PEF – 3528 – Ferramentas Computacionais na Mecânica das Estruturas: Criação e Concepção

Prof. Dr. Rodrigo Provasi

e-mail: provasi@usp.br

Sala 09 – LEM – Prédio de Engenharia Civil

Introdução ao C#

Parte I – Variáveis

Intruções (*Statements*)

- Um programa é composto de instruções (*statements*) que permitem realizar operações e cálculos. A estrutura básica de comandos é a seguinte:

```
Console.WriteLine("Hello World!");
```

- Toda instrução em C# termina com um ponto e vírgula. C# é *case sensitive* (difere entre maiúsculas e minúsculas) e *free format* (não se importa com espaços).

Palavras-chave

- O C# possui um conjunto de palavras-chave reservadas e que não podem ser usados como nome de variáveis. Existe um conjunto que também é usado pelo C# mas que não é reservado, porém seu uso é desencorajado.

Palavras-chave

abstract	do	in	protected	true
as	double	int	public	try
base	else	interface	readonly	typeof
bool	enum	internal	ref	uint
break	event	is	return	ulong
byte	explicit	lock	sbyte	unchecked
case	extern	long	sealed	unsafe

Palavras-chave

catch	false	namespace	short	ushort
char	finally	new	sizeof	using
checked	fixed	null	stackalloc	virtual
class	float	object	static	void
const	for	operator	string	volatile
continue	foreach	out	struct	while
decimal	goto	override	switch	
default	if	params	this	
delegate	implicit	private	throw	

Palavras-chave não reservadas

add	get	remove
alias	global	select
ascending	group	set
async	into	value
await	join	var
descending	let	where
dynamic	orderby	yield
from	partial	

Variáveis

- Para utilizar-se uma variável no C# é necessário, antes de mais nada, declará-la.
- Na declaração se define o tipo de variável e, como se verá, isso acaba restringindo o tipo de operações que podem ser feitas nas mesmas.

Variáveis

- Um exemplo de declaração de variável o seguinte:

```
int age;
```

- Nesse caso se declarou uma variável do tipo inteiro (*int*). Para atribuir um valor usa-se o seguinte comando:

```
age = 20;
```

Variáveis

Data type	Description	Size (bits)	Range	Sample usage
int	Whole numbers (integers)	32	-2^{31} through $2^{31} - 1$	int count; count = 42;
long	Whole numbers (bigger range)	64	-2^{63} through $2^{63} - 1$	long wait; wait = 42L;
float	Floating-point numbers	32	$\pm 1.5 \times 10^{-45}$ through $\pm 3.4 \times 10^{38}$	float away; away = 0.42F;
double	Double-precision (more accurate) floating-point numbers	64	$\pm 5.0 \times 10^{-324}$ through $\pm 1.7 \times 10^{308}$	double trouble; trouble = 0.42;
decimal	Monetary values	128	28 significant figures	decimal coin; coin = 0.42M;
string	Sequence of characters	16 bits per character	Not applicable	string vest; vest = "forty two";
char	Single character	16	0 through $2^{16} - 1$	char grill; grill = 'x';
bool	Boolean	8	True or false	bool teeth; teeth = false;

Variáveis

- O compilador gera um erro em tempo de compilação quando utiliza-se uma variável não inicializada:

```
int age;
Console.WriteLine(age); // compile-time error
```

Introdução ao C#

Parte I – Operadores

Operadores

- Operadores fazem ações sobre as variáveis. São '+', '-', '*' e '/'.
- Operadores são relacionados aos tipos. Se não for possível fazer a operação, tem-se um erro de compilação.
- Se for possível e acontecer algum erro (ex: divisão por zero), uma exceção é lançada (mais sobre exceções durante o curso).

```
// compile-time error
Console.WriteLine("Gillingham" - "Forest Green Rovers");
```

Operadores

- Precedência:
  - Multiplicação e divisão têm precedência sobre soma e subtração.

    Assim, 2 + 3 * 4 = 14

  - Se têm mesma precedência, segue a ordem em que estão dispostos:

    Assim, 4 * 2 / 6 = 12

Operadores

- Operador de atribuição (=):

```
int myInt;
myInt = 10; // value of assignment expression is 10


int myInt;
int myInt2;
myInt2 = myInt = 10;
```

Operadores

- Seja:

  int myInt, myInt2, myInt3 = 10;

  myInt3 = myInt / myInt2;

- Erros:

  Use of unassigned local variable 'myInt'

  Use of unassigned local variable 'myInt2'

Operadores compostos

- Pode-se compor operadores:

Don't write this	Write this
`variable = variable * number;`	`variable *= number;`
`variable = variable / number;`	`variable /= number;`
`variable = variable % number;`	`variable %= number;`
`variable = variable + number;`	`variable += number;`
`variable = variable - number;`	`variable -= number;`

Incrementos

- Os incrementos podem ser feitos através do operador ++:

```
count++; // postfix increment
++count; // prefix increment
count--; // postfix decrement
--count; // prefix decrement
```

Introdução ao C#

Parte I – Métodos

Métodos

- Um método (função) tem a seguinte sintaxe:

```
returnType methodName ( parameterList )
 {
      // method body statements go here
}
```

Métodos

- Exemplo (função que adiciona dois inteiros e retorna um inteiro):

```
int addValues(int leftHandSide, int rightHandSide)
 {
      return leftHandSide + rightHandSide;  //Retorna a soma dos dois inteiros
}
```

Métodos

- Para invocar um método segue-se a seguinte sintaxe:

result = methodName (argumentList)

- O *result* é opcional (o método pode retornar valor, mas ele não ser utilizado)

Métodos

- Exemplo: considere a função

  int addValues(int leftHandSide, int rightHandSide){ // … }

- Usos:

  addValues(39, 3); // okay

  int arg1 = 99;

  int arg2 = 1;

  addValues(arg1, arg2);

Métodos

- Erros:

  addValues; // compile-time error, no parentheses

  addValues(); // compile-time error, not enough arguments

  addValues(39); // compile-time error, not enough arguments

  addValues("39", "3"); // compile-time error, wrong types for arguments

Escopo

```
class Example
{
        void firstMethod()
        {
                int myVar; ...
        }
        void anotherMethod()
        {
                myVar = 42; // error - variable not in scope ...
        }
}
```

Escopo

```
class Example
{
        void firstMethod()
        {
                myField = 42; // ok ...
        }
        void anotherMethod()
        {
                myField++; // ok ...
        }
        int myField = 0;
}
```

Introdução ao C#

Parte I – Decisões

Blocos de decisão

- Para decidir o fluxo a seguir em um código, expressões booleanas devem ser avaliadas através de operadores relacionais (== e !=)

Operator	Meaning	Example	Outcome if age is 42
==	Equal to	age == 100	false
!=	Not equal to	age != 0	true

Operator	Meaning	Example	Outcome if age is 42
<	Less than	age < 21	false
<=	Less than or equal to	age <= 18	false
>	Greater than	age > 16	true
>=	Greater than or equal to	age >= 30	true

Operadores e precedência

Category	Operators	Description	Associativity
Primary	() ++ --	Precedence override Post-increment Post-decrement	Left
Unary	! + - ++ --	Logical NOT Returns the value of the operand unchanged Returns the value of the operand negated Pre-increment Pre-decrement	Left
Multiplicative	* / %	Multiply Divide Division remainder (modulus)	Left

Operadores e precedência

Category	Operators	Description	Associativity
Additive	+ -	Addition Subtraction	Left
Relational	<	Less than	Left
	<=	Less than or equal to	
	>	Greater than	
	>=	Greater than or equal to	
Equality	== !=	Equal to Not equal to	Left
Conditional AND	&&	Conditional AND	Left
Conditional OR	||	Conditional OR	Left
Assignment	=	Assigns the right-hand operand to the left and returns the value that was assigned	Right

Comando *if*

- A sintaxe de um *if* é a seguinte:

```
if ( booleanExpression )
    statement-1;
else
    statement-2;
```

Comando *if*

- Se o comando for na verdade um bloco de comandos, utiliza-se as chaves:

```
if (seconds == 59)
{
    seconds = 0;
    minutes++;
}
else
{
    seconds++;
}
```

Escopo (*if*)

```
if (...)
{
        int myVar = 0; // myVar can be used here ...
} // myVar disappears here
else
{
        // myVar cannot be used here ...
}
// myVar cannot be used here
```

Usando *if* em cascata

```
if (day == 0) { dayName = "Sunday"; }
else if (day == 1) { dayName = "Monday" ; }
else if (day == 2) { dayName = "Tuesday"; }
else if (day == 3) { dayName = "Wednesday"; }
else if (day == 4) { dayName = "Thursday"; }
else if (day == 5) { dayName = "Friday"; }
else if (day == 6) { dayName = "Saturday"; }
else { dayName = "unknown"; }
```

Comando *switch*

```
switch (day)
{
        case 0 :
                dayName = "Sunday";
                break;
        case 1 :
                dayName = "Monday";
                break;
        case 2 :
                dayName = "Tuesday";
                break;
        ...
        default :
                dayName = "Unknown";
                break;
}
```

Comando *switch*

- Regras:
  - Apenas alguns tipos de variáveis são permitidos no switch (*int*, *char* ou *string*, por exemplo)
  - *cases* precisam ser constantes
  - *cases* precisam ser únicos
  - caso especial: *'fall through'*

Comando *switch*

```
switch (trumps)
{
        case Hearts :
        case Diamonds :         // Fall-through allowed - no code between labels
                color = "Red";    // Code executed for Hearts and Diamonds
                break;
        case Clubs :
                color = "Black";
        case Spades : // Error - code between labels
                color = "Black";
                break;
}
```

Introdução ao C#

Parte I – Iteradores

Comando *while*

- Formato:

  while (booleanExpression)

  statement

- É necessário colocar a condição entre parênteses
- A expressão tem que ser um booleano
- Se na primeira vez que é checada, a expressão retornar *false*, o comando não é executado
- Se existir mais de um comando, utilizar chaves para delimitar o escopo

Comando *while*

- Exemplo:

```
int i = 0;
while (i < 10)
{
    Console.WriteLine(i);
    i++;
}
```

Comando *for*

- Sintaxe:

  for (initialization; Boolean expression; update control variable)

  statement

- Equivalente a:

  initialization

  while (Boolean expression)

  {

  statement

  update control variable

  }

Comando *for*

- O comando for equivalente ao *while* exibido anteriormente é:

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine(i);
}
```

Comando *for*

- Algumas partes *do for podem ser omitidas:*

```
for (int i = 0; ;i++)
{
    Console.WriteLine("somebody stop me!");
}
```

- Nesse caso, o *for* é executado eternamente!

Comando *for*

```
int i = 0;
for (; i < 10; )
{
        Console.WriteLine(i);
        i++;
}
```

- Nesse caso, a inicialização e incremento são feitos em outras regiões do código → Equivalente a um *while*!

Comando *for*

- Múltiplas inicializações:

```
for (int i = 0, j = 10; i <= j; i++, j--)
{ ...}
```

- Exemplo – leitura de arquivos:

```
for (string line = reader.ReadLine(); line != null; line = reader.ReadLine())
{
        source.Text += line + '\n';
}
```

Escopo do *for*

```
for (int i = 0; i < 10; i++)
{ ...}
Console.WriteLine(i); // compile-time error

for (int i = 0; i < 10; i++)
{ ...}
for (int i = 0; i < 20; i += 2) // okay
{ ...}
```

Comando *do* (*do ... while*)

- Sintaxe:

  do

  statement

  while (booleanExpression);

- Importante: a condição só é checada no final da primeira iteração. Mesmo que a condição seja *false* desde o início, o comando é executado pelo menos uma vez!

Comando *do* (*do ... while*)

- Exemplo:

```
int i = 0;
do
{
        Console.WriteLine(i);
        i++;
}while (i < 10);
```

Introdução ao C#

Parte I – Exceções

Lidando com erros

- Programas podem estar completamente corretos e, por causa de alguma entrada equivocada do usuário ou uma eventual situação não prevista, erros podem acontecer.

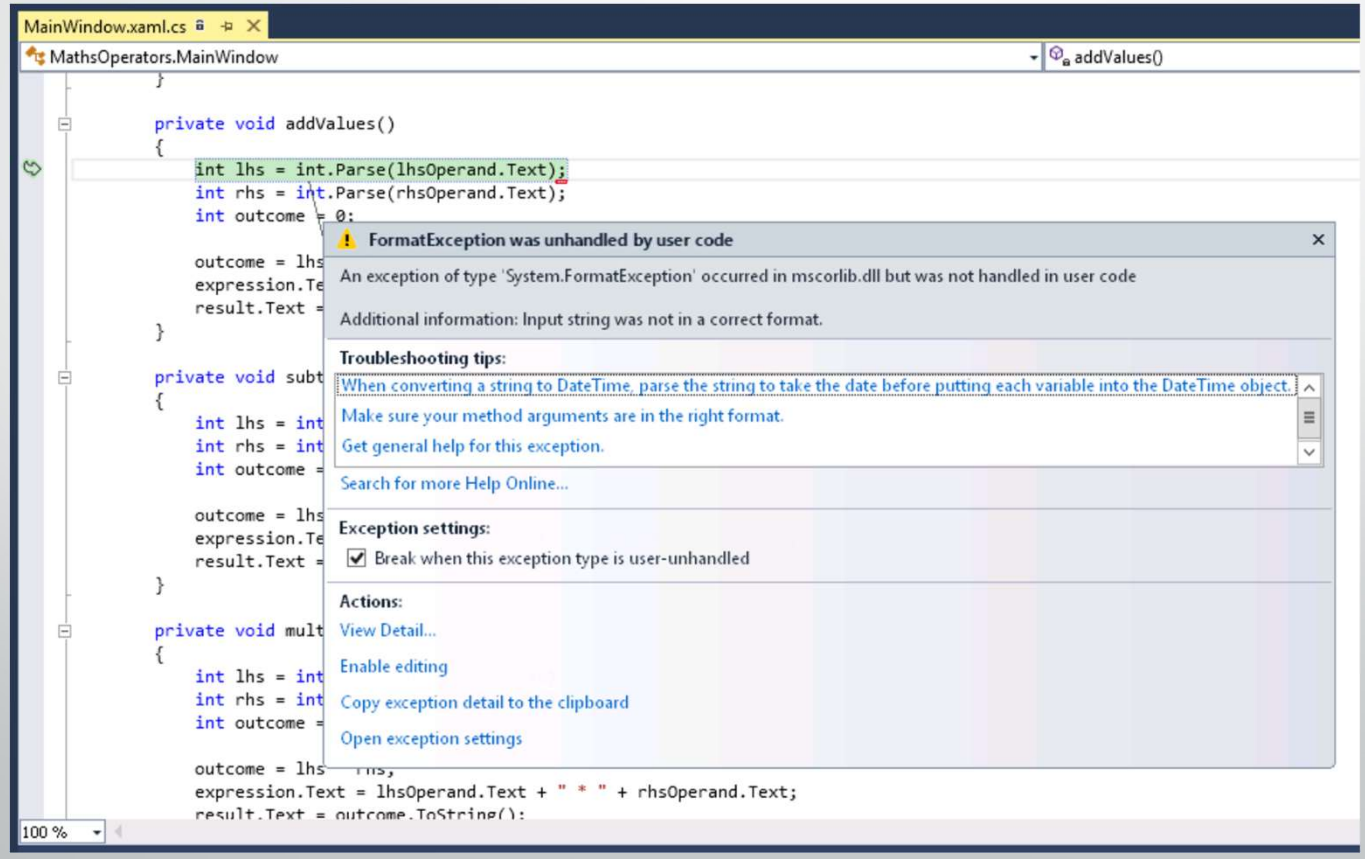
- Por exemplo, numa divisão de dois números, os seguintes erros podem ocorrer:
  - Variáveis com formato incompatível
  - Divisão por zero

Lidando com erros

- Para evitar possíveis erros, deve-se tratar exceções.
- Nesse caso, utiliza-se a seguinte sintaxe:

```
try
{
        int leftHandSide = int.Parse(lhsOperand.Text);
        int rightHandSide = int.Parse(rhsOperand.Text);
        int answer = doCalculation(leftHandSide,
                                   rightHandSide);
        result.Text = answer.ToString();
}
catch (FormatException fEx)
{
        // Handle the exception ...
}
```

Exceção no Visual Studio



Múltiplas exceções

```
try
{
        int leftHandSide = int.Parse(lhsOperand.Text);
        int rightHandSide = int.Parse(rhsOperand.Text);
        int answer = doCalculation(leftHandSide, rightHandSide);
        result.Text = answer.ToString();
}
catch (FormatException fEx)
{
        //...
}
catch (OverflowException oEx)
{
        //...
}
```

Múltiplas exceções (genérico)

```
try
{
        int leftHandSide = int.Parse(lhsOperand.Text);
        int rightHandSide = int.Parse(rhsOperand.Text);
        int answer = doCalculation(leftHandSide, rightHandSide);
        result.Text = answer.ToString();
}
catch (Exception ex) // this is a general catch handler
{
        //...
}
```

Lançando exceções

- A palavra chave *throw* permite lançar uma exceção do seu código. Exemplo:

```
public static string monthName(int month)
{
    switch (month)
    {
        case 1 :
            return "January";
        case 2 :
            return "February";
        ...
        case 12 :
            return "December";
        default :
            throw new ArgumentOutOfRangeException("Bad month");
    }
}
```

Usando o *finally*

- O fluxo de um bloco *try ... catch* funciona da seguinte forma:
  - O que está dentro do bloco try é executado até uma exceção ocorrer (ou ser lançada).
  - Nesse momento, o fluxo é interrompido e recomeça na primeira instrução do *catch*
  - Dessa forma, tudo mais que estava depois da instrução em que ocorreu a exceção não é executado.
  - Se alguma operação precisa ser executada, independente de ter ocorrido uma execeção ou não, utiliza-se o *finally.*

Usando o *finally*

- Considere o seguinte exemplo:

```
TextReader reader = ...;
...
string line = reader.ReadLine();
while (line != null)
{
        ...
        line = reader.ReadLine();
}
reader.Dispose();
```

- Nesse exemplo, lê-se um arquivo (abre o *reader*, lê-se o conteúdo e depois fecha-se o *reader*).

Usando o *finally*

- Nesse exemplo, pode ocorrer algumas exeções:
  - O arquivo a ser lido não existe
  - O caminho para o arquivo é inválido
  - Alguma coisa que foi lida (e utilizada) gerou uma exceção.
- Como solução para o problema apresenta-se o seguinte código:

Usando o *finally*

```
TextReader reader;
try
{
        reader = ...;
        string line = reader.ReadLine();
        while (line != null)
        {
                ...
                line = reader.ReadLine();
         }
}
```

```
finally
{
        if (reader != null)
        {
                reader.Dispose();
        }
}
```