acerca da inscrição
    lista += string.Format("{0} {1}\t\t{2} valores\n", numeroInscritos.ToString("00"), nome, nota);

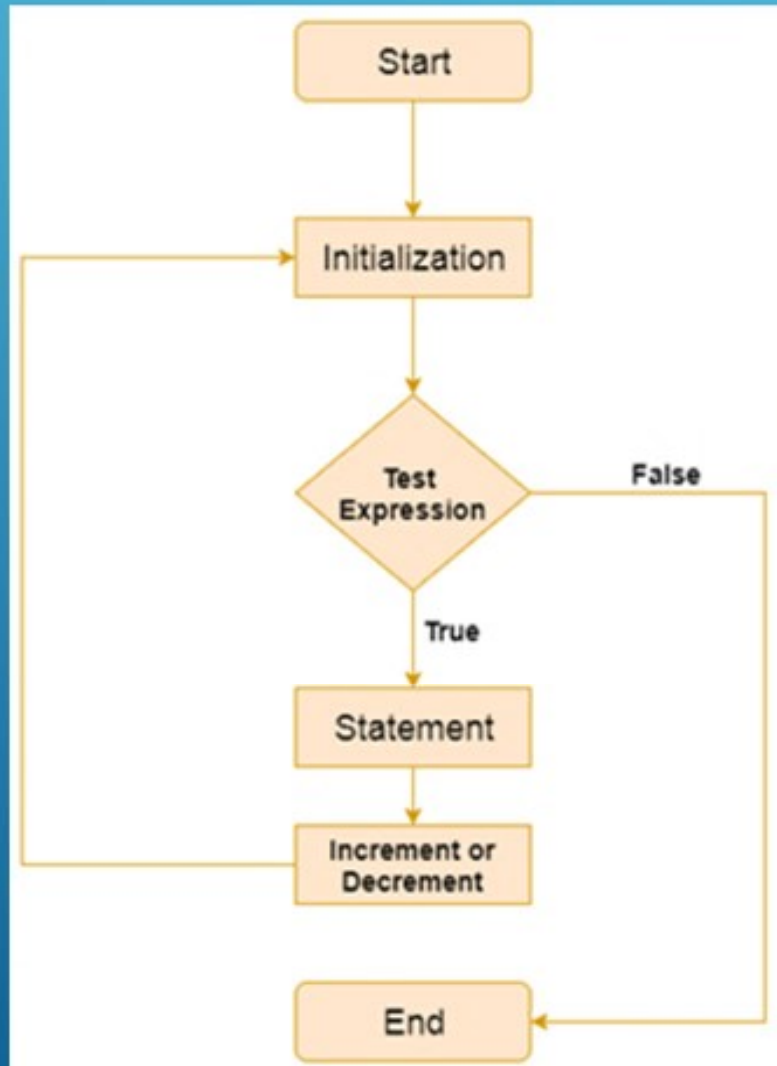
    // informar que a inscrição foi aceite
    Console.WriteLine("\t\tINSCRIÇÃO ACEITE");
    Console.ReadKey();
}

// mostrar resultados
Console.Clear();
Console.WriteLine("");
Console.WriteLine("");
Console.WriteLine("== RESULTADOS (inscrições) ===== by Célio Carvalho ==");
Console.WriteLine("TOTAL DE INSCRITOS: {0}", numeroInscritos.ToString());
Console.WriteLine("MÉDIA DOS INSCRITOS: {0}", (soma / numeroInscritos).ToString());
Console.WriteLine("..... RELATÓRIO DE INSCRITOS .....");
```



# INSTRUÇÕES DE REPETIÇÃO

- `for ( ; ; ) { ... }`



```
static void Main()  
{  
    for (int i = 0; i < 10; i++)  
    {  
        Console.WriteLine("The value of i is : {0}", i);  
    }  
}
```

```
The value of i is : 0  
The value of i is : 1  
The value of i is : 2  
The value of i is : 3  
The value of i is : 4  
The value of i is : 5  
The value of i is : 6  
The value of i is : 7  
The value of i is : 8  
The value of i is : 9
```

# INSTRUÇÕES DE REPETIÇÃO . EXEMPLO

Desenvolva um programa capaz de gerar e apresentar ao utilizador 10 números inteiros aleatórios entre 0 e 100.

```
// declarar variaveis
int numero;
Random random = new Random();

// preparar ecrã
Console.Clear();
Console.WriteLine("=== GERADOR DE NUMEROS ALEATÓRIOS === \nby Célio Carvalho");
Console.WriteLine();

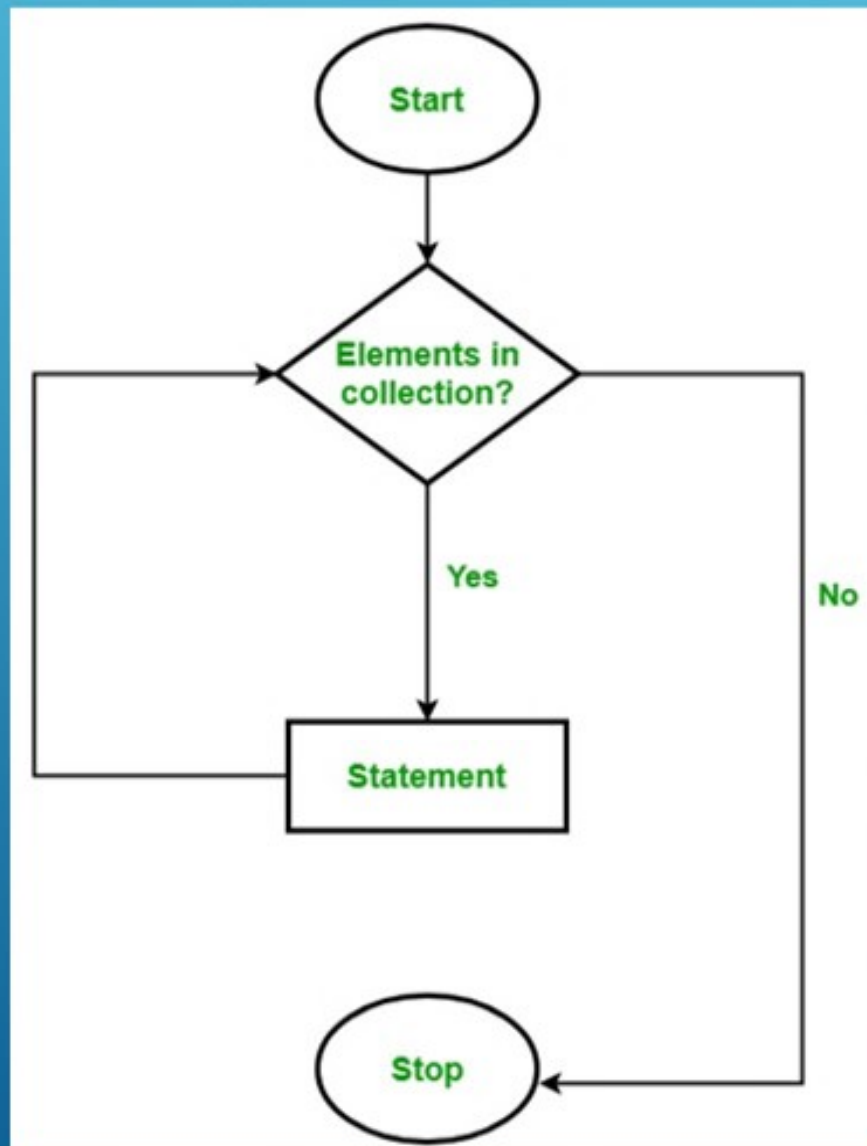
// gerar numeros aleatórios e mostrar ao utilizador
for (int i = 0; i < 10; i++)
{
    // gerar numero aleatorio
    numero = random.Next(0, 100);

    // mostrar numero gerado
    Console.WriteLine("{0}º gerado: {1}", (i+1).ToString("00"), numero.ToString());
}

// mostrar fim de geração
Console.WriteLine();
Console.WriteLine("== Fim");
```

# INSTRUÇÕES DE REPETIÇÃO

- `foreach (var element in elements) {...}`



```
// variables
int[] numbers = { 10, 30, 20, 5, 40 };

// print all numbers
foreach (var number in numbers)
{
    Console.WriteLine($"Number: { number }");
}
```

```
Number: 10
Number: 30
Number: 20
Number: 5
Number: 40
```

```
// variables
Student[] students = {
    new Student(1001, "manel", 15),
    new Student(1002, "maria", 16),
    new Student(1003, "fernando", 13)
};

// print all student grades
foreach (Student student in students)
{
    Console.WriteLine($"Nome: { student.Name }; Grade: { student.Grade}");
}
```

```
Nome: manel; Grade: 15
Nome: maria; Grade: 16
Nome: fernando; Grade: 13
```



# INSTRUÇÕES DE REPETIÇÃO . EXEMPLO

Crie uma aplicação que apresente na consola o **Code** e **Price** de todos os elementos da **List<Cobra>**.

Utilize a instrução de repetição **foreach**.

```
/// <summary>
/// This class is a template of a snake entity
/// </summary>
0 references
class Snake
{
    /// <summary>
    /// Code of this snake
    /// </summary>
    0 references
    public string Code { get; set; }

    /// <summary>
    /// Price to sell of this snake
    /// </summary>
    0 references
    public float Price { get; set; }
}
```

```
// variables
List<Snake> snakes = new List<Snake>();

// fill snakes collection
snakes.Add(new Snake { Code = "Snake1", Price = 15.60F });
snakes.Add(new Snake { Code = "Snake2", Price = 53.25F });
snakes.Add(new Snake { Code = "Snake3", Price = 8F });
snakes.Add(new Snake { Code = "Snake4", Price = 1062.50F });

// print all snakes
foreach (Snake snake in snakes)
{
    Console.WriteLine($"Code: { snake.Code }; " +
        $"Price: { snake.Price.ToString("###,##0.00").PadLeft(10) }");
}
```


```
Code: Snake1; Price:      15,60
Code: Snake2; Price:     53,25
Code: Snake3; Price:       8,00
Code: Snake4; Price:    1 062,50
```

# ENUMERADORES

- Os enumeradores são utilizados para definir um conjunto finito de opções ou estados;

```
2 references
enum Months { January, February, March, April, May, June,
              July, August, September, October,
              November, December};

0 references
static void Main(string[] args)
{
    // variables
    Months month = Months.
```



```
1 reference
enum WeekDays {Sunday, Monday, Tuesday, Wednesday,
               Thursday, Friday, Saturday};


0 references
void DoThings(WeekDays day, int age)...
```

```
6 references
enum Roles { Warrior, Wizard, Elf, Spy };

0 references
static void Main(string[] args)
{
    // variables
    Roles role1 = Roles.Warrior;
    Roles role2 = Roles.Elf;

    // print roles
    Console.WriteLine($"{ role2.ToString() }");

    Roles role3 = Roles.
```





# BIBLIOGRAFIA

- This.GetStarted(), IPCA, Luís Ferreira, João C. Silva, Patrícia Leite, Marta Martinho e Célio Carvalho.
- C# Essencial, IPCA, António Tavares, Luís Ferreira, Eva Oliveira, Manuela Cunha, João C. Silva, Marco Costa e Célio Carvalho.
- Data Structures And Algorithms Using C#, Pulaski Technical College, Michael McMillan.
- C# School, [www.programmersheaven.com](http://www.programmersheaven.com), Faraz Rasheed.
- Beginning C# 7 Programming with Visual Studio 2017, Wrox, Benjamim Perkins, Jacob Vibe Hammer e Jon D. Reid.
- <https://docs.microsoft.com/en-us/dotnet/csharp/>
- <https://www.tutorialsteacher.com/csharp>