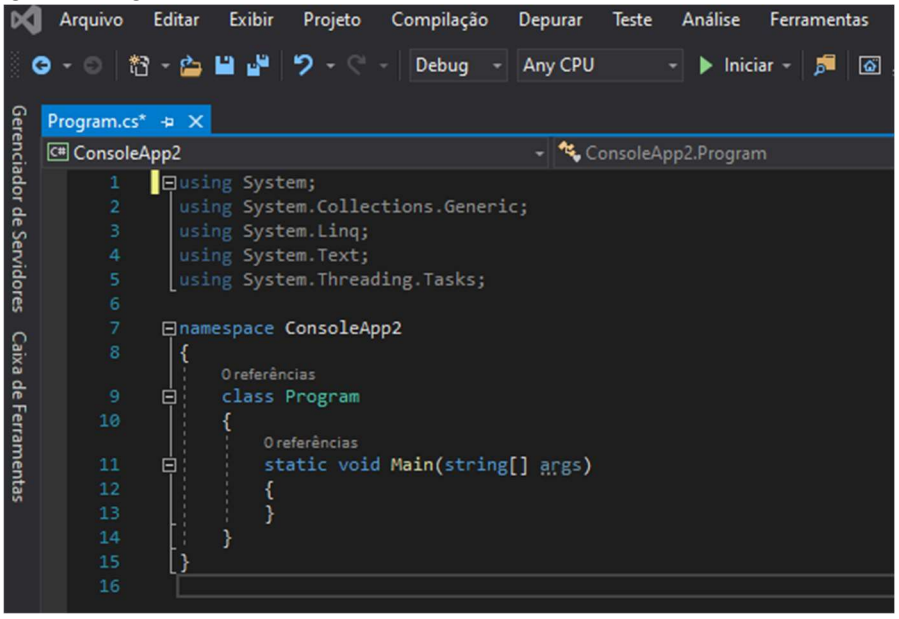
Olá, seja bem-vindo!

Nesta aula, você vai explorar alguns conceitos essenciais na construção de um programa na linguagem Visual C#, como tipos de dados e estruturas de controle e repetição. Talvez você saiba que todos os aplicativos usam dados, sejam eles fornecidos por usuário, através de alguma interface ou até mesmo de um banco de dados. Assim, para lidar da melhor forma com esses dados, você vai conhecer e entender o uso de variáveis, sabendo como elaborar e implementar expressões no Visual C#.

O primeiro passo é criar um projeto no Visual C# para, então, criar um programa a ser definido. Depois, carregue o Visual Studio 2019, selecionando a opção Criar um projeto. Não se esqueça também de escolher o tipo de projeto para o programa. Agora, armazene, no disco rígido, o projeto que está sendo criado e a versão do framework. Feito isso, será mostrada a interface do Visual Studio, apresentando, na parte superior da tela, a lista de menus e os botões de atalhos mais usados, como o botão Salvar. Além disso, no ambiente de edição da ferramenta, tem a principal classe, que será usada para elaborar o código, chamada Program.cs. Assim, considere um ConsoleApp2 preenchido com um exemplo de código na imagem a seguir.



Audiodescrição: A imagem mostra a interface do Visual Studio. Na parte superior da tela, temos a lista de menus: “Arquivo”, “Editar”, “Exibir, “Projeto”, “Compilação”, “Depurar”, “Teste”, “Análise” e “Ferramentas”. Além disso, temos os botões de atalhos mais usados, como Salvar. À esquerda, há uma aba intitulada “Programa.cs” e, abaixo, temos o ConsoleApp2 que está preenchido pelo código:
“using System
using System.Collection.Generic;
using System.Linq;
using System.Text ;
using System.Threading.Tasks;

namespace ConsoleApp1

0 referências
class Program

```
 0 referências
static void Main(string [] args)
{
}
}
```

}".

E aí, vamos aprender a programar?

Em primeiro lugar, saiba, antes de tudo, que uma variável é importantíssima em um programa de computador, pois ela é um objeto capaz de guardar um valor ou expressão, sendo identificadas por um nome. Dessa maneira, uma variável precisa ser obrigatoriamente criada, informando o tipo de dado, no qual o dado será armazenado devidamente, sendo essencial esse recurso na codificação de um programa, já que precisaremos armazenar os dados que iremos trabalhar no programa.

Para ilustrar melhor este conceito, imagine um armário de sua casa em que há diversas gavetas, as quais, em cada uma, você organiza objetos diferentes, sejam eles livros, fotografias, dentre outros. Pois bem, a variável é essa gaveta que tem um tipo de dado específico, assim como a gaveta com os objetos que você organizou. Agora, vamos contextualizar isso no Visual #C, que tal?

No seguinte trecho de código, são definidas algumas variáveis com seus respectivos tipos de dados. O código requer do usuário a digitação do valor para a variável solicitada.

```
classProgram
   {
staticvoid Main(string[] args)
{
int         var_inteiro;
long        var_inteiro_longo;
float       var_ponto_flutuante;
double      var_dupla_precisao;
decimal     var_monetario;
char        var_caracter;
bool        var_boleano;
string      var_cadeia_caracteres;

          var_inteiro = Console.Read();

          var_inteiro_longo = Console.Read();

          var_ponto_flutuante = Console.Read();

var_dupla_precisao = Console.Read();

          var_monetario = Console.Read();
```

```
        var_caracter = 'm';

        var_boleano = true;

        var_cadeia_caracteres = "Exercitando o uso de variaveis";



    }
}
```

No bloco de código, são definidas as variáveis var_inteiro, var_inteiro_longo, var_ponto_flutuante, var_dupla_precisao, var_monetario com entrada de dados de um aplicativo do tipo Console, por meio da digitação do usuário. No entanto, as variáveis var_caracter, var_boleano e var_cadeia_caracteres são definidas como valores constantes no próprio código, logo, o usuário não precisará informar valores para estas variáveis.

Bem, ainda falando sobre os tipos de dados, há um dado chamado string que você deve conhecer. Ele serve, caso seja preciso, para se trabalhar com texto no Visual C#, que é uma atividade bastante comum no desenvolvimento de códigos. Além disso, este tipo de dado possui uma característica inusitada: ele é definido como imutável, ou seja, você precisa criar uma nova string na memória e descartar uma anterior. Então, para a linguagem Visual C#, é necessário usar o tipo de dado para a definição da variável.

Ah! Agora, você deve estar ser perguntando, “será que os outros tipos de dados anteriores também possuem esta particularidade?” A resposta é não, quer saber por quê? As variáveis do tipo número, como int e float, são consideradas tipos de dados mutáveis, ou seja, elas podem ser alteradas, e o valor anterior não estará disponível na composição de uma sequência, como no tipo de dado string.

A partir desse momento, considere um exemplo de trecho de código, no qual foi criada uma variável endereço do tipo string para manipular um texto e, com isso, qualquer alteração feita nessa variável resulta na criação de uma nova cópia do valor modificado ao lado do valor antigo.

```
classProgram
  {
staticvoid Main(string[] args)
{

        string endereco = “1015”;
                endereço = “Rua – Pres. Getulio Vargas,” + endereço;
                endereco = endereço + “-Bairro: Brasilia”;

    }
  }
```

Logo, você pode fazer uma concatenação, realizando uma sequência: una o conteúdo de duas strings em uma única string. Na variável endereço, presente no trecho de código, inicialmente, é atribuído um texto “1015” que, posteriormente, concatenará com o novo valor, que é “Rua- Pres. Getulio Vargas”, sendo, portanto, o valor da variável endereço, após esta operação, assim representada: “Rua- Pres. Getulio Vargas, 1015”.
Até este ponto, você aprendeu sobre os principais tipos de variáveis que você pode usar na linguagem Visual C#. Porém, será que podemos elaborar estruturas mais complexas, usando variáveis? Vamos verificar!

Na linguagem Visual C#, chamamos este artifício de expressões, que são, praticamente, a parte central de uma aplicação, já que elas são indispensáveis para que você possa usá-las para avaliar e manipular dados. Logo, as expressões são coleções de operandos e operadores, que podem ser definidos assim:
Operandos são variáveis com valores associados, por exemplo, números e texto;
Operadores são as ações que podemos executar com os operandos, como as operações aritméticas da Matemática. Nesse caso, imagine uma expressão numérica assim definida: x + 2y, em que os operandos são, respectivamente, as variáveis x e y; e a operação de adição é o operador que informa qual ação deverá ser feita com os operandos.

Com esses conceitos compreendidos, vamos implementar um trecho de código que denota o uso de operadores e operandos?

```
classProgram
   {
staticvoid Main(string[] args)
{
        int   preco, desconto, total;
        preco = 100;
                desconto = 15;

                total = preco – desconto;

     }
   }
```

Considere um trecho de código, no qual são definidas três variáveis do tipo inteiro, que são: preco, desconto e total. Na sequência, serão atribuídos valores para os operando preco e desconto, respectivamente, 100 e 15. Portanto, é na variável total que será armazenado o resultado da expressão entre estes operandos com o operador usado, que está representado pela subtração. Entendeu?

Talvez você esteja se perguntando: “e caso eu precise fazer uma operação, e os tipos de dados não são compatíveis, como ficaria?” Bem, quando você está desenvolvendo um programa, geralmente, você precisa converter dados de um tipo para outro, não é mesmo? Por exemplo, considere duas variáveis do tipo inteiro e outra variável do tipo string. Assim, para entender melhor a implementação, considere um trecho de código que apresenta uma expressão com valortotal, contendo duas variáveis de tipos de dados distintos.

```
classProgram
   {
staticvoid Main(string[] args)
{
        int   valor1, valortotal;
        valor1 = 100;

                string valor2;
                valor2 = “50”;

                valortotal = valor1 + (int)valor2;  // O resultado deve ser 150

     }
   }
```

Você precisará converter o valor2 para valor inteiro, armazenando na variável valortotal, que é do tipo inteiro. Este processo de conversão de um valor de um tipo de dados para outro é chamado de conversão ou casting.

Quer uma dica? Utilize classes auxiliares para lhe ajudar no processo de conversão de tipos, pois, na linguagem Visual C#, deve-se usar a classe System.Convert que fornece métodos que podem converter um determinado tipo de dados em outro. Os métodos disponíveis que você utilizará, por exemplo, são: ToInt32, para converter para inteiro; ToString, para converter para string; e ToDouble, para converter para double.

Bom, já foi bastante assunto estudado até aqui, mas ainda há muito o que acompanhar. Acesse o próximo vídeo para conferir o conteúdo sobre estruturas de controle e repetição.

Vídeo 2 – Estruturas de controle e repetição

Agora, vamos tratar de um assunto que diz respeito a estruturas de controle e repetição, OK?

É importante que você tenha em mente que a lógica, em um desenvolvimento de um aplicativo, pode apresentar diferentes seções no código, dependendo do estado dos dados neste programa. Para isso, imagine que você esteja trabalhando em um arquivo de Word, e que precisa fechá-lo. O sistema perguntará se você deseja salvar as modificações, não é? Ao elaborar o código, o programa vai executar algum trecho de código para salvar o arquivo, certo? Porém, se esta condição não foi pensada, simplesmente irá fechar o arquivo e as alterações feitas não serão persistidas. Neste exemplo, observe que foi necessário fazer alguma escolha, e isso requer que o pensamento lógico, na resolução do problema na codificação, possa contemplar alguma estrutura de controle. Sendo assim, que tal aprender as estruturas de controle e repetição que a linguagem Visual C# apresenta?

Na linguagem Visual C#, a principal estrutura condicional é a instrução IF. Há também outra estrutura que você pode usar para tomar decisões mais complexas com base em escolha. Neste caso, seria a instrução SWITCH, que iremos tratar na sequência, tudo bem?

Você deve usar a instrução IF, caso a condição seja verdadeira; se a instrução for verdadeira, o bloco de código associado à instrução IF será executado. Caso contrário, a instrução é finalizada, e o controle do programa passa para a próxima instrução. Agora, considere um trecho de código com a instrução IF implementada.

```
classProgram
   {
staticvoid Main(string[] args)
{
        String clima =“Bom”;

                If (clima == “Bom”)
                        {
                                Console.Writeln(“Vamos à praia!!!”);
                        }
                Console.Write(“O dia pode estar nublado”);

      }
   }
```

Perceba que o bloco de código apresenta uma variável clima do tipo string e uma estrutura condicional, usando IF. A variável clima está associada à condição “bom”. De acordo com o código, se a variável clima avaliar que o clima está bom, a execução do código apresentará, no console, esta mensagem: “vamos à praia”. Caso contrário, o resultado será a mensagem “o dia pode estar nublado”.

As instruções IF podem estar associadas à cláusula ELSE. Dessa forma, se a instrução for falsa, o fluxo de execução é desviado para a cláusula ELSE. Para entender isso melhor, considere um trecho de código que apresenta, praticamente, a mesma estrutura do trecho anterior, mas com alguns ajustes, ou seja, a cláusula ELSE é acrescentada:

```
classProgram
   {
staticvoid Main(string[] args)
{
        String clima =“Bom”;

                If (clima == “Bom”)
                        {
                                Console.Writeln(“Vamos à praia!!!”);
                        }
                else
                        {
```

```
                    Console.Write("O dia pode estar nublado");
                    }

    }
}
```

As instruções IF podem ainda estar associadas com a clausula ELSE IF. A ideia desta clausula é possibilitar um encadeamento condicional, em que as cláusulas são testadas na ordem em que aparecem no código após a instrução IF. Se alguma delas retornar true (verdadeiro), o bloco de código associado a essa instrução será executado, e o controle deixará o bloco de código associado à estrutura inteira do IF. Agora, considere um trecho de código que adiciona a cláusula ELSE IF. Assim, o código apresenta, em sua estrutura, uma condição dentro de outra condição, formando um alinhamento condicional.

```
classProgram
{
staticvoid Main(string[] args)
{
        String clima ="Bom";

                If (clima == "Bom")
                        {
                                Console.Writeln("Vamos a praia!!!");
                        }
                else if (clima == "Nublado")
                        {
                        Console.Write("Vamos assistir um filme no cinema");
                        }
                else
                        {
                        Console.Write("O dia pode estar chovendo");
                        }



    }
}
```

Consequentemente, o código se tornou mais complexo com o uso de IF/ELSE. No entanto, há uma forma de deixá-lo mais simples, caso você queira.

Isso pode ser feito a partir da instrução SWITCH. Ela pode ser usada para substituir a IF/ELSE. Assim, é necessário alterar a estrutura do código, substituindo o uso do IF/ELSE pela a estrutura da instrução SWITCH, criando as condições de acordo com o valor de entrada. No caso do exemplo de código em questão, as condições são: bom e nublado, e, caso nenhum seja informado, será considerada a opção padrão definida no bloco de código:

```
classProgram
   {
staticvoid Main(string[] args)
{
                string clima = Console.Readln();

                Switch (clima)

                {

                Case “Bom”:
                        Console.Writeln(“Vamos à praia!!!”);
                        break;

                Case “Nublado”:
                        Console.Write(“Vamos assistir a um filme no cinema”);
                        Break;

                default:
                        Console.Write(“O dia pode estar nublado”);
                        Break;

        }
        }
   }
```

No código, há duas declarações de CASE e, em cada declaração, há uma palavra-chave, que é break. Isso faz que o controle da execução do programa vá para o final da instrução SWITCH, após processar o bloco de código.

Muito legal estas estruturas condicionais, não é? Agora, vamos aprender como implementar uma iteração lógica composta por vários comandos. Esse tipo de iteração é executado diversas vezes enquanto uma certa condição de teste é considerada verdadeira. Caso ela se torne falsa, a iteração é finalizada.

Pode-se dizer que uma iteração fornece uma maneira conveniente de executar um bloco de código diversas vezes, como que interagindo sobre uma coleção de itens em um vetor de string, ou ainda, executando várias vezes a mesma função. No Visual C#, temos várias estruturas que podem implementar esta lógica de iteração por meio de uma repetição iterativa, que ocorre quando um código é executado várias vezes. Essa repetição iterativa também é chamada de loop.

Agora, vamos conhecer a estrutura de repetição FOR, que executa um bloco de código repetidamente até que a expressão especificada seja avaliada como falsa. Nela, você pode definir uma repetição com FOR, como o valor inicial, valor final, e até de que maneira será feita a iteração ao chegar no valor final. Considere um trecho de código com o uso da estrutura FOR:

```
classProgram
   {
staticvoid Main(string[] args)
{

                Int a;
                        For (int a = 0; a < 5; a++)
                                {
                                 Console.WriteLine(“{0}”,a);
                                }
        }
   }
```

Ele executa um loop interativo que é executado 5 vezes. Ao usar o FOR, primeiro, deve-se inicializar um valor, como um contador. Portanto, a=0. Em cada loop, verifique se o valor do contador, que neste exemplo irá incrementar em 1 (a++), está dentro do intervalo para executar mais uma iteração, e, assim, executar o que estiver definido dentro do corpo da instrução; em seguida, o trecho do código irá mostrar o valor na tela do computador. Para isso, use esta estrutura de repetição quando tiver a definição exata da quantidade de repetições que serão executadas.

Embora o tipo de repetição FOR seja mais fácil, definir uma instrução corretamente pode ser difícil. Vamos a um exemplo, OK?! Para um desenvolvedor trabalhar com um vetor, é necessário que ele saiba quantos elementos esse vetor contém previamente. Em muitos casos, pode parecer fácil, porém gera mais trabalho no desenvolvimento. Assim, caso você queira trabalhar com os elementos de um vetor que não exija a quantidade de vezes, inicialmente, a ser executada, use a estrutura foreach para repetição.
Agora, considere um trecho de código no qual é descrito como a estrutura FOREACH funciona. Nele, inicialmente, é definida uma variável do tipo string, e os valores que irão compor o vetor de string são informados. Em seguida, é usado o comando Foreach. As iterações devem ser feitas até o último elemento do vetor. Nesse exemplo do trecho de código, esse elemento é Ana.

```
classProgram
   {
staticvoid Main(string[] args)
{

                String[] professor = {“Leonardo”, “Marcondes”, “Ana”};

                        Foreach (string professor in nomes
                                {
                                 Console.WriteLine(professor);
                                }
        }
```

```
}
```

Dessa maneira, define-se um vetor do tipo string, no qual espera-se que todos os nomes, que estão presentes no vetor, sejam exibidos na tela. No caso, não há verificação e condição de parada da estrutura de repetição.

Estude, agora, a estrutura de repetição While, que permite uma iteração em loop enquanto for satisfeita a condição inicial; sendo executado, em cada iteração, o bloco de código presente dentro da estrutura. Por exemplo, você usaria o loop do tipo while para processar a entrada do usuário até que ele indique que não há mais dados para inserir. Para entender melhor, considere um trecho de código que contenha a estrutura de repetição WHILE no lugar da estrutura de repetição FOR.

```
classProgram
  {
staticvoid Main(string[] args)
{

            Int a=0;
                  While (a < 5)

                        {
                          If (a == 9)
                                Break;
                         Console.WriteLine(a);

                          a++;
                        }
      }
  }
```

No trecho, é definida a variável “a”do tipo inteiro com o valor inicial zero, depois é criada uma condição inicial de teste (a < 5) para que possa ser verificada; para, então, ser executado o bloco de código dentro da estrutura de repetição. Considere que no trecho de código há um comando break. Com ele, pode-se incorporar ao seu código definindo uma condição de interrupção na execução, passando um parâmetro (a == 9).

Por fim, será abordada a última instrução de repetição interativa chamada DO, que é bastante semelhante a estrutura WHILE, excetuando-se que, no DO, sempre será executado pelo menos uma vez. Caso a condição não seja atendida inicialmente, a estrutura de repetição WHILE nunca será executado. Para fixar melhor este último conteúdo, vamos tomar como exemplo o trecho de código feito anteriormente e reescrevê-lo usando a estrutura de repetição DO.

```
classProgram
  {
```

```
staticvoid Main(string[] args)
{

                Int a=0;

                        DO
                                {
                                  If (a == 9)
                                        Break;
                                Console.WriteLine(a);

                                  a++;

                                } While (a < 5);
        }
  }
```

Perceba que a validação, na estrutura de repetição DO, passa a ser no final do bloco de código ao invés do que é empregado na estrutura WHILE.

Pois bem, chegamos ao final do conteúdo. Nesta aula, você aprendeu os principais tipos de dados, como trabalhar com expressões e estruturas de controle e repetição na linguagem Visual C#. Estes tópicos são fundamentais em seu estudo na linguagem e correspondem às noções básicas que você precisa ter para fazer seu primeiro programa usando C#.

Mas não é só isso! Ainda tem muito o que aprender sobre essa linguagem, por isso, continue estudando. Até mais!