Curso C# Conceitos Básicos

Diogo Cezar Teixeira Batista

xgordo@gmail.com

Universidade Tecnológica Federal do Paraná
Campus Cornélio Procópio
UTFPR-CP

Cornélio Procópio - 2008

Agenda I

Agenda II

4. Operadores
   - Operadores Aritméticos
   - Operadores de Incremento e Decremento
   - Operadores Lógico, Relacional e Condicional
   - Operação de Atribuição

Introdução

- O que é uma plataforma de desenvolvimento?
- Platadorma .NET;
- Características:
  - Independência de linguagem de programação (C#, J#, C++, VB);
  - Reutilização de código (DLL e Outra Bibliotecas);
  - Tempo de execução compartilhado (*runtime* compartilhado entre as linguagens);
  - Sistemas auto-explicativos e controle de versões;
  - Simplicidade na resolução de problemas complexos;
  - Multi-plataforma (independente do sistema operacional): CLR;

Introdução

- O que é uma plataforma de desenvolvimento?
- Platadorma .NET;
- Características:
  - Independência de linguagem de programação (C#, J#, C++, VB);
  - Reutilização de código (DLL e Outra Bibliotecas);
  - Tempo de execução compartilhado (*runtime* compartilhado entre as linguagens);
  - Sistemas auto-explicativos e controle de versões;
  - Simplicidade na resolução de problemas complexos;
  - Multi-plataforma (independente do sistema operacional): CLR;

Introdução

- O que é uma plataforma de desenvolvimento?
- Platadorma .NET;
- Características:
  - Independência de linguagem de programação (C#, J#, C++, VB);
  - Reutilização de código (DLL e Outra Bibliotecas);
  - Tempo de execução compartilhado (*runtime* compartilhado entre as linguagens);
  - Sistemas auto-explicativos e controle de versões;
  - Simplicidade na resolução de problemas complexos;
  - Multi-plataforma (independente do sistema operacional): CLR;

Introdução

- O que é uma plataforma de desenvolvimento?
- Platadorma .NET;
- Características:
  - Independência de linguagem de programação (C#, J#, C++, VB);
  - Reutilização de código (DLL e Outra Bibliotecas);
  - Tempo de execução compartilhado (*runtime* compartilhado entre as linguagens);
  - Sistemas auto-explicativos e controle de versões;
  - Simplicidade na resolução de problemas complexos;
  - Multi-plataforma (independente do sistema operacional): CLR;

Introdução

- O que é uma plataforma de desenvolvimento?
- Platadorma .NET;
- Características:
  - Independência de linguagem de programação (C#, J#, C++, VB);
  - Reutilização de código (DLL e Outra Bibliotecas);
  - Tempo de execução compartilhado (*runtime* compartilhado entre as linguagens);
  - Sistemas auto-explicativos e controle de versões;
  - Simplicidade na resolução de problemas complexos;
  - Multi-plataforma (independente do sistema operacional): CLR;

Introdução

- O que é uma plataforma de desenvolvimento?
- Platadorma .NET;
- Características:
  - Independência de linguagem de programação (C#, J#, C++, VB);
  - Reutilização de código (DLL e Outra Bibliotecas);
  - Tempo de execução compartilhado (*runtime* compartilhado entre as linguagens);
  - Sistemas auto-explicativos e controle de versões;
  - Simplicidade na resolução de problemas complexos;
  - Multi-plataforma (independente do sistema operacional): CLR;

Introdução

- O que é uma plataforma de desenvolvimento?
- Platadorma .NET;
- Características:
  - Independência de linguagem de programação (C#, J#, C++, VB);
  - Reutilização de código (DLL e Outra Bibliotecas);
  - Tempo de execução compartilhado (*runtime* compartilhado entre as linguagens);
  - Sistemas auto-explicativos e controle de versões;
  - Simplicidade na resolução de problemas complexos;
  - Multi-plataforma (independente do sistema operacional): CLR;

Introdução

- O que é uma plataforma de desenvolvimento?
- Platadorma .NET;
- Características:
  - Independência de linguagem de programação (C#, J#, C++, VB);
  - Reutilização de código (DLL e Outra Bibliotecas);
  - Tempo de execução compartilhado (*runtime* compartilhado entre as linguagens);
  - Sistemas auto-explicativos e controle de versões;
  - Simplicidade na resolução de problemas complexos;
  - Multi-plataforma (independente do sistema operacional): CLR;

Introdução

- O que é uma plataforma de desenvolvimento?
- Platadorma .NET;
- Características:
  - Independência de linguagem de programação (C#, J#, C++, VB);
  - Reutilização de código (DLL e Outra Bibliotecas);
  - Tempo de execução compartilhado (*runtime* compartilhado entre as linguagens);
  - Sistemas auto-explicativos e controle de versões;
  - Simplicidade na resolução de problemas complexos;
  - Multi-plataforma (independente do sistema operacional): CLR;

Arquitetura .NET

CLR (*Commom Language Runtime*)

É o ambiente de execução das aplicações .NET

CLS (*Common Language Specification*)

É um conjunto de regras que têm como objetivo gerar uma só resultante para a compilação de qualquer uma das linguagens suportadas pela plataforma .NET

BCL (*Base Classe Library*)

É uma biblioteca de componentes de software reutilizáveis

Arquitetura .NET

CLR (*Commom Language Runtime*)

É o ambiente de execução das aplicações .NET

CLS (*Common Language Specification*)

É um conjunto de regras que têm como objetivo gerar uma só resultante para a compilação de qualquer uma das linguagens suportadas pela plataforma .NET

BCL (*Base Classe Library*)

É uma biblioteca de componentes de software reutilizáveis

Arquitetura .NET

CLR (*Commom Language Runtime*)

É o ambiente de execução das aplicações .NET

CLS (*Common Language Specification*)

É um conjunto de regras que têm como objetivo gerar uma só resultante para a compilação de qualquer uma das linguagens suportadas pela plataforma .NET

BCL (*Base Classe Library*)

É uma biblioteca de componentes de software reutilizáveis

A linguagem C#

- É uma linguagem da plataforma .NET;
- Derivada do C++;
- Orientada a objetos.

A linguagem C#

- É uma linguagem da plataforma .NET;
- Derivada do C++;
- Orientada a objetos.

A linguagem C#

- É uma linguagem da plataforma .NET;
- Derivada do C++;
- Orientada a objetos.

Características do C#

- Simplicidade;
- Completamente orientada a objetos;
- Fortemente tipada;
- Tudo é um objeto;
- Controle de versões;
- Entre outros.

Características do C#

- Simplicidade;
- Completamente orientada a objetos;
- Fortemente tipada;
- Tudo é um objeto;
- Controle de versões;
- Entre outros.

Características do C#

- Simplicidade;
- Completamente orientada a objetos;
- Fortemente tipada;
- Tudo é um objeto;
- Controle de versões;
- Entre outros.

Características do C#

- Simplicidade;
- Completamente orientada a objetos;
- Fortemente tipada;
- Tudo é um objeto;
- Controle de versões;
- Entre outros.

Características do C#

- Simplicidade;
- Completamente orientada a objetos;
- Fortemente tipada;
- Tudo é um objeto;
- Controle de versões;
- Entre outros.

Características do C#

- Simplicidade;
- Completamente orientada a objetos;
- Fortemente tipada;
- Tudo é um objeto;
- Controle de versões;
- Entre outros.

Características do C#

Alguns detalhes que devem ser observados:

- Sensitive CASE, diferencia maiúsculas e minúsculas;
- As instruções terminam com ponto e virgula;
- As extensões dos arquivos são .cs;

Comentários

```
// Comenta apenas uma linha por vez

/* Este é um comentário de múltiplas linhas.
Ele pode ser dividido Em muitas linhas */
```

Características do C#

Alguns detalhes que devem ser observados:

- Sensitive CASE, diferencia maiúsculas e minúsculas;
- As instruções terminam com ponto e virgula;
- As extensões dos arquivos são .cs;

Comentários

```
// Comenta apenas uma linha por vez

/* Este é um comentário de múltiplas linhas.
Ele pode ser dividido Em muitas linhas */
```

Características do C#

Alguns detalhes que devem ser observados:

- Sensitive CASE, diferencia maiúsculas e minúsculas;
- As instruções terminam com ponto e virgula;
- As extensões dos arquivos são .cs;

Comentários

```
// Comenta apenas uma linha por vez

/* Este é um comentário de múltiplas linhas.
Ele pode ser dividido Em muitas linhas */
```

Características do C#

Alguns detalhes que devem ser observados:

- Sensitive CASE, diferencia maiúsculas e minúsculas;
- As instruções terminam com ponto e virgula;
- As extensões dos arquivos são .cs;

Comentários

```
// Comenta apenas uma linha por vez

/* Este é um comentário de múltiplas linhas.
Ele pode ser dividido Em muitas linhas */
```

Primeiro programa em C#

Escrevendo o tradicional programa Hello World, em C#:

Código 1: Hello World em C#

```
1  using System;
2  class Hello{
3          public static void Main(){
4                  Console.WriteLine("Hello World!!!");
5          }
6  }
```

Estrutura de um programa em C#

Código 2: Estrutura de um programa em C#

```
1  using System;
2  namespace MathNamespace{
3          public class MathClass{
4          /* Main: exibe no prompt */
5                  public static void Main(string[] args){
6                          Math m = new Math();
7                          Console.WriteLine(m.Sum(1,1));
8                  }
9                  //<summary>Classe Math</summary>
10                 public class Math:Object{
11                         public int Sum(int a, int b){
12                                 return (a+b);
13                         }
14                 }
15         }
16 }
```

Estrutura de um programa em C#

- *Namespaces*: são a forma lógica de organizar o código-fonte;
- Tipos: classes, estruturas, interfaces, delegações, enums;
- Membros: constantes, campos, métodos, propriedades, indexadores, eventos, operadores, construtores;
- Outros: coméntarios e instruções.

Estrutura de um programa em C#

- *Namespaces*: são a forma lógica de organizar o código-fonte;
- Tipos: classes, estruturas, interfaces, delegações, enums;
- Membros: constantes, campos, métodos, propriedades, indexadores, eventos, operadores, construtores;
- Outros: coméntarios e instruções.

Estrutura de um programa em C#

- *Namespaces*: são a forma lógica de organizar o código-fonte;
- Tipos: classes, estruturas, interfaces, delegações, enums;
- Membros: constantes, campos, métodos, propriedades, indexadores, eventos, operadores, construtores;
- Outros: coméntarios e instruções.

Estrutura de um programa em C#

- *Namespaces*: são a forma lógica de organizar o código-fonte;
- Tipos: classes, estruturas, interfaces, delegações, enums;
- Membros: constantes, campos, métodos, propriedades, indexadores, eventos, operadores, construtores;
- Outros: coméntarios e instruções.

Tipos de dados em C#

Tabela: Tipos primitivos do C#

Tipo C#	Tipo .NET	Descrição	Faixa de dados
bool	System.Boolean	Booleano	true ou false
byte	System.Byte	Inteiro de 8-bit com sinal	-127 a 128
char	System.Char	Caracter Unicode de 16-bit	U+0000 a U+ffff
decimal	System.Decimal	Inteiro de 96-bit com sinal com 28-29 dígitos significativos	$1{,}0 \times 10^{-28}$ a $7{,}9 \times 10^{28}$
double	System.Double	Flutuante IEEE 64-bit com	$+\text{-}5{,}0 \times 10^{-324}$ a $+\text{-}1{,}7 \times 10^{324}$
float	System.Single	Flutuante IEEE 32-bit com	$+\text{-}1{,}5 \times 10^{-45}$ a $+\text{-}3{,}4 \times 10^{38}$
int	System.Int32	Inteiro de 32-bit com sinal	-2.147.483.648 a 2.147.483.647
long	System.Int64	Inteiro de 64-bit com sinal	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
Object	System.Object	Classe base	-
Sbyte	System.Sbyte	Inteiro de 8-bit sem sinal	0 a 255
Short	System.Int16	Inteiro de 16-bit com sinal	-32,768 a 32,767
String	System.String	String de caracteres Unicode	-
Uint	System.UInt32	Inteiro de 32-bit sem sinal	0 a 4,294,967,295
Ulong	System.UInt64	Inteiro de 64-bit sem sinal	0 a 18,446,744,073,709,551,615
Ushort	System.UInt16	Inteiro de 16-bit sem sinal	0 a 65,535

Tipos Valor e Tipos Referência

- Tipos valor(*value types*): dados em memória;
- Tipos referência(*reference types*): referência para um valor;
- Tipos ponteiro(*pointer types*): apontam para um endereço de memória.

Tipos Valor e Tipos Referência

- Tipos valor(*value types*): dados em memória;
- Tipos referência(*reference types*): referência para um valor;
- Tipos ponteiro(*pointer types*): apontam para um endereço de memória.

Tipos Valor e Tipos Referência

- Tipos valor(*value types*): dados em memória;
- Tipos referência(*reference types*): referência para um valor;
- Tipos ponteiro(*pointer types*): apontam para um endereço de memória.

Conversão de tipos

- Valores são convertidos sempre para o tipo de maior faixa de valores;
- A conversão dos tipos de ponto flutuante(float, double) para decimal causa erro;
- A conversão entre os tipos com sinal e sem sinal de valores inteiros com o mesmo tamanho causa erro.

Conversão de tipos

- Valores são convertidos sempre para o tipo de maior faixa de valores;
- A conversão dos tipos de ponto flutuante(float, double) para decimal causa erro;
- A conversão entre os tipos com sinal e sem sinal de valores inteiros com o mesmo tamanho causa erro.

Conversão de tipos

- Valores são convertidos sempre para o tipo de maior faixa de valores;
- A conversão dos tipos de ponto flutuante(float, double) para decimal causa erro;
- A conversão entre os tipos com sinal e sem sinal de valores inteiros com o mesmo tamanho causa erro.

Tipos de conversão automática

Tabela: Tipos de conversão automática

Tipo	Converte em
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	long, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal
float	double

O Objeto *Convert*

- Em C# temos o objeto *Convert* que é usado para converter um tipo de dado em outro;
- Os tipos de dados suportados são: *Boolean*, *Char*, *SByte*, *Byte*, *Int16*, *Int32*, *Int64*, *UInt16*, *UInt32*, *UInt64*, *Single*, *Double*, *Decimal*, *DateTime* e *String*.

Código 3: Exemplo de utilização do objeto Convert

```
1  double dNumber = 23.15;
2
3  int    iNumber = System.Convert.ToInt32(dNumber);
4  bool   bNumber = System.Convert.ToBoolean(dNumber);
5  String strNumber = System.Convert.ToString(dNumber);
6  char   chrNumber = System.Convert.ToChar(strNumber[0]);
```

Arrays I

- *array* é uma matriz de valores do mesmo tipo, criada em tempo de execução, acessada por meio de um índice;
- declaração do *array* sempre faz o uso de um colchete([]);
- O tipo *array* pode ter diversas dimensões;

Código 4: Sintaxe para a declaração de Arrays

```
1  <TIPO>[ ] NomeDoArray = new <TIPO> [ tamanho do array ];
2
3  float[] ValorIndice = new float [10];
4  string[] ElementoVetor = new string[10];
```

Arrays II

Código 5: Sintaxe para a declaração de Arrays com duas ou mais dimensões

```
1  <TIPO> [,] NomeDoArray = new <TIPO> [tamanho do array,
       tamanho do array];
2
3  float [,] ValorIndice = new float [10,10];
4  string [,,] ElementoVetor = new string [10,10,10];
```

Arrays III

Código 6: Sintaxe para a declaração de uma matriz de Arrays com duas ou mais dimensões

```
1  <TIPO> [][] NomeDoArray = new <TIPO> [tamanho do array] [
       tamanho do array];
2
3  float [][] ValorIndice = new float [10][10];
4  string [][][] ElementoVetor = new string[10][10][10];
```

Arrays IV

Código 7: Sintaxe para a inicialização de Arrays com duas ou mais dimensões

```
1  <TIPO> [] NomeDoArray = new <TIPO> [tamanho do array]{
       valores separados por ,};
2
3  float [] ValorIndice = new float[5]{1.25,2,3.23,1.32,5};
4
5  string [,] ElementoVetor = new string[3,3] {{"ab", "ac", "bc
       "} ,
6  {"ab", "ac", "bc"}};
7
8  int [][] MatrizDeInteiro = new int [2][];
9  MatrizDeInteiro[ 0 ] = new int [ 5 ] {1,3,5,7,9};
10 MatrizDeInteiro[ 1 ] = new int [ 4 ] {2,4,6,8};
```

Arrays V

- Para passar um argumento array para um método, especifique o nome do array sem usar colchetes.
- Para que um método receba um *array*, a lista de parâmetros deve especificar que um array será recebido.

Código 8: Passando arrays à métodos

```
1  int[] vetor = {10, 20, 30, 40, 50};
2  Array.IndexOf(vetor, 22);
3
4  public void ExibeVetor( int[] vetor );
```

Arrays VI

Propriedades e Métodos dos Arrays

- `obj.Length` $\longrightarrow$ Tamanho do vetor;
- `Array.IndexOf(Array vetor, object value)` $\longrightarrow$ Procura a primeira ocorrência de valor em vetor;
- `Array.LastIndexOf(Array vetor, object value)` $\longrightarrow$ Procura a última ocorrência de valor em vetor;
- `Array.Sort(Array vetor)` $\longrightarrow$ Ordena um vetor crescentemente;
- `Array.Reverse(Array vetor)` $\longrightarrow$ Ordena um vetor decrescentemente.

Comandos de Seleção

- Escolha de uma possibilidade entre uma ou mais possíveis;
- Os comandos *if* e *switch* fazem parte deste grupo.

Comandos de Seleção

- Escolha de uma possibilidade entre uma ou mais possíveis;
- Os comandos *if* e *switch* fazem parte deste grupo.

Comando de Seleção If

- Expressão booleana para executar um bloco de comando;
- Cláusula else opcional.

Código 9: Exemplo do comando If em C#

```
1  if(a==true){
2  System.Console.Write("Verdadeiro");
3  }
4
5  if(a==true){
6    System.Console.Write("Verdadeiro");
7  }
8  else{
9    System.Console.Write("Falso");
10 }
```

Operador ternário

- *If* com a cláusula *else* única;
- Ternário: 3 expressões;
- Representado por interrogação;

Código 10: Operador Ternário

```
1  int x;
2  if(f==true)
3    x = 100;
4  else
5    x = 1000;
6
7  // As linhas acima podem ser substituídas por:
8
9  int x = f==true?100:1000;
```

Comando de Seleção Switch

- Switch: interruptor;
- Verifica se um valor existe em uma lista de opções;

Código 11: Comando Switch

```
1  switch(chOpt){
2      case '1':
3        Console.WriteLine("Inserir...");
4        break;
5      case '2':
6        Console.WriteLine("Atualizar...");
7        break;
8      case '3':
9        Console.WriteLine("Apagar...");
10       break;
11     default:
12       Console.WriteLine("Procurar...");
13 }
```

Break e Continue

- *Break*: Utilizada para saída prematuramente de um laço ou uma instrução *switch*;
- *Continue*: Pula as instruções restantes no corpo da estrutura e passa para a próxima iteração do laço;

Break e Continue

- *Break*: Utilizada para saída prematuramente de um laço ou uma instrução *switch*;
- *Continue*: Pula as instruções restantes no corpo da estrutura e passa para a próxima iteração do laço;

Comandos de Iteração ou Loop

- Conhecidos como laço ou loop;
- Executam repetidamente um comando ou bloco de comandos;
- Encerram o ciclo com uma condição;

Comando de Iteração For

- Possui 3 declarações opcionais;
- Separadas por ponto e vírgula (;);
- Declarações: inicialização, condição e iteração;
- Mais de uma expressão em cada parâmetro: separação por vírgula(,).

Código 12: Iteração For

```
1  for(int x=0; x < 100; ++x){
2          System.Console.WriteLine(x);
3  }
4  for(;;){ // Laço infinito
5          System.Console.WriteLine("Hello, World!");
6  }
7  for(int y=100, int x = 0;x < y; ++x, --y){ // Laço com mais
       de 1 variável
8          System.Console.WriteLine(y);
9  }
```

Comando de Iteração Foreach

- Enumera os elementos de uma coleção;

Código 13: Iteração foreach (exemplo)

```
1  int[] iSeq = new int[10];
2
3  foreach(int a in iSeq){
4          Console.Write(a);
5          Console.Write(" ");
6  }
```

Comandos de Iteração do e while

- Características semelhantes;
- Executam condicionalmente um comando ou bloco de comandos;
- Comando *do* $\longrightarrow$ executado uma ou mais vezes;
- Comando *while* $\longrightarrow$ executado nenhuma ou mais vezes;

Código 14: Iteração do while (exemplo)

```
1  int a = 0; bool f = true;
2  while(f){
3    if(++a==100) f = true;
4    System.Console.WriteLine(a);
5  }
6  int a = 0; bool f = true;
7  do{
8    if(++a==100) f = true;
9    System.Console.WriteLine(a);
10 } while(f);
```

Operadores em C#

- C# é uma linguagem muito rica em operadores;
- Representados por símbolos;
- Utilizados na construção de expressões;
- Sintaxe é baseada na sintaxe do C++;
- Múltiplas operações: precedência dos operadores (pode ser modificado por parênteses);

Operadores em C#

- C# é uma linguagem muito rica em operadores;
- Representados por símbolos;
- Utilizados na construção de expressões;
- Sintaxe é baseada na sintaxe do C++;
- Múltiplas operações: precedência dos operadores (pode ser modificado por parênteses);

Operadores em C#

- C# é uma linguagem muito rica em operadores;
- Representados por símbolos;
- Utilizados na construção de expressões;
- Sintaxe é baseada na sintaxe do C++;
- Múltiplas operações: precedência dos operadores (pode ser modificado por parênteses);

Operadores em C#

- C# é uma linguagem muito rica em operadores;
- Representados por símbolos;
- Utilizados na construção de expressões;
- Sintaxe é baseada na sintaxe do C++;
- Múltiplas operações: precedência dos operadores (pode ser modificado por parênteses);

Operadores em C#

- C# é uma linguagem muito rica em operadores;
- Representados por símbolos;
- Utilizados na construção de expressões;
- Sintaxe é baseada na sintaxe do C++;
- Múltiplas operações: precedência dos operadores (pode ser modificado por parênteses);

Operadores do C#

Tabela: Operadores do C#

Categoria	Operadores
Aritmética	& + - * / %
Lógica (boolena e bitwise)	& | ^ ! && || true false
Concatenação de string	+
Incremento e decremento	++ –
Shift	<< >>
Relacional	== != <> <= >=
Atribuição	= += -= *= /= %= &= —= ≙ <<= >>=
Acesso a membro	.
Indexação	[]
Indexação	()
Condicional	?:
Delegate (concatenação e remoção)	+ -
Delegate (concatenação e remoção)	new
Informação de tipo	is sizeof typeof
Controle de excessão de overflow	checked unchecked
Indireção e endereço	* -¿ [] &

Operadores Aritméticos

- Usados para execução de cálculos;
- Divididos em operadores unários e binários;

Operadores Aritméticos

- Usados para execução de cálculos;
- Divididos em operadores unários e binários;

Operadores Aritméticos Unários

Operadores Unários

(atribuídos a 1 atributo) + e - são utilizados para representar se o número é positivo ou negativo, respectivamente.

Código 15: Operadores Unários

```
1  x = +1000 // x = 1000
2  x = -1000 // x = -1000
```

Operadores Aritméticos Binários

Operadores Binários

+, -, *, / e % são utilizados nas expressões para execução de cálculos tais como soma, subtração, multiplicação, divisão e sobra.

Código 16: Operadores Binários

```
1  string x = "Hello" + "World"          // x = "HelloWorld"
2  string x = "Valor = " + 100           // x = "Valor = 100"
3  int    x = 1000 % 11                  // x = 10
```

Operadores de Incremento e Decremento

Incremento e Decremento

Os operadores $++$ e $--$ aumentam ou diminuem por um o valor correspondente.

Código 17: Operadores de Incremento e Decremento

```
1  int x = 1000; // x = 1000
2  x++;          // x = 1001
3  int y = x++;  // x = 1002 , y = 1001
4  x--;          // x = 1001
5  y = --x;      // x = 1000 , y = 1000
6  ++x;          // x = 1001
7  --x;          // x = 1000
8  y = ++x;      // x = 1001 , y = 1001
```

Operadores Lógico, Relacional e Condicional

- Utilizados em expressões onde o resultado retornado ou a característica é booleana.
- O operador de negação ! retorna o complemento (contrário) de um valor booleano.

Operadores Lógico, Relacional e Condicional

Os operadores relacionais ==, !=, <, >, <=, >=, resultam em um valor booleano e representam igual, não igual ou diferente, menor, maior, menor ou igual e maior ou igual, respectivamente.

Operadores Lógico, Relacional e Condicional

- Utilizados em expressões onde o resultado retornado ou a característica é booleana.
- O operador de negação ! retorna o complemento (contrário) de um valor booleano.

Operadores Lógico, Relacional e Condicional

Os operadores relacionais ==, !=, <, >, <=, >=, resultam em um valor booleano e representam igual, não igual ou diferente, menor, maior, menor ou igual e maior ou igual, respectivamente.

Operação de Atribuição

- Divididos entre simples e compostos;
- Utilizados na designação de um valor para uma variável;
- O operador = representa a atribuição simples;

Operadores de atribuição compostos

Os operadores +=, -=, *=, /=, %=, &=, —=, ≙, <<= e >>= representam a atribuição composta, que normalmente atuam como um atalho na construção de uma expressão.

Operação de Atribuição

- Divididos entre simples e compostos;
- Utilizados na designação de um valor para uma variável;
- O operador = representa a atribuição simples;

Operadores de atribuição compostos

Os operadores +=, -=, *=, /=, %=, &=, —=, ≙, <<= e >>= representam a atribuição composta, que normalmente atuam como um atalho na construção de uma expressão.

Operação de Atribuição

- Divididos entre simples e compostos;
- Utilizados na designação de um valor para uma variável;
- O operador = representa a atribuição simples;

Operadores de atribuição compostos

Os operadores +=, -=, *=, /=, %=, &=, —=, ≙, <<= e >>= representam a atribuição composta, que normalmente atuam como um atalho na construção de uma expressão.

Operação de Atribuição

Código 18: Exemplo do operador de atribuição

```
1  int x = 10;
2  int y = x;
3  int z = x + y;
```

Código 19: Exemplo do operador de atribuição composta

```
1  x = x + 10; //Pode ser escrito como:
2  x+= 10;     // x <op>= <valor>
```