

## Mãos na massa

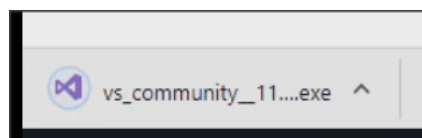
Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

---

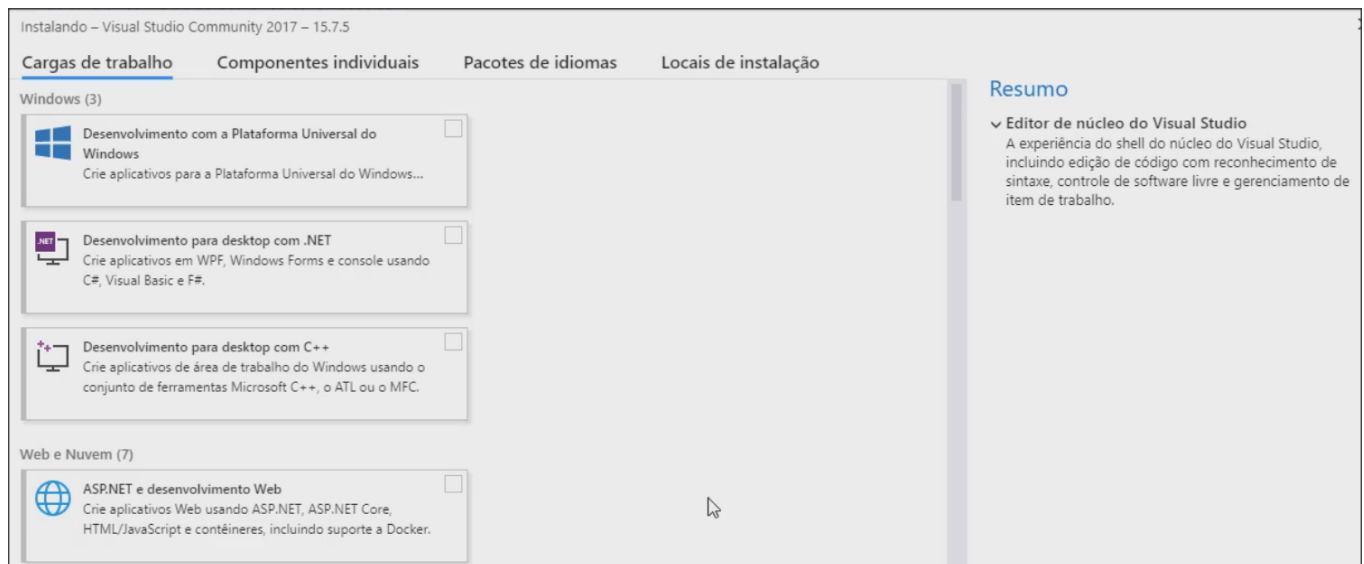
- 1) A primeira coisa que você precisa fazer, para trabalhar com o **Visual Basic .NET**, é baixar e instalar o Visual Studio.NET. Se você já o tem instalado na sua máquina, apenas certifique-se se você pode criar programas em **Visual Basic .NET**. Se tudo estiver certo, você pode pular para o **passo 11**, já que a seguir será mostrado como fazer a instalação do Visual Studio em uma máquina limpa, que não tem o programa instalado previamente.
- 2) Acesse a [página de Download \(https://visualstudio.microsoft.com/pt-br/downloads/\)](https://visualstudio.microsoft.com/pt-br/downloads/) do Visual Studio. Você verá quatro versões para download. Use versão **Visual Studio Community 2017**, que é uma versão gratuita e pode ser usada para estudos e desenvolvimento:



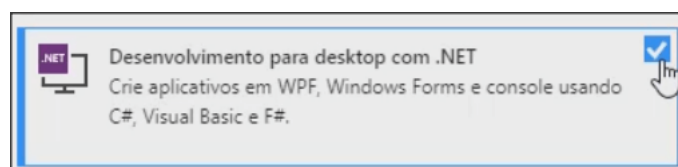
- 3) Será baixado um pequeno programa, que deve ser executado para iniciar a instalação do Visual Studio:



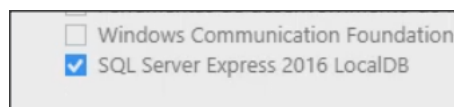
- 4) Continue a instalação, selecionando todos os botões padrões, até verificar a seguinte tela:



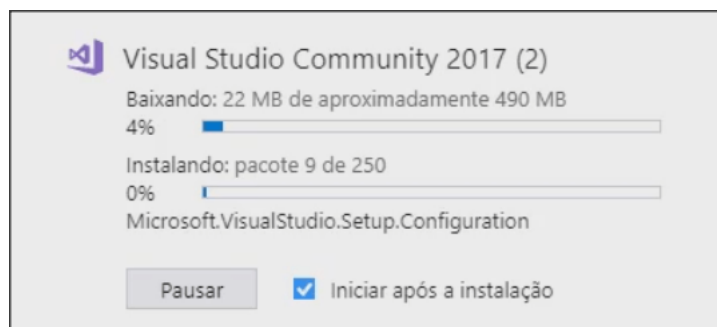
5) Selecione a opção **Desenvolvimento para desktop com .NET**:



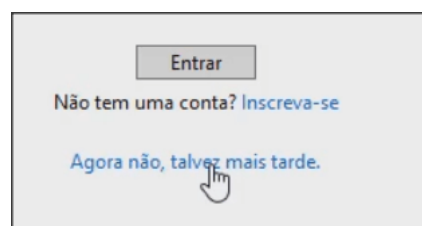
6) Selecione também, do lado direito, a opção **SQL Server Express 2016 LocalDB**:



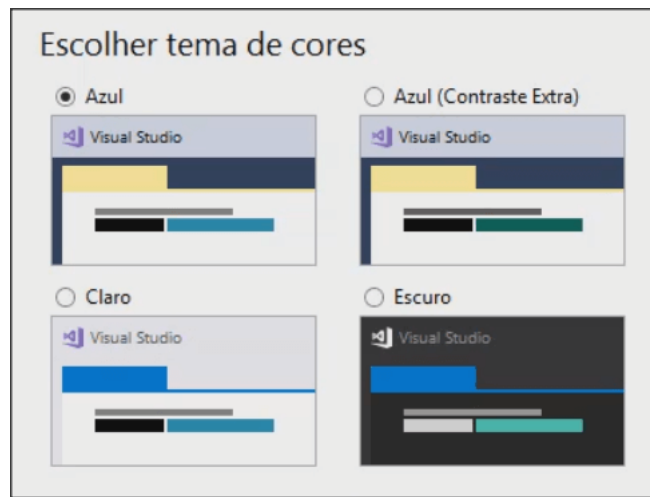
7) Clique no botão **Instalar** e aguarde a instalação ser finalizada:



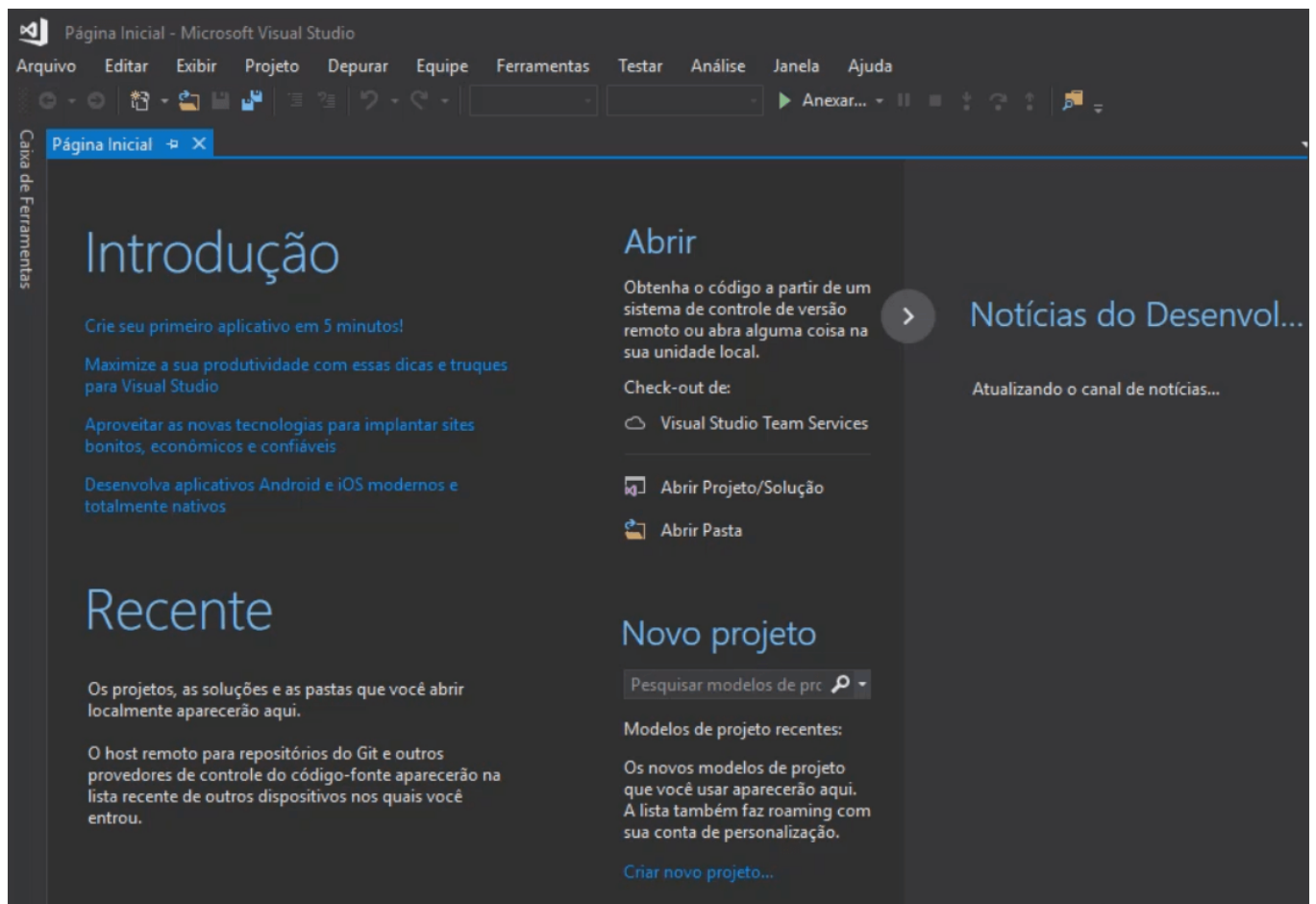
8) Após o fim da instalação, o Visual Studio irá pedir que você se conecte com uma conta da Microsoft, mas não faça isto por enquanto. Escolha a opção **Agora não, talvez mais tarde**:



9) Escolha o seu tema do ambiente de desenvolvimento:



10) Pronto, agora o Visual Studio está instalado na sua máquina:



11) Abra o Visual Studio e construa dois programas: um em **Visual Basic .NET** e outro em **C#**, que faça as mesmas coisas:

- **Visual Basic .NET:**

```
Module Module1
```

```
    Sub Main()
```

```
        Dim X As Integer = 1
```

```
        Dim Y As Integer = 2
```

```
        Dim Z As Integer
```

```
        Z = X + Y
```

```

        Console.WriteLine(Z)
    End Sub

End Module

```

- C#:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {

            int X = 1;
            int Y = 2;
            int Z;

            Z = X + Y;

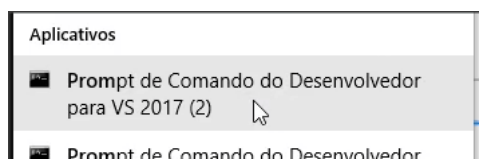
            Console.WriteLine(Z);

        }
    }
}

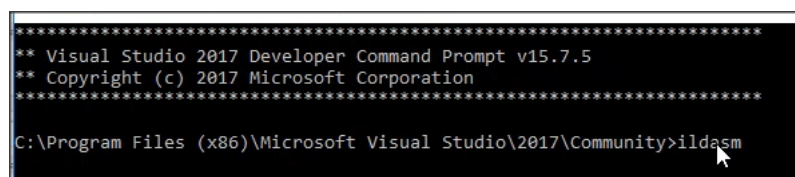
```

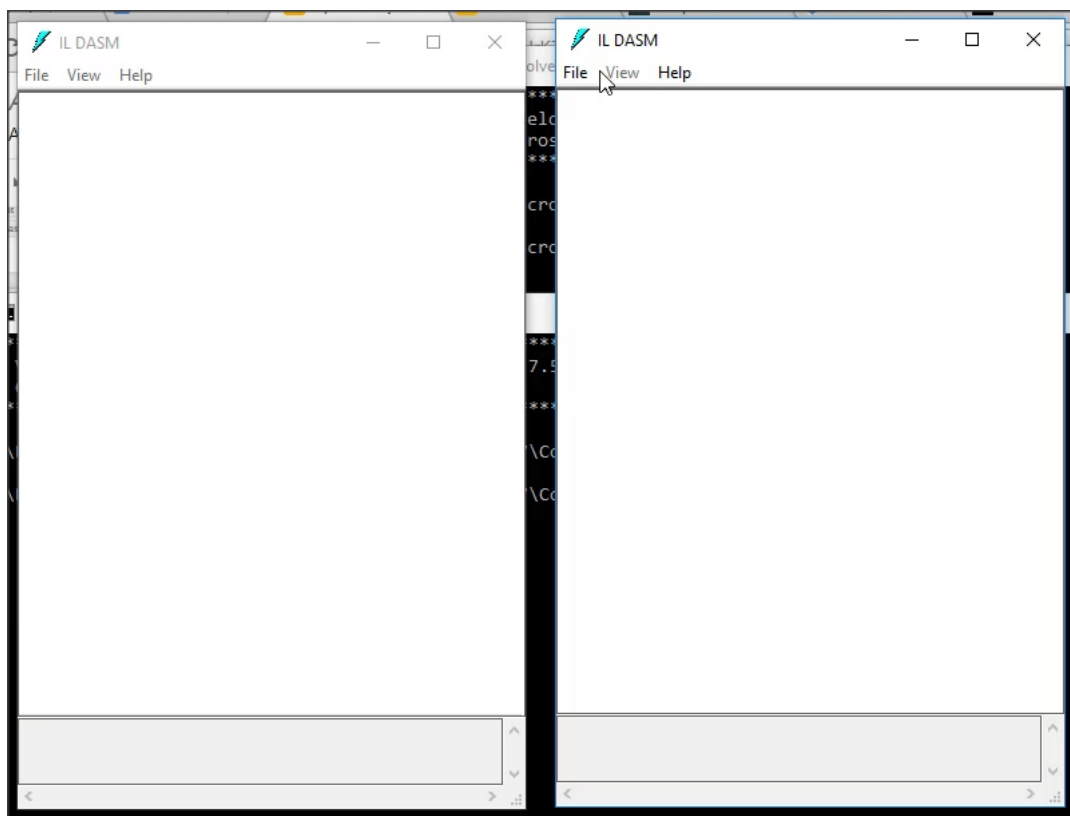
12) Com esses dois projetos prontos, gere os respectivos arquivos executáveis para a execução desses programas.

13) Depois disso abra duas janelas usando a aplicação **Prompt de Comando do Desenvolvedor para o VS 2017**:

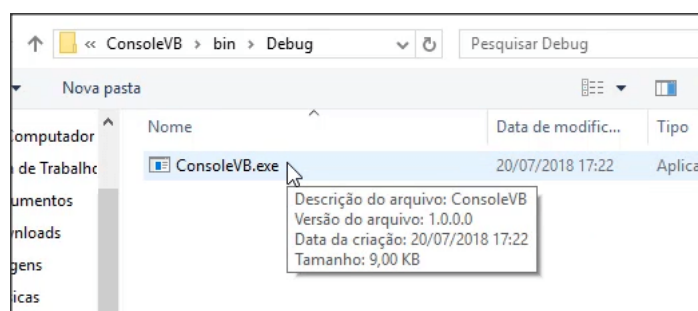


14) Em ambas as janelas, execute o programa `ildasm` :

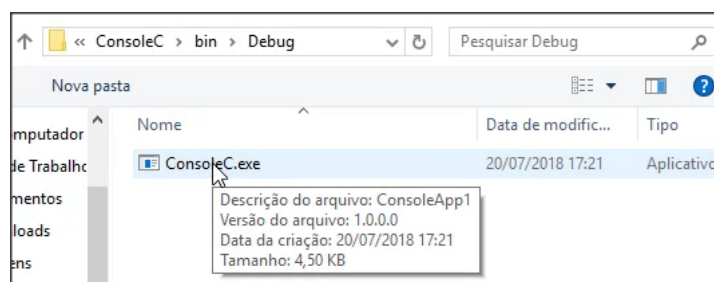




15) Na primeira janela, no menu, clique em **File -> Open** e escolha o executável do projeto feito em **Visual Basic .NET**:



16) Na outra, abra o executável do projeto desenvolvido em **C#**:



17) Depois, nas duas janelas, dê um duplo clique na pasta **MANIFEST**:



18) Assim, você pode visualizar o código interpretado pelo **.NET Framework** das duas aplicações:

```
.ctor(string) = ( 01 00 00 00 00 )  
.ctor(string) = ( 01 00 09 43 6F 6E 73 6F 6C 65 56 42 00 00 ) // ...C  
::ctor(string) = ( 01 00 12 43 6F 70 79 72 69 67 68 74 20 C2 A9 20 // ..  
20 32 30 31 38 00 00 ) // 2  
::ctor(string) = ( 01 00 00 00 00 )  
ribute::ctor(bool) = ( 01 00 00 00 00 )  
::ctor(string) = ( 01 00 24 66 39 64 34 64 37 39 37 2D 63 37 63 64 // ...  
2D 34 34 38 30 2D 61 65 63 38 2D 36 64 33 32 35 // -4  
34 61 33 37 33 32 65 00 00 ) // 4a  
te::ctor(string) = ( 01 00 07 31 2E 30 2E 30 2E 30 00 00 ) //  
ribute::ctor(string) = ( 01 00 1C 2E 4E 45 54 46 72 61 6D 65 77 6F 72 68  
2C 56 65 72 73 69 6F 6E 3D 76 34 2E 36 2E 31 01  
00 54 0E 14 46 72 61 6D 65 77 6F 72 68 44 69 73  
70 6C 61 79 4E 61 6D 65 14 2E 4E 45 54 20 46 72  
61 6D 65 77 6F 72 68 20 34 2E 36 2E 31 )  
te::ctor(string) = ( 01 00 00 00 00 )  
te::ctor(string) = ( 01 00 08 43 6F 6E 73 6F 6C 65 41 70 70 31 00 00 ) //  
bute::ctor(string) = ( 01 00 12 43 6F 70 79 72 69 67 68 74 20 C2 A9 20 //.  
20 32 30 31 38 00 00 ) //  
bute::ctor(string) = ( 01 00 00 00 00 )  
eAttribute::ctor(bool) = ( 01 00 00 00 00 )  
bute::ctor(string) = ( 01 00 24 66 62 33 61 64 37 31 65 2D 38 38 34 34 //.  
2D 34 62 61 65 2D 61 33 35 35 2D 64 33 66 61 64 //.  
32 38 61 37 63 39 32 00 00 ) //  
ribute::ctor(string) = ( 01 00 07 31 2E 30 2E 30 2E 30 00 00 )  
kAttribute::ctor(string) = ( 01 00 1C 2E 4E 45 54 46 72 61 6D 65 77 6F 72  
2C 56 65 72 73 69 6F 6E 3D 76 34 2E 36 2E 31  
00 54 0E 14 46 72 61 6D 65 77 6F 72 68 44 69  
70 6C 61 79 4E 61 6D 65 14 2E 4E 45 54 20 46  
61 6D 65 77 6F 72 68 20 34 2E 36 2E 31 )
```

Fica claro concluir que, apesar de ter desenvolvido duas aplicações fazendo a mesma coisa, mas usando linguagens de programação diferentes, o interpretador **.NET Framework** traduz as duas aplicações em praticamente o mesmo código. E é este código que será executado no hardware da máquina.