



Residência em Software

Residência em Tecnologia da Informação e Comunicação

APRESENTAÇÃO DO CURSO

Professores:

Alvaro Degas Coelho,
Edgar Alexander,
Esbel Thomas Valero,
Helder Conceição Almeida

INSTITUIÇÃO EXECUTORA



UESC

COORDENADORA



APOIO

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO



Aula 1 - Apresentação do Curso

- Sistema de avaliação do curso
- Linguagens Imperativas:
 - O que são, quais as principais linguagens utilizadas no mercado.
 - A linguagem C++ e como ela está inserida no mercado atual de desenvolvimento de software.
 - Histórico e características de C/C++
- Ferramentas que serão utilizadas durante o curso:
 - O VS Code como IDE para desenvolvimento de software em C++.
 - Alguns exemplos simples em C++
 - Como editar, compilar e executar eles no VS Code.

Sistema de Avaliação do Curso

- A avaliação se dará da seguinte forma:
 - **30% participação ativa** nas aulas presenciais, participação nos fóruns de discussão
 - **30% entregas realizadas** (cumprimento de prazos, adequação das entregas realizadas)
 - **40% avaliação final realizada pelo professor**
 - Uma prova prática individual
 - Uma prova prática em grupo

Linguagem de Programação

O Paradigma Imperativo

- Foco principal é em descrever explicitamente o "como" uma tarefa deve ser realizada.
- O programador escreve uma sequência de comandos, indicando ao computador quais ações executar e em qual ordem.
- As ações geralmente incluem atribuições de valores a variáveis, estruturas de controle (como loops e condicionais) e alterações diretas no estado do programa e dos dados.

Características do Paradigma Imperativo

- **Instruções sequenciais**

- As ações são executadas na ordem em que foram escritas no código, seguindo uma sequência de passos.

- **Variáveis e estado mutável**

- São utilizadas variáveis para armazenar e modificar dados temporários, e o estado do programa pode ser alterado ao longo da execução.

- **Foco em algoritmos**

- O programador descreve diretamente os algoritmos e detalhes de implementação para atingir um objetivo específico.

- **Controle explícito**

- O controle do fluxo de execução é explicitamente definido pelo programador, utilizando estruturas de controle como loops e condicionais.

Linguagens Imperativas

- As **linguagens de programação imperativas** se baseiam na noção de que um programa consiste em **uma sequência de comandos que alteram o estado do computador**.
- Nesse paradigma, **o programador especifica exatamente como as tarefas devem ser realizadas**, descrevendo o fluxo de controle e os passos para alcançar o resultado desejado.
- **As instruções são executadas na ordem em que foram escritas**, e o foco principal é em "como" as coisas devem ser feitas.

Principais características das linguagens imperativas:

- **Variáveis**

- É possível declarar e utilizar variáveis para armazenar dados e valores temporários.

- **Estruturas de controle**

- Oferecem estruturas como loops (laços) e condicionais para controlar o fluxo do programa.

- **Procedimentos e funções**

- Permitem dividir o código em partes menores e reutilizáveis através de procedimentos (ou rotinas) e funções.

- **Modificação do estado**

- As instruções podem alterar o estado do programa e das variáveis.

- **Memória explícita**

- A manipulação da memória é geralmente feita de forma explícita, através do uso de ponteiros ou referências.



Residência
em Software

Programação Imperativa

Os programas baseados em programação imperativa são focados em instruções exatas que devem ser passadas ao computador na sequência em que serão executadas.

Algumas das principais linguagens imperativas utilizadas no mercado

● Linguagem C

- Lançada em 1972, é uma linguagem de programação imperativa e procedural que teve grande influência no desenvolvimento de sistemas operacionais e outras aplicações de baixo nível.
- É conhecida por oferecer controle direto sobre a memória e recursos do computador.

Exemplo:

```
#include <stdio.h>
int main() {
    printf("Olá, mundo!\n");
    return 0;
}
```

Algumas das principais linguagens imperativas utilizadas no mercado

● Linguagem C++

- Criada como uma extensão da linguagem C, o C++ é uma linguagem multiparadigma que inclui recursos da programação orientada a objetos.
- É amplamente usada em desenvolvimento de jogos, sistemas embarcados, aplicativos de alto desempenho e outros projetos de larga escala.

Exemplo:

```
include <iostream>
int main() {
    std::cout << "Olá, mundo!" << std::endl;
    return 0;
}
```

Algumas das principais linguagens imperativas utilizadas no mercado

● Linguagem Java

- Introduzida em 1995 pela Sun Microsystems (adquirida pela Oracle), Java é uma linguagem de programação imperativa e orientada a objetos, projetada para ser portátil e independente de plataforma.
- É amplamente utilizada em desenvolvimento web, aplicações empresariais e dispositivos móveis Android.

Exemplo:

```
public class OlaMundo {  
    public static void main(String[] args) {  
        System.out.println("Olá, mundo!");  
    }  
}
```

Algumas das principais linguagens imperativas utilizadas no mercado

● Linguagem Python

- Embora seja mais conhecida como uma linguagem de programação multi-paradigma, Python também suporta programação imperativa.
- Ela se tornou muito popular por sua sintaxe simples e legibilidade, sendo usada em diversas áreas, como desenvolvimento web, análise de dados, inteligência artificial, entre outros.

Exemplo:

```
print("Olá, mundo!")
```

Algumas das principais linguagens imperativas utilizadas no mercado

● Linguagem Javascript

- Inicialmente criada como uma linguagem de scripting para páginas web, o JavaScript se expandiu e se tornou uma linguagem poderosa para o desenvolvimento de aplicativos front-end e back-end.
- Possui paradigma imperativo, além de oferecer suporte a programação funcional e orientada a objetos.

Exemplo:

```
<!DOCTYPE html>
<html>
<head>
  <title>Olá, mundo!</title>
</head>
<body>
  <script>
    // JavaScript
    alert("Olá, mundo!");
  </script>
</body>
</html>
```

Algumas das principais linguagens imperativas utilizadas no mercado

● Linguagem Go (Golang)

- Criada pela Google, Go é uma linguagem projetada para ser rápida, eficiente e fácil de trabalhar.
- Ela oferece recursos da programação imperativa e concorrência embutida, sendo bastante utilizada em desenvolvimento de servidores e aplicativos de alta performance.

Exemplo:

```
package main

import "fmt"

func main() {
    fmt.Println("Olá, mundo!")
}
```



Residência
em Software

Linguagens de programação mais populares

Rank	Change	Language	Share	1-year trend
1		Python	27.43 %	-0.2 %
2		Java	16.19 %	-1.0 %
3		JavaScript	9.4 %	-0.1 %
4		C#	6.77 %	-0.3 %
5		C/C++	6.44 %	+0.2 %
6		PHP	5.03 %	-0.4 %

Fonte: <https://pypl.github.io/PYPL.html>

Outros paradigmas de programação

- Além do paradigma imperativo, existem outros paradigmas de programação, cada um com suas próprias características e abordagens para resolver problemas computacionais.
- Abaixo estão alguns dos principais paradigmas de programação:
 - Paradigma Funcional
 - Paradigma Orientado a Objetos (POO)
 - Paradigma Declarativo
 - Paradigma Reativo
 - Paradigma Lógico
 - Paradigma Concorrente e Paralelo

Funcional

- No paradigma funcional, o foco principal é na avaliação de funções matemáticas e na aplicação de transformações de dados imutáveis.
- As funções são tratadas como cidadãos de primeira classe (recurso que permite armazenar a função como valor de uma constante ou variável), o que significa que podem ser passadas como argumentos para outras funções e retornadas como resultados.
- A programação funcional evita efeitos colaterais e modificações diretas em variáveis, buscando imutabilidade e a escrita de funções puras.
- Exemplo: Haskell, Erlang, Clojure, Scala, F# e Lisp

Orientado a Objetos

- No paradigma orientado a objetos, o programa é estruturado em torno de objetos, que são instâncias de classes.
- Cada objeto contém dados (atributos) e comportamentos (métodos) relacionados.
- A programação orientada a objetos enfatiza a abstração, encapsulamento, herança e polimorfismo, permitindo a modelagem de entidades do mundo real em código.
- Exemplo: Java, C++, C#, Python, Ruby, Swift e etc

Declarativo

- No paradigma declarativo, o programador descreve o que deseja alcançar, sem especificar explicitamente como fazê-lo.
- Ele se concentra em declarar as relações lógicas, restrições e propriedades do problema, deixando que o sistema de execução encontre a melhor forma de resolver o problema.
- Exemplo: SQL, Prolog, Datalog e etc...

Reativo

- É uma forma de programar a eventos ou alterações nos estados da aplicação.
- A sintaxe do código e as funções executadas pelo programa são feitas de forma a responder determinados eventos.
- Um exemplo onde observamos o paradigma Reativo sendo utilizado é no desenvolvimento de aplicativos para Android.
- Outro exemplo, a linguagem de programação JavaScript, que conta com funções multi-paradigmas, ao ter a biblioteca React implementada, permite que se desenvolva programas reativos também.

Lógico

- O paradigma lógico é baseado na lógica de primeira ordem, onde o programa consiste em regras lógicas e fatos.
- O programador especifica relações entre fatos e como inferir novos fatos baseados nessas regras.
- A execução do programa envolve a busca por soluções através da prova de teoremas.
- Exemplo: Prolog, Datalog e etc...

Concorrente e paralelo

- Esse paradigma lida com a execução simultânea de tarefas, onde várias partes do programa podem ser executadas em paralelo ou de forma concorrente para melhorar a eficiência e o desempenho.
- Exemplo de suporte nativo: Go, Erlang, Scala e etc

Linguagem C

- A linguagem de programação C é uma das mais influentes e amplamente utilizadas na história da computação.
- Foi desenvolvida por Dennis Ritchie no Bell Labs durante a década de 1970 como uma evolução da linguagem de programação B, criada por Ken Thompson.
- O desenvolvimento do C ocorreu simultaneamente ao desenvolvimento do sistema operacional UNIX.
- A linguagem C é conhecida por sua eficiência, portabilidade e capacidade de acesso direto à memória, tornando-a uma escolha popular para sistemas embarcados, desenvolvimento de software de sistema e aplicativos que requerem alto desempenho.
- Sua influência pode ser vista em muitas outras linguagens de programação que foram desenvolvidas posteriormente.

Histórico Resumido

- Anos 1960:
 - Ken Thompson desenvolveu a linguagem de programação B, uma linguagem de alto nível baseada em BCPL (Basic Combined Programming Language), para programar o sistema operacional Multics.
- Início dos anos 1970:
 - Dennis Ritchie, trabalhando nos Laboratórios Bell, criou uma linguagem chamada "NB" (New B), que mais tarde evoluiu para a linguagem C. A linguagem NB permitia a manipulação de hardware e a programação do sistema operacional, tornando-a mais poderosa e flexível que a linguagem B.
- 1972:
 - O sistema operacional UNIX é reescrito na linguagem C por Dennis Ritchie e Ken Thompson. Isso permitiu que o UNIX fosse facilmente portado para diferentes plataformas de hardware, facilitando sua adoção e propagação.
- 1973:
 - A linguagem C é formalizada e documentada por Dennis Ritchie em um artigo intitulado "The C Programming Language," coescrito com Brian Kernighan. O livro "The C Programming Language," publicado em 1978, se tornou a referência padrão para a linguagem C.
- Final dos anos 1970:
 - Com a crescente popularidade do UNIX, a linguagem C começou a se espalhar amplamente e a ganhar suporte em várias plataformas de hardware e sistemas operacionais.

Histórico Resumido

- 1983:
 - A American National Standards Institute (ANSI) estabelece um comitê para padronizar a linguagem C. Isso levou à criação do padrão ANSI C, formalmente conhecido como ANSI X3.159-1989.
- 1989:
 - O padrão ANSI C é adotado como um padrão internacional pela International Organization for Standardization (ISO) e é conhecido como ISO/IEC 9899:1989. Este padrão definiu a linguagem C que é amplamente utilizada até hoje.
- 1990s e além:
 - A linguagem C continuou a ser uma das linguagens de programação mais populares e influentes. Ela foi amplamente utilizada para desenvolvimento de sistemas operacionais, compiladores, bibliotecas, drivers de dispositivos e uma variedade de outros aplicativos.
- C99 e C11:
 - A linguagem C passou por revisões subsequentes, com o lançamento dos padrões C99 (ISO/IEC 9899:1999) e C11 (ISO/IEC 9899:2011), que adicionaram novos recursos e melhorias à linguagem.

Linguagem C++

- A linguagem de programação C++ foi criada como uma extensão da linguagem C e se tornou uma das linguagens de programação mais populares e amplamente utilizadas em todo o mundo.
- Seu desenvolvimento começou nos anos 1980 e desde então passou por várias revisões e aprimoramentos.
- O objetivo da linguagem de programação C++ é fornecer uma extensão da linguagem C com recursos adicionais de programação orientada a objetos (POO) e programação genérica, enquanto mantém a eficiência e o controle direto de memória característicos da linguagem C.
- C++ é uma linguagem poderosa, versátil e amplamente utilizada que combina eficiência com recursos de alto nível, tornando-a adequada para uma ampla variedade de aplicações e cenários de desenvolvimento de software.

Principais objetivos de C++

1. Orientação a objetos:

- C++ permite que os desenvolvedores usem conceitos de POO, como encapsulamento, herança e polimorfismo, para criar programas com uma abordagem mais modular e reutilizável. Isso ajuda a organizar o código em unidades independentes chamadas de classes, o que facilita a manutenção e a extensão do software.

2. Compatibilidade com C:

- C++ é compatível com a linguagem C, o que significa que os programas em C podem ser compilados e executados em um compilador de C++. Além disso, o C++ pode usar bibliotecas e funções escritas em C, permitindo que os desenvolvedores reutilizem código existente.

3. Eficiência e controle de memória:

- C++ permite que os desenvolvedores tenham controle direto sobre a alocação e desalocação de memória, o que é crucial em cenários em que o desempenho é uma prioridade, como desenvolvimento de sistemas operacionais, jogos e aplicativos de tempo real.

4. Programação genérica:

1. C++ suporta programação genérica através de templates. Essa característica permite criar algoritmos e estruturas de dados que podem ser usados com diferentes tipos de dados, aumentando a flexibilidade e a reutilização de código.

Principais objetivos de C++

5. Portabilidade:

- C++, assim como C, é uma linguagem de nível médio, isso significa que os programas escritos em C++ podem ser portados facilmente para diferentes plataformas, desde sistemas embarcados até servidores de alto desempenho.

6. Abstração de hardware:

- Embora mantenha o acesso direto à memória e outras características de baixo nível, C++ também permite que os desenvolvedores criem abstrações de hardware através da orientação a objetos, facilitando a criação de programas que podem ser facilmente adaptados a diferentes ambientes.

7. Escalabilidade:

- C++ é usado em uma ampla variedade de domínios, desde pequenos programas até projetos complexos e de grande escala. Sua flexibilidade e recursos avançados permitem que ele seja uma escolha adequada para projetos de diferentes tamanhos e requisitos.

Breve histórico da linguagem C++

- **Anos 1970:**

- Bjarne Stroustrup, um pesquisador dinamarquês, trabalhava no Laboratório Bell da AT&T (mesmo local onde a linguagem C foi desenvolvida).
- Ele desenvolveu a linguagem "C with Classes" (C com Classes) como uma extensão da linguagem C, com o objetivo de adicionar recursos de orientação a objetos.

- **Anos 1980:**

- A linguagem "C with Classes" foi renomeada para C++ (lê-se "C plus plus") por Bjarne Stroustrup em 1983.
- O nome "C++" representa um incremento em relação à linguagem C.

- **1985:**

- O compilador C++ da AT&T, chamado Cfront, foi lançado e tornou-se uma das primeiras implementações da linguagem.
- Nesse mesmo ano, foi publicada uma referência formal da linguagem C++ no livro "The C++ Programming Language," também escrito por Bjarne Stroustrup.

- **1989:**

- O padrão ANSI C++ foi proposto para padronizar a linguagem C++.
- No entanto, a versão final do padrão foi lançada apenas em 1998.

Breve histórico da linguagem C++

- 1998:
 - O padrão ANSI/ISO C++98 (ISO/IEC 14882:1998) foi lançado. Esse padrão trouxe melhorias significativas à linguagem e estabeleceu a base para a evolução futura do C++.
- Anos 2000:
 - Várias revisões e emendas ao padrão C++98 foram lançadas.
 - Essas revisões, como C++03 e C++11, adicionaram novos recursos, como suporte a threads, melhor gerenciamento de memória, recursos de programação genérica e lambdas.
- 2011:
 - O padrão C++11 (ISO/IEC 14882:2011) foi lançado, trazendo uma série de recursos importantes, incluindo suporte a movimentação semântica (move semantics), inicialização uniforme, loops baseados em intervalos, lambdas e muito mais.
- 2014:
 - O padrão C++14 (ISO/IEC 14882:2014) foi lançado, fornecendo pequenas melhorias em relação ao C++11.

Breve histórico da linguagem C++

- 2017:

- O padrão C++17 (ISO/IEC 14882:2017) foi lançado, com recursos adicionais, como estruturas de dados "std::optional" e "std::variant," fold expressions e atributos `[[nodiscard]]`.

- 2020:

- O padrão C++20 (ISO/IEC 14882:2020) foi lançado, trazendo diversas melhorias e recursos novos, incluindo conceitos, ranges, melhorias em lambdas, expressões constantes, entre outros.



Residência
em Software

Ferramentas de Programação

Ambiente de Programação

- Um ambiente de programação de computador é um conjunto de ferramentas, recursos e interfaces que permitem aos desenvolvedores criar, testar, depurar e implementar programas de computador.
- Esses ambientes fornecem uma plataforma onde os programadores podem trabalhar de forma mais eficiente e produtiva, tornando o processo de desenvolvimento de software mais ágil e acessível.
- IDE (Integrated Development Environment) ou Ambiente de Desenvolvimento Integrado é um tipo específico de ambiente de programação que combina várias ferramentas e recursos em uma única interface unificada.
- Essa integração facilita o desenvolvimento de software, pois oferece aos programadores uma plataforma completa para escrever, depurar, testar e implementar seus programas.

O uso de IDEs no desenvolvimento de programas

- O uso de uma IDE pode melhorar significativamente a produtividade do desenvolvedor, pois ela oferece ferramentas integradas para quase todas as etapas do ciclo de desenvolvimento de software.
- No entanto, algumas pessoas preferem ambientes de programação mais leves e personalizados, que podem ser compostos por editores de código e ferramentas independentes, dependendo das necessidades do projeto.
- A escolha da IDE ou ambiente de programação é uma questão de preferência pessoal e do tipo de projeto que está sendo desenvolvido.

Características das IDEs

- As IDEs são projetadas para suportar uma ou várias linguagens de programação, fornecendo recursos específicos para cada uma delas.
- Algumas das características mais comuns encontradas em uma IDE incluem:
 - **Editor de código:** Uma interface onde o código fonte é digitado. Geralmente, oferece recursos como realce de sintaxe, recuo automático, sugestões de código e correção ortográfica.
 - **Depurador:** Uma ferramenta para depurar o código, permitindo que os desenvolvedores inspecionem variáveis, executem o código passo a passo, coloquem pontos de interrupção e rastreiem a execução do programa para encontrar e corrigir erros.
 - **Compilador/Interpretador:** A IDE pode incluir um compilador para traduzir o código fonte em código de máquina ou um interpretador para executar diretamente o código, dependendo da linguagem de programação.

Características das IDEs

- **Gerenciador de projetos:** Uma ferramenta para organizar os arquivos do projeto, gerenciar dependências e configurar opções específicas do projeto.
- **Autocompletar e sugestões:** Recursos que oferecem sugestões e completam automaticamente partes do código à medida que o programador digita, economizando tempo e reduzindo erros.
- **Navegador de classes e métodos:** Permite que o programador navegue rapidamente entre as classes e métodos do projeto.
- **Integração com sistemas de controle de versão:** Alguns IDEs são integrados a sistemas de controle de versão, como o Git, para facilitar o versionamento do código.
- **Ferramentas de construção:** Alguns IDEs têm ferramentas de construção que facilitam a compilação e o empacotamento de projetos para distribuição.

Exemplos de IDEs

- **Visual Studio:**

- IDE da Microsoft, com suporte a várias linguagens como C#, C++, Python e outras.

- **Eclipse:**

- Uma plataforma de desenvolvimento de código aberto que suporta várias linguagens por meio de plugins.

- **IntelliJ IDEA:**

- Uma IDE específica para desenvolvimento em Java, mas também oferece suporte a outras linguagens por meio de plugins.

- **PyCharm:**

- IDE focada em Python, mas também disponível em outras edições para diferentes linguagens.

Visual Studio Code (VS Code)



- IDE desenvolvida pela Microsoft.
- Uma das ferramentas mais populares e amplamente usadas entre desenvolvedores de software.
- Software de código aberto licenciado sob a licença MIT.
- Conhecido por sua interface de usuário limpa e moderna, oferecendo recursos poderosos para edição, depuração e gerenciamento de projetos.
- Suporta várias linguagens de programação, incluindo JavaScript, TypeScript, Python, Java, C++, entre outras.

Algumas características do VS Code



1. Extensibilidade:

- É possível personalizar o editor instalando extensões, que adicionam funcionalidades e suporte para diferentes linguagens e frameworks. Há uma ampla variedade de extensões disponíveis na "Visual Studio Code Marketplace".

2. Integração com Git:

- O VS Code possui integração embutida com o sistema de controle de versão Git, permitindo que os desenvolvedores gerenciem seus repositórios diretamente no editor.

3. Depuração:

- O editor oferece recursos de depuração integrados que permitem aos desenvolvedores inspecionar o código em execução, definir pontos de interrupção e examinar variáveis durante a execução do programa.

4. Terminal Integrado:

- O VS Code possui um terminal integrado que permite aos desenvolvedores executar comandos diretamente dentro do editor, o que é especialmente útil para tarefas de gerenciamento de projetos e automação.

5. Auto-completar e sugestões inteligentes:

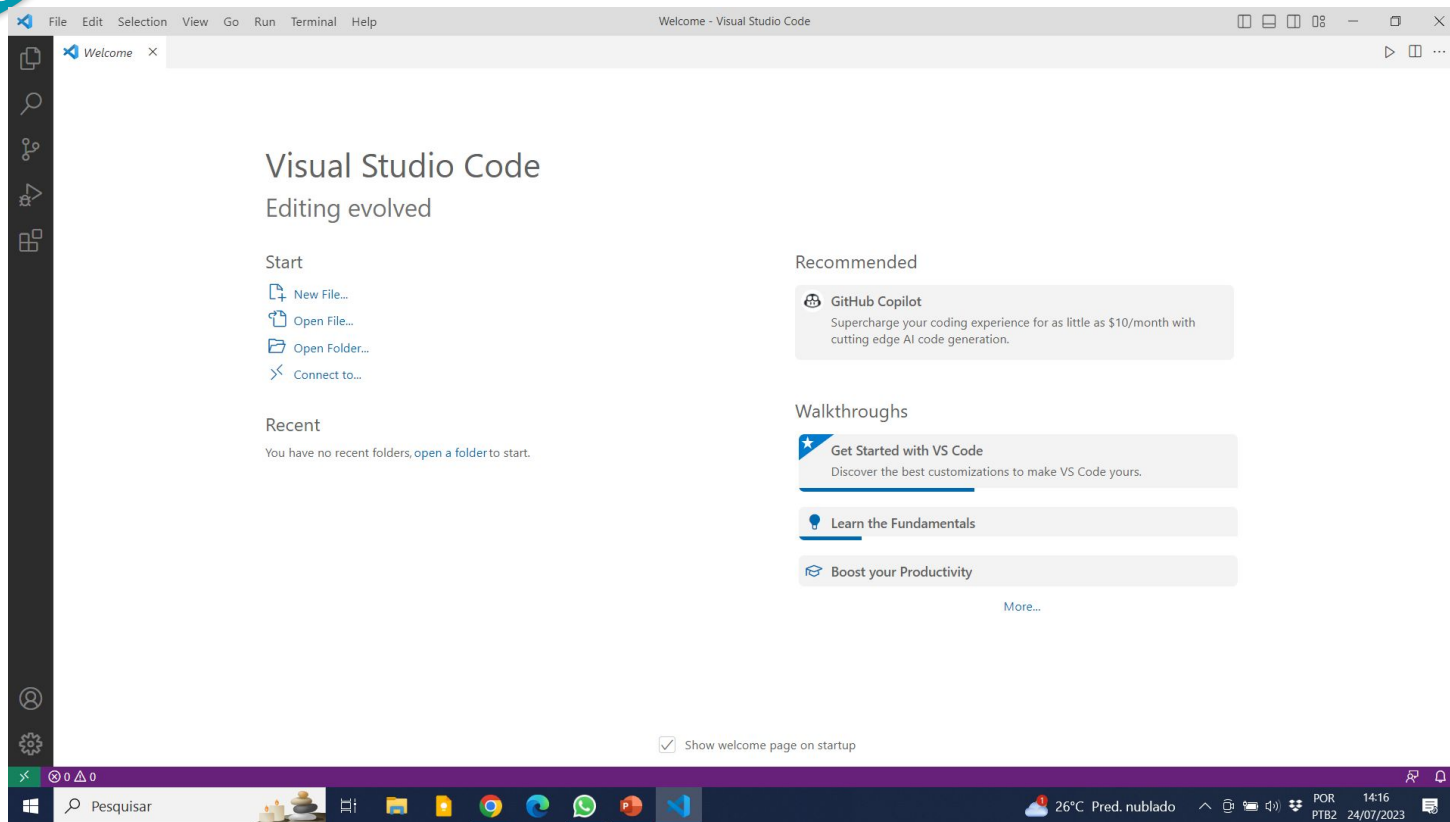
- O editor oferece sugestões automáticas e inteligentes de código à medida que você digita, o que agiliza o processo de escrita do código e reduz erros.

6. Suporte a múltiplos sistemas operacionais:

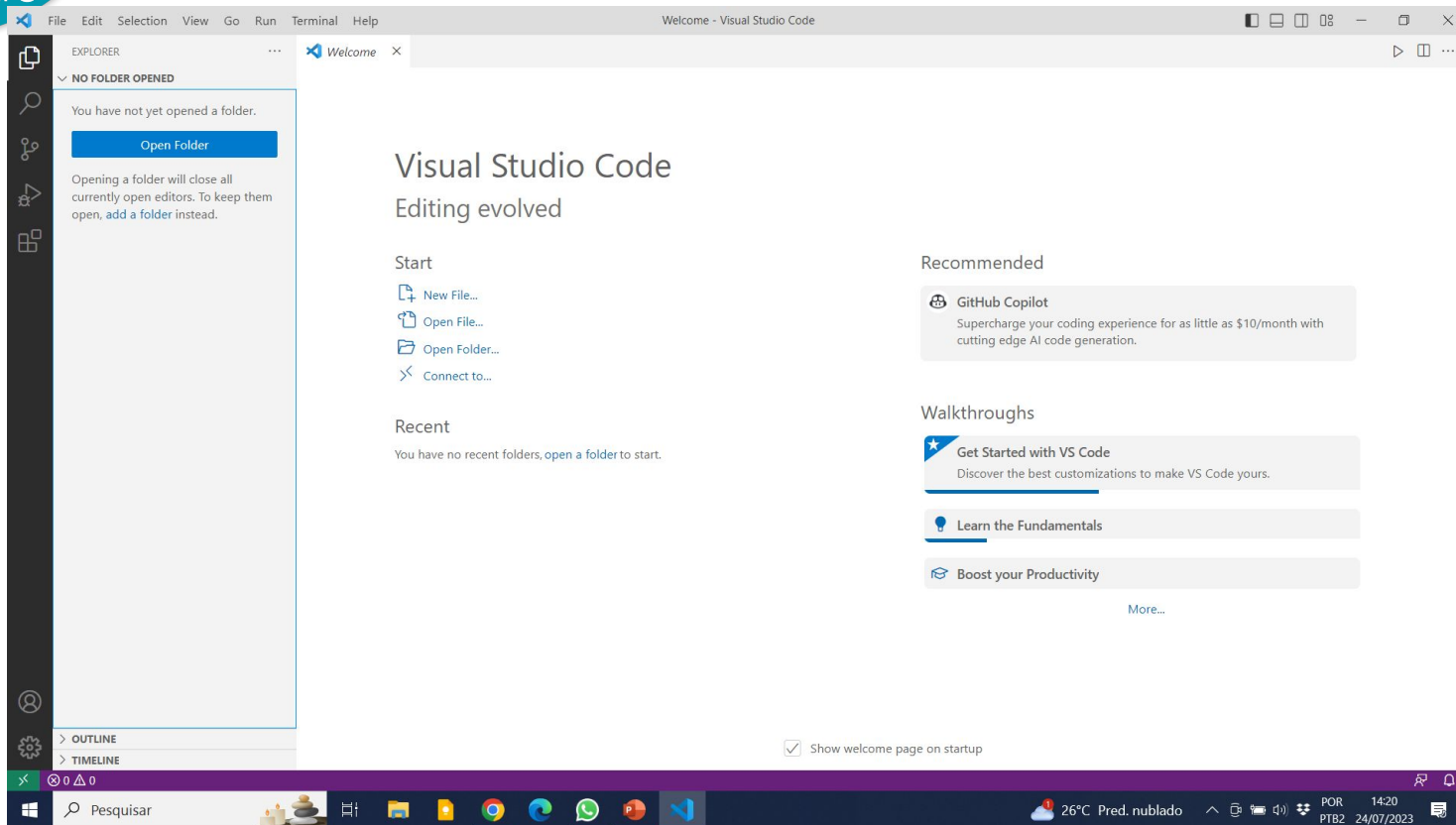
- O VS Code é compatível com Windows, macOS e Linux, tornando-o uma escolha viável para desenvolvedores que utilizam diferentes plataformas.

7. Comunidade ativa:

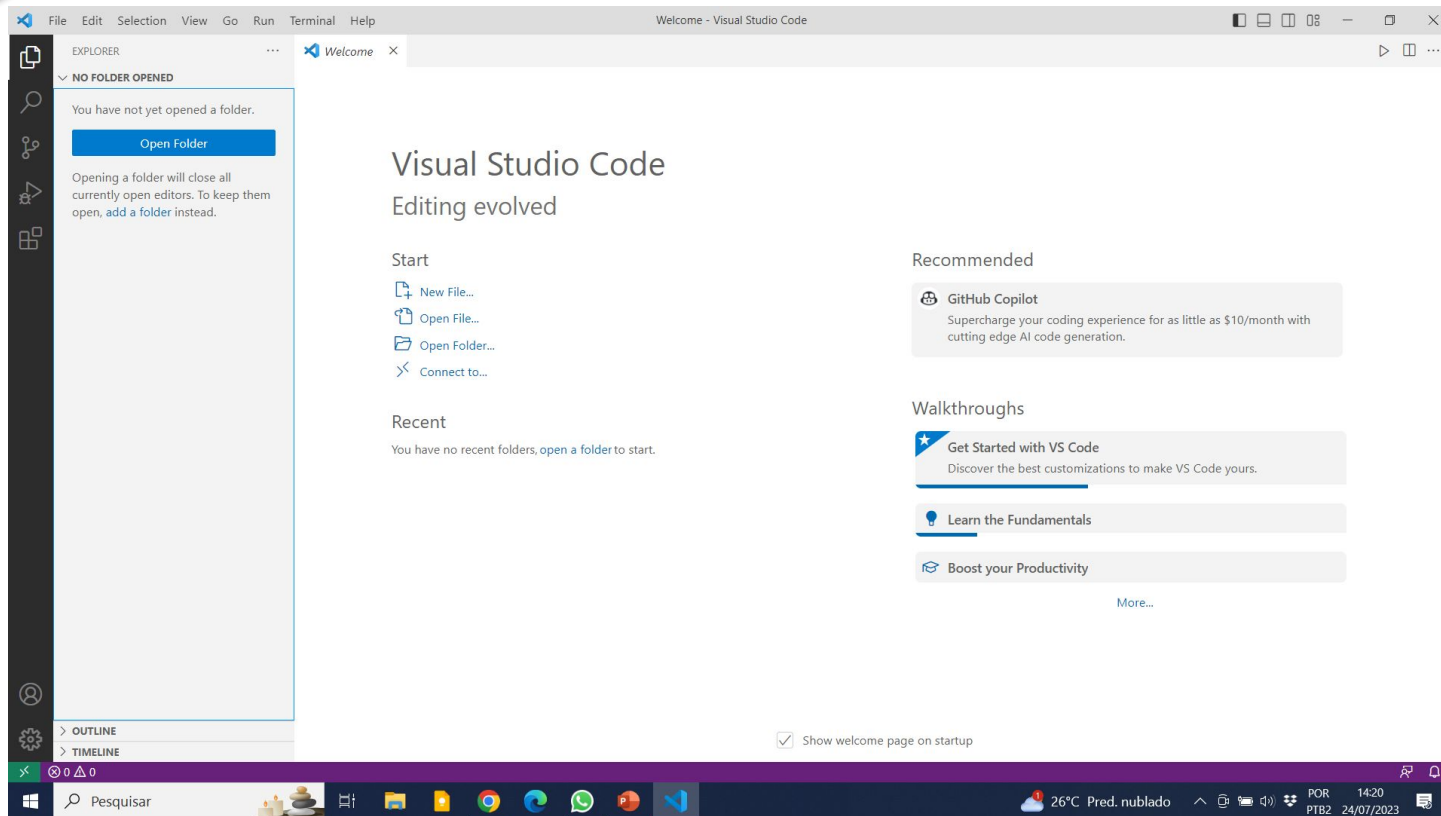
- Devido à sua popularidade, o VS Code possui uma comunidade ativa e em crescimento, o que significa que há muitos recursos, tutoriais e suporte disponíveis online.



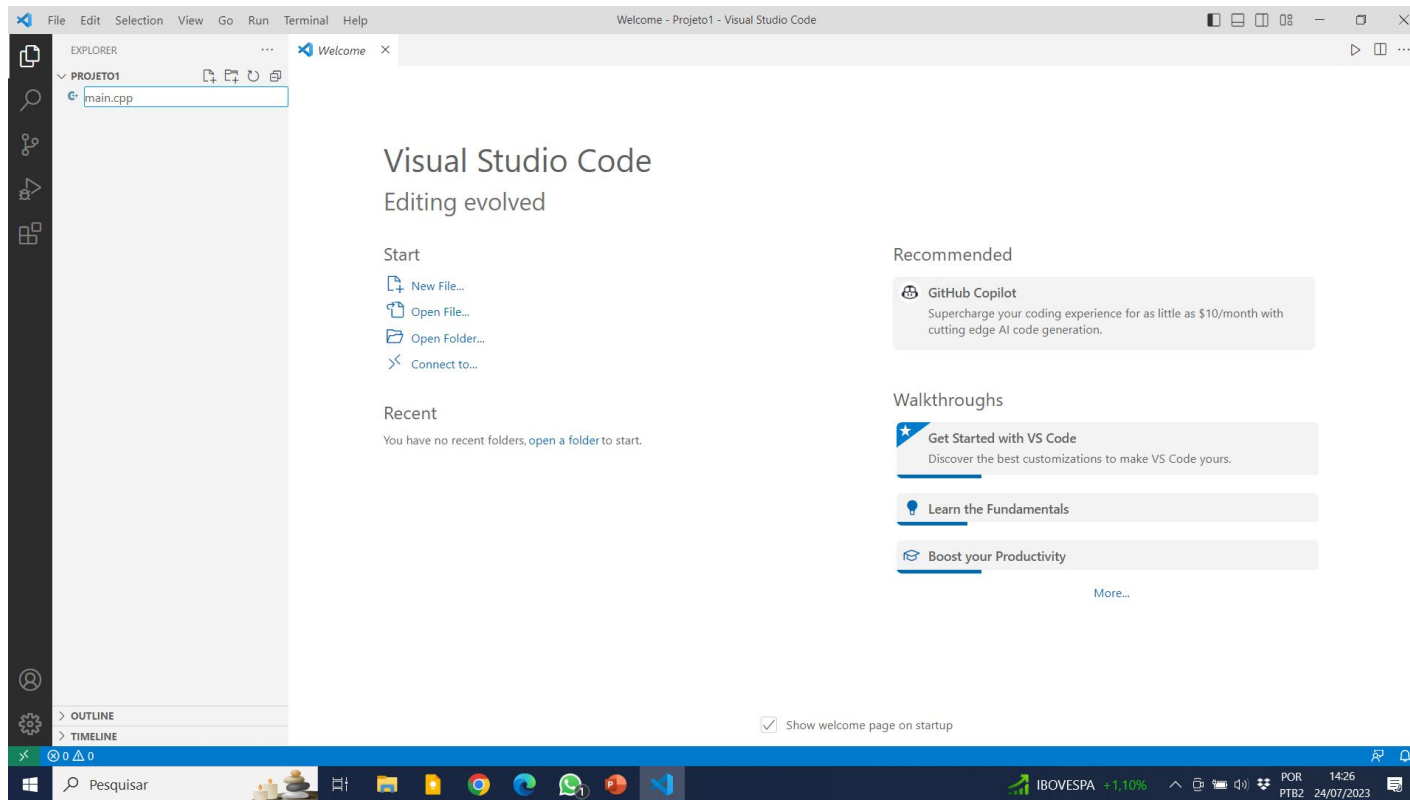
VS Code – Criando um Projeto



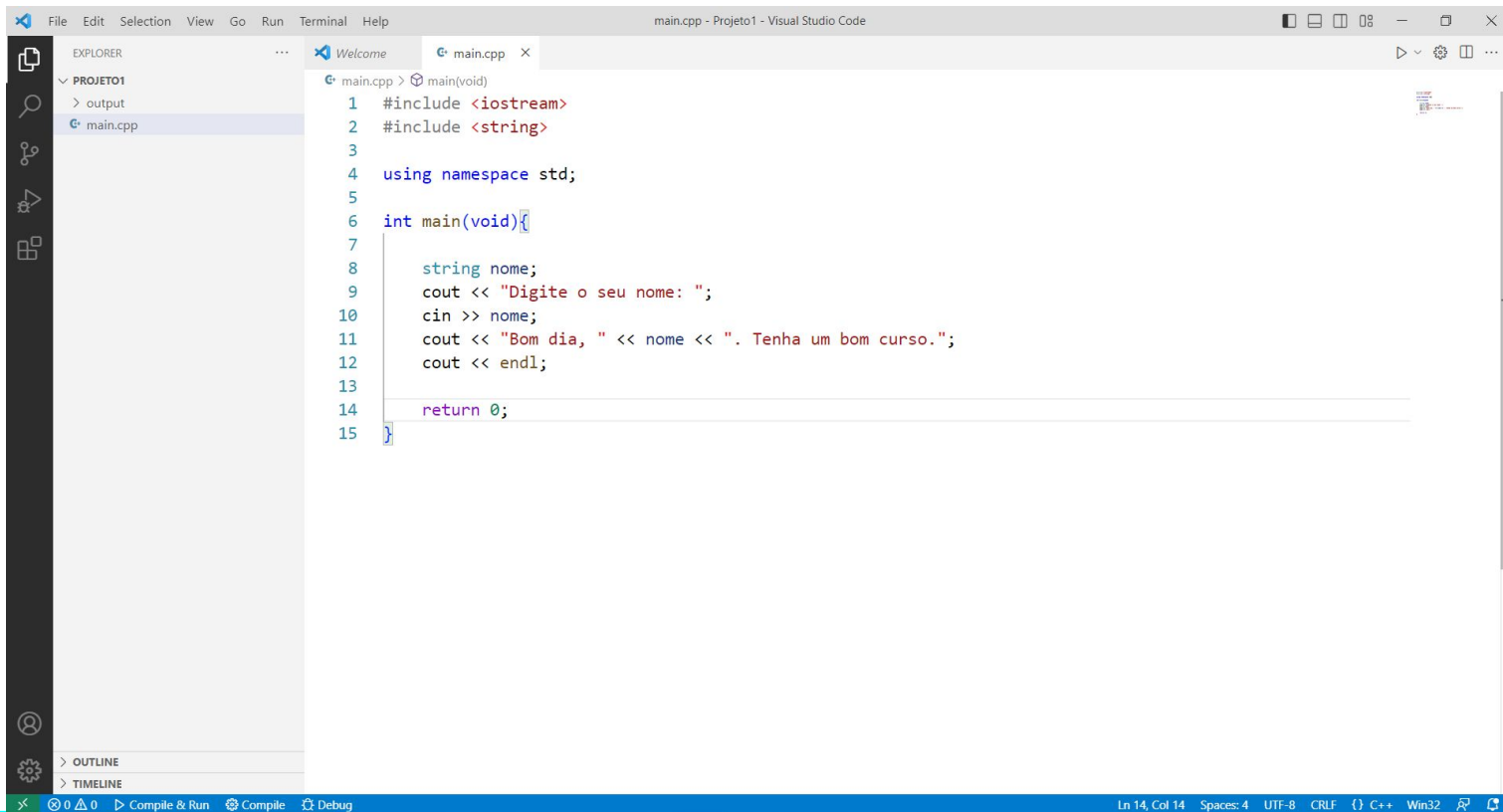
VS Code – Criando um Projeto



VS Code – Criando um Projeto

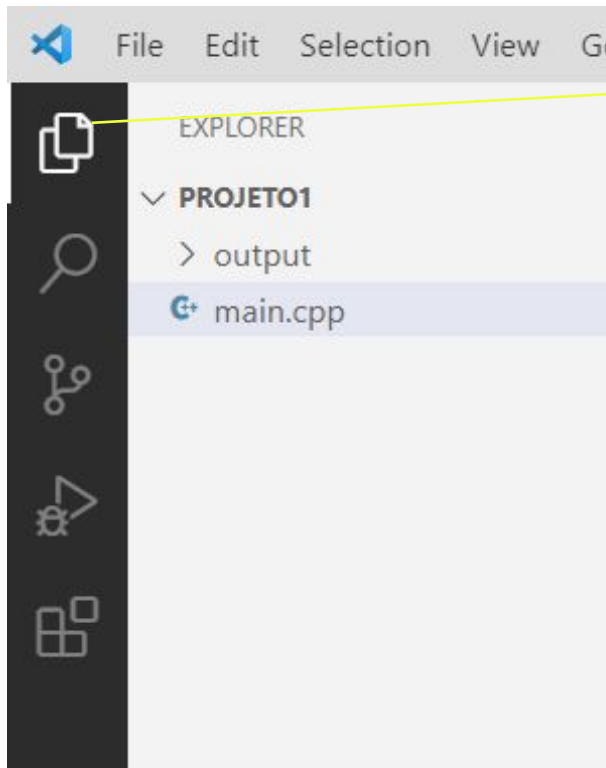


VS Code – Criando um Projeto



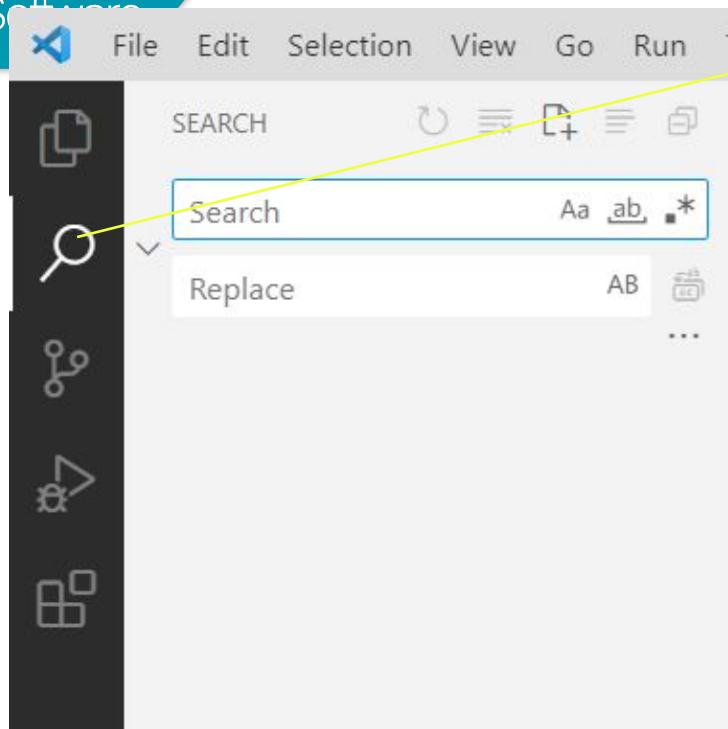
```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main(void){
7
8     string nome;
9     cout << "Digite o seu nome: ";
10    cin >> nome;
11    cout << "Bom dia, " << nome << ". Tenha um bom curso.";
12    cout << endl;
13
14    return 0;
15 }
```

VS Code – Criando um Projeto



Explorer
- Mostra as pastas e os arquivos
do projeto

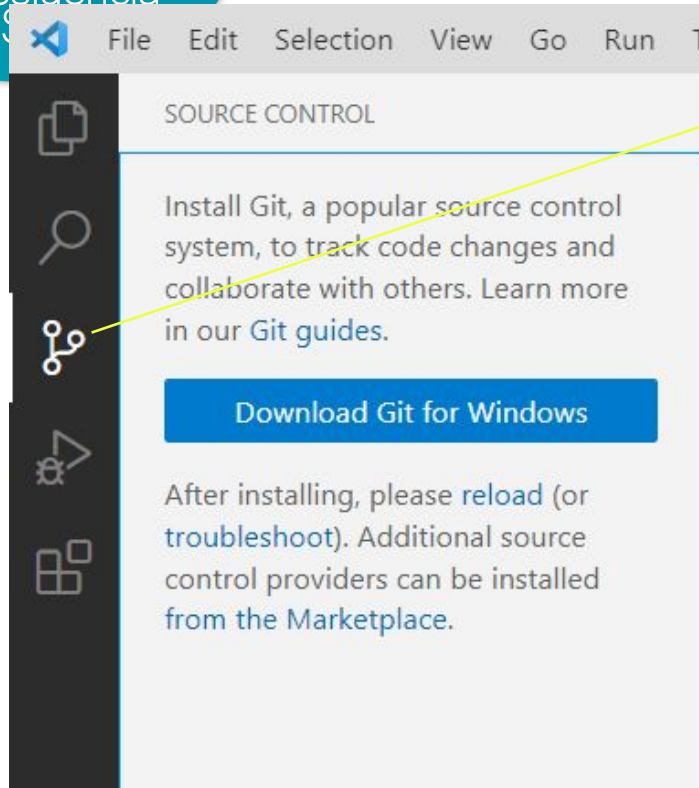
VS Code – Criando um Projeto



Search

- Procura por um termo
- Substitui um termo por outro

VS Code – Criando um Projeto

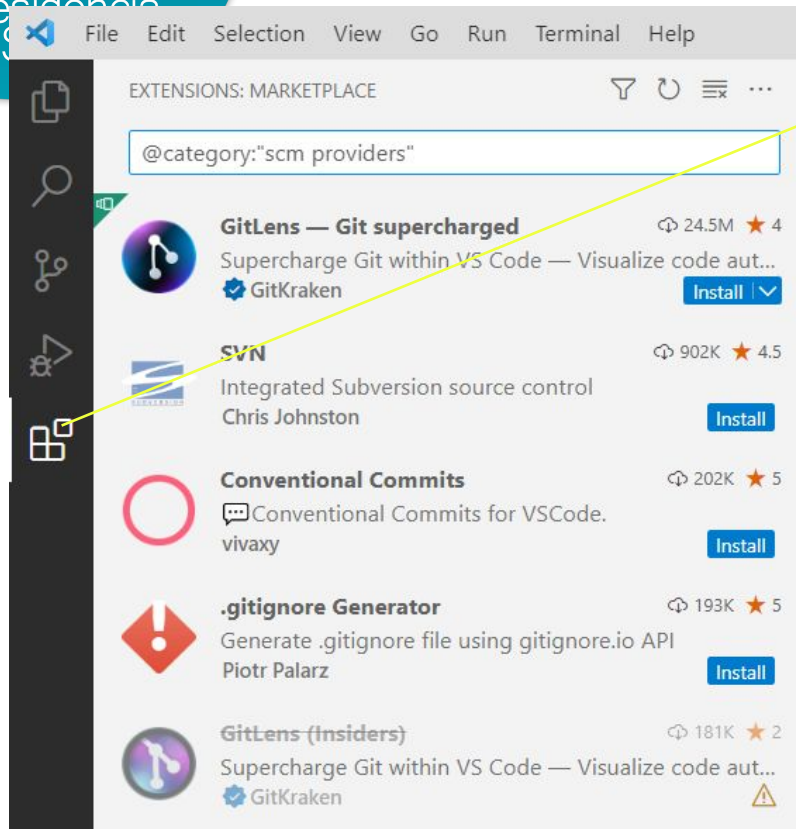


Source Control
- Sistema de Versionamento

VS Code – Criando um Projeto



Residência
em



Extensions

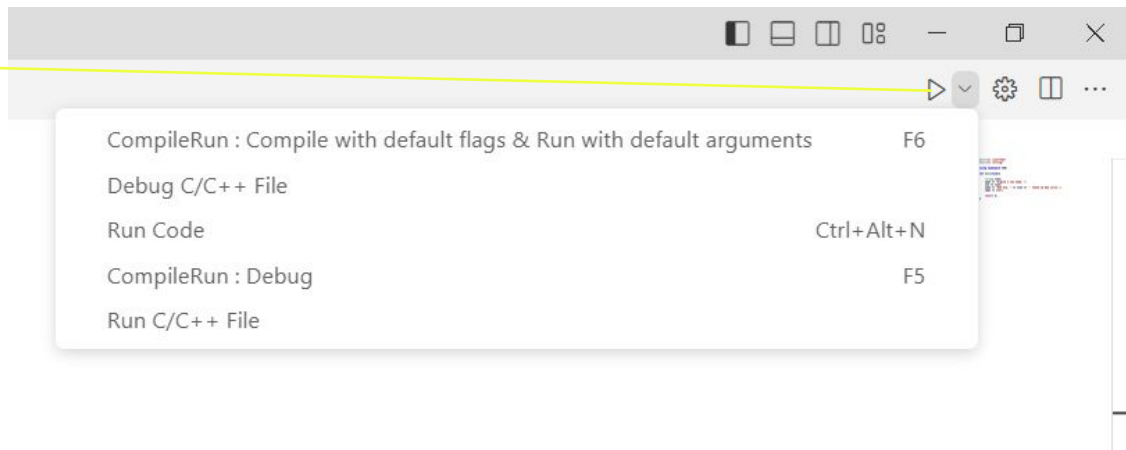
- Instalação de Extensões



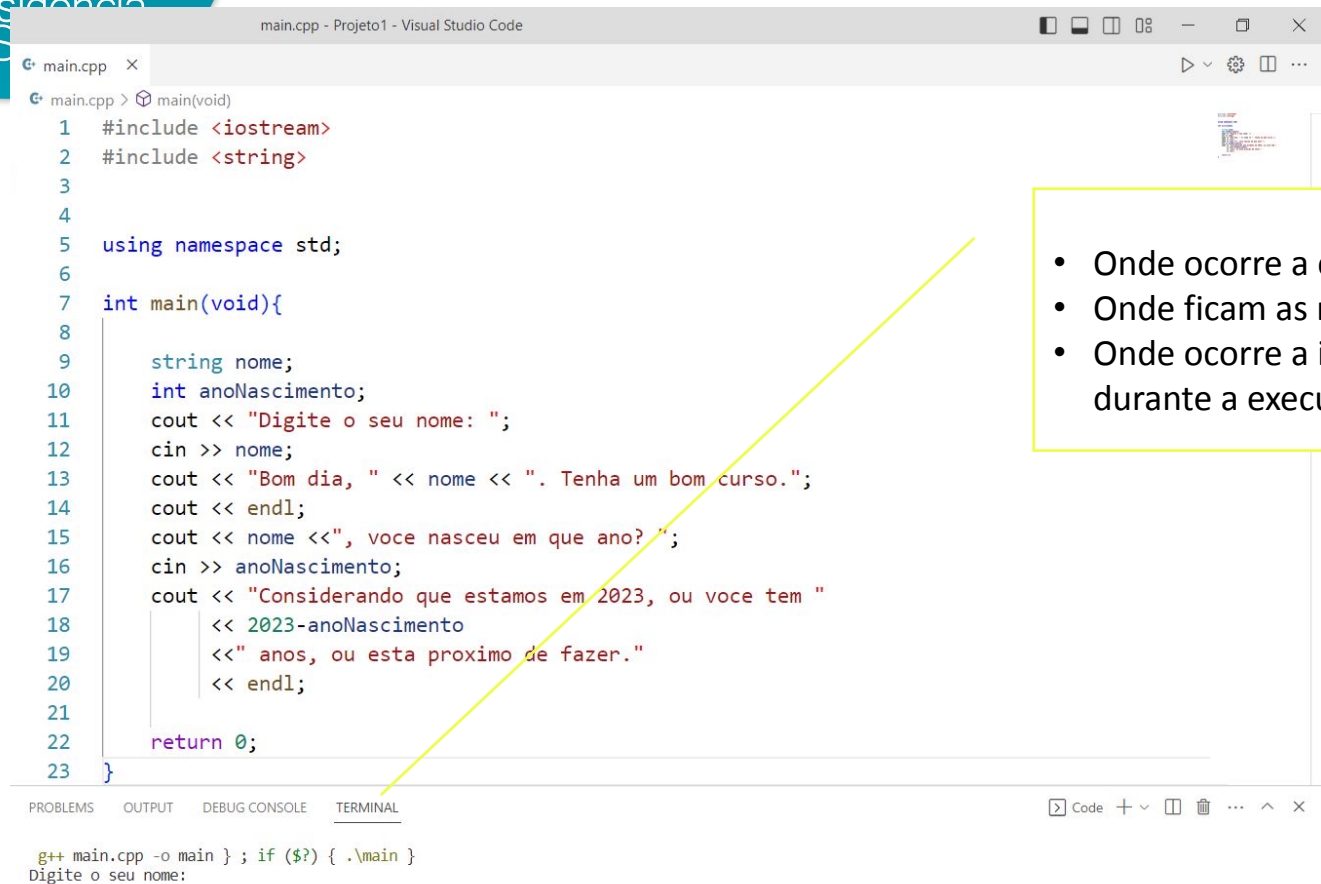
VS Code – Criando um Projeto

Residência

- Compilando um programa
- Executando um programa
- Depurando um programa



VS Code – Compilando um programa



The screenshot shows the Visual Studio Code interface. The editor window displays a C++ file named `main.cpp` with the following code:

```
1 #include <iostream>
2 #include <string>
3
4
5 using namespace std;
6
7 int main(void){
8     string nome;
9     int anoNascimento;
10    cout << "Digite o seu nome: ";
11    cin >> nome;
12    cout << "Bom dia, " << nome << ". Tenha um bom curso.";
13    cout << endl;
14    cout << nome << ", voce nasceu em que ano? ";
15    cin >> anoNascimento;
16    cout << "Considerando que estamos em 2023, ou voce tem "
17        << 2023-anoNascimento
18        << " anos, ou esta proximo de fazer."
19        << endl;
20
21    return 0;
22 }
```

The terminal at the bottom shows the command `g++ main.cpp -o main ; if ($?) { .\main }` and the prompt `Digite o seu nome:`. A yellow line points from the `main` function in the code to the terminal.

Terminal

- Onde ocorre a compilação.
- Onde ficam as mensagens de erro
- Onde ocorre a interação com o usuário durante a execução do programa.