



AED - Algoritmos e Estruturas de Dados

Aula 2 – C++ e Introdução à Orientação a Objetos

Prof. Rodrigo Mafort

Revisão Programação

- Escreva um programa que simule uma corrida de carros.
- Implemente funções para Acelerar, Frear, Virar a Direita e a Esquerda
- Exemplo:
 - Virar a Direita: Acelerar roda esquerda e frear a direita
 - Virar a Esquerda: Acelerar roda direita e frear a esquerda
 - Acelerar: Acelerar as duas rodas dianteiras
- Suponha agora que a corrida passou a admitir motos
 - Implemente funções para Acelerar, Frear, Virar a Direita e a Esquerda
 - E agora???
 - Temos duas funções acelerar?

Revisão Programação

- Suponha agora que a corrida passou a admitir motos
 - Implemente funções para:
 - Acelerar
 - Frear
 - Virar a Direita
 - Virar a Esquerda
- E agora???
- Temos duas funções acelerar?

Dados

Carro1

Placa = AAA-1234

Modelo = Audi R8

Piloto = Zé Carioca

AirBag = Ativado

...

Moto5

Placa = BBB-6789

Modelo = BMW HP4

Piloto = Pato Donald

O sistema é responsável por associar os dados as funções adequadas.

Os dados e as funções são do programa.

Mas e agora?

Temos duas funções acelerar? Qual é para carros e qual é para motos?

Funções

Acelerar

Frear

VirarEsquerda

VirarDireita

AcenderFarol

Acelerar

Frear

VirarEsquerda

VirarDireita

Motivação para Orientação a Objetos

- Quando a computação se tornou mais abrangente, resolvendo problemas cada vez mais diversos e com mais dados, a programação estruturada chegou a um limite... Os programas se tornaram muitos complexos...
- Essas limitações motivaram o desenvolvimento de um outro paradigma: Programação Orientada a Objetos

Programação
Imperativa



Programação
Estruturada



Programação
Orientada a
Objetos

Programação Orientada a Objetos

Carro
Placa
Modelo
Piloto
Airbag
Acelerar
Frear
VirarEsquerda
VirarDireita
AcenderFarol

Moto
Placa
Modelo
Piloto
Acelerar
Frear
VirarEsquerda
VirarDireita

Agrupamos os dados e as funções em um único contexto:
Objetos



Programação Estruturada

vs

Programação Orientada a Objetos

- Definimos funções e procedimentos para lidar/resolver o problema.
- O sistema é responsável por usar as funções apropriadas para cada informação (tipo)
- Os dados e as funções não são integrados. Não são modelados como uma única entidade.

- Modelamos os dados e as funções em conjunto. Eles são integrados.
- No momento do desenvolvimento já criamos uma associação entre as funções e os dados.
- O resultado desse agrupamento de dados e funções são chamados de **objetos**.

Objetos

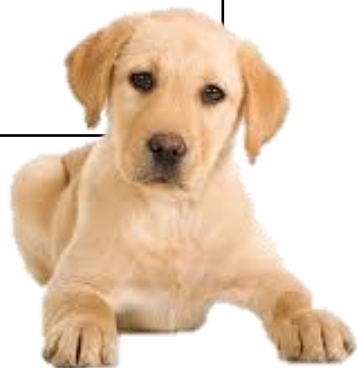
- Um objeto é uma entidade modelada pelo programador.
- Essa entidade é representada dentro dos programas como um objeto.



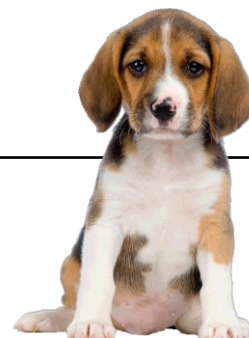
Objetos - Cachorros

Objeto - Cachorro

Cachorro
Nome: Marley Raça: Labrador Idade: 1 ano Peso: 10kg
Latir() Correr() Andar() Dormir() Comer() 💩💩()



Cachorro
Nome: Snoopy Raça: Beagle Idade: 4 meses Peso: 2 kg
Latir() Correr() Andar() Dormir() Comer() 💩💩()



As
características
e as ações dos
cachorros
foram
agrupados
como uma
única entidade:

objetos

Atributos e Métodos

- Atributos são os dados associados a cada objeto (as características)
 - Nome, Raça, Idade, Peso....
- Métodos são as ações que cada objeto pode desempenhar (as ações)
 - Latir, Correr, etc...
- Ao observar uma entidade
 - Os dados ou características são os atributos
 - As ações e formas de interagir com essa entidade são os métodos
- Na programação estruturada:
 - Atributos podem ser vistos como os parâmetros passados para as funções
 - Métodos são as funções que manipulam essas informações

Conceito de Classe

- Nos exemplos anteriores, todos os cachorros tem os mesmos atributos e métodos...
- Como definir esse modelo para todos os cachorros?
- Classes são como moldes ou formas para construir objetos



Classe Castelo de Areia

Objetos Castelo de Areia

Classes

- As classes são usadas como moldes para criar objetos
- Nos descrevemos os atributos e métodos nesse modelo

Cachorro
Nome
Raça
Idade
Peso
Latir()
Correr()
Andar()
Dormir()
Comer()
💩💩()

Todo cachorro definido a partir desse molde terá os atributos

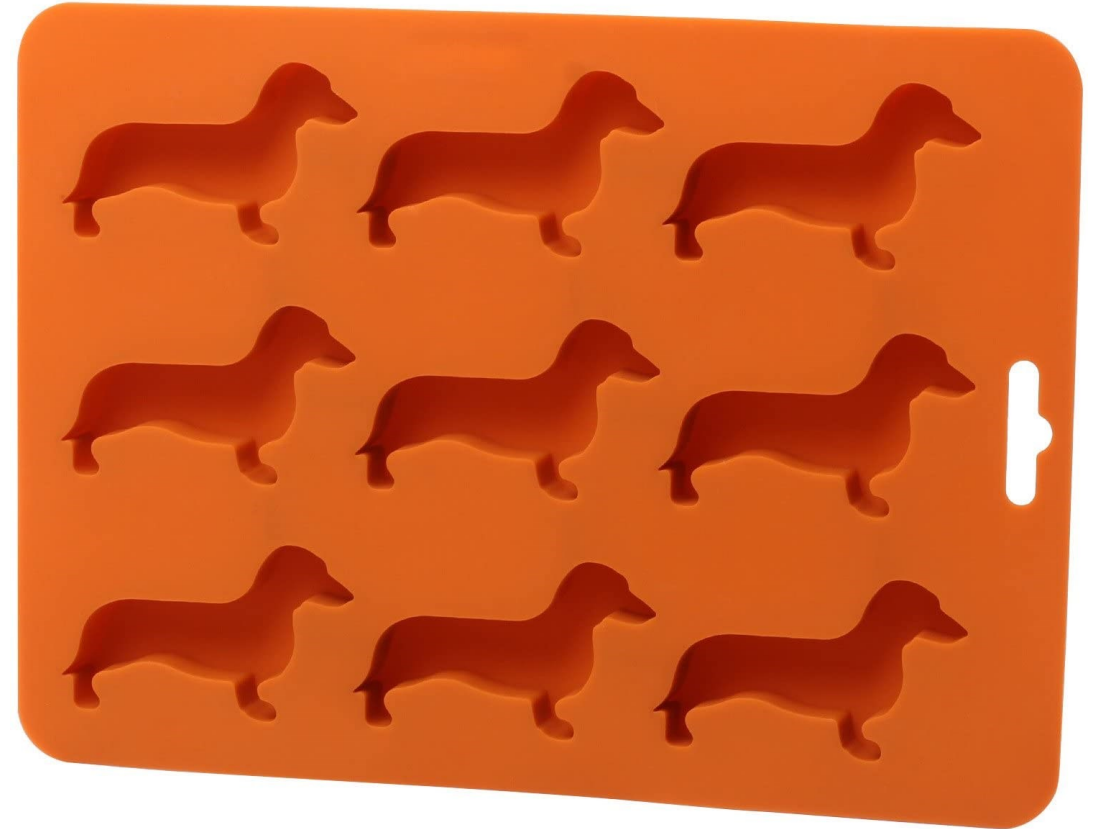
Nome, Raça, Idade e Peso

E os métodos

Latir, Correr, Andar, Comer, ...

Classes

Cachorro
Nome
Raça
Idade
Peso
Latir()
Correr()
Andar()
Dormir()
Comer() 💩💩()



Objetos vs Classes

- Classes são os modelos
 - Objetos são criados (instanciados) a partir desse modelo
 - Objetos são instâncias de classes
-
- Classe: SerHumano
 - Objetos: Eu, você, todos os alunos nessa sala



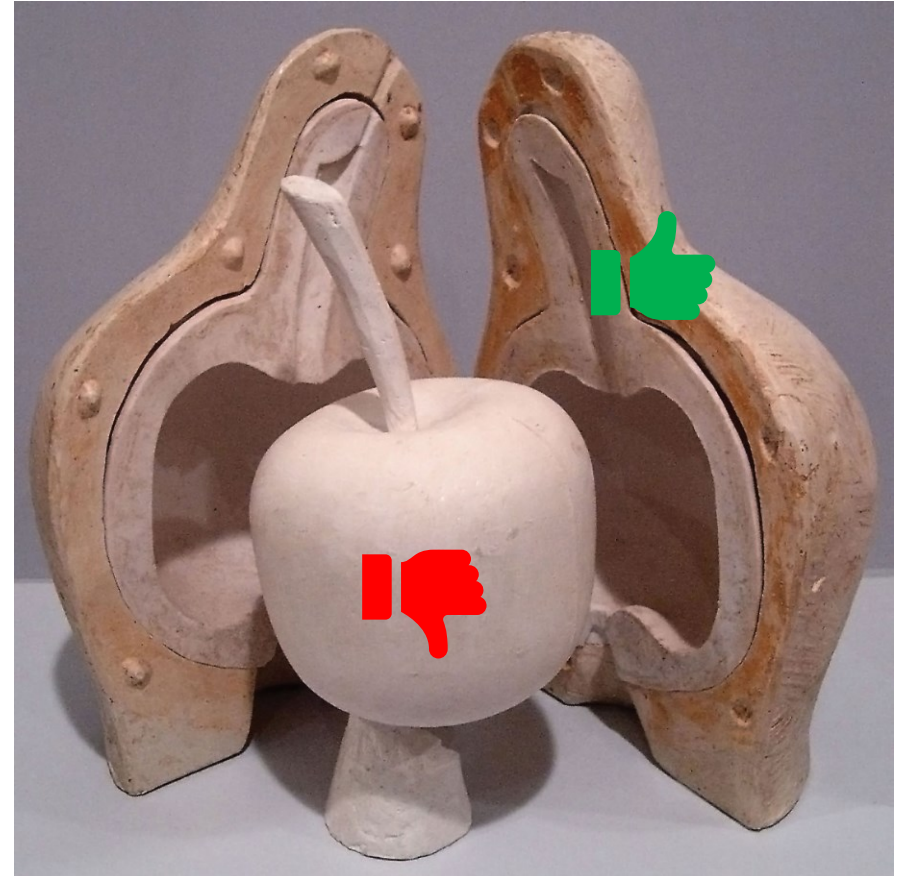
Classe

Objetos



Como programar objetos?

- Embora a programação seja orientada a objetos...
- Nós programamos as classes.
- Uma vez que a classe foi definida, podemos criar inúmeros objetos a partir dela
- Trabalhamos "esculpindo" o molde para criar objetos
- Todo objeto é uma instância de uma classe.



Exercício

- Pense em um consultório médico.
- Quais são os objetos que podem ser modelados nesse cenário?
- Vamos pensar apenas no paciente.
 - Quais são os dados ou características que precisamos representar?
 - Quais ações (ou funcionalidades) precisamos implementar para o paciente?

Paciente
Nome
Endereço
Data de Nascimento
Telefone
Plano de Saúde
Agendar Consulta
Alterar Consulta
Cancelar Consulta
Pagar Consulta

Programação Estruturada

vs

Programação Orientada a Objetos

- Definimos um struct para reunir os dados
- Criamos métodos para lidar com o struct

- Por exemplo:

```
typedef struct {  
    char cpf[12];  
    char nome[255];  
    int idade;  
    float valorHora;  
} pessoa;  
float CalcularSalario  
(pessoa p, int  
horasTrabalhadas);
```

- Reunimos os atributos (os dados) e os métodos usados para lidar com uma pessoa.

Pessoa
CPF
Nome
Idade
ValorHora
CalcularSalario

Dicas sobre Orientação a Objetos

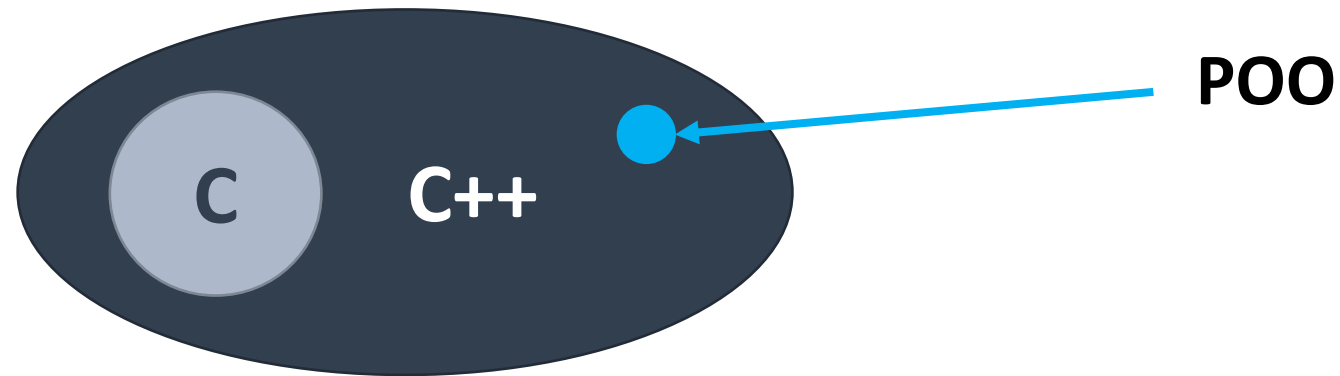
- Como identificar o que deve ser modelado como um objeto
 - Tudo pode ser visto e modelado como um objeto: Entidades concretas (pessoas, animais, carros) ou abstratas podem ser interpretados como objetos
- Como identificar os atributos de um objeto
 - Todos os valores, características e detalhes das entidades são modelados como atributos do objeto. No caso de uma pessoa: nome, idade, CPF, etc...
- Como identificar os métodos de um objeto
 - As ações que a entidade por executar (ou participar) podem ser vistos como métodos: Andar, ProcurarEmprego, ReceberSalario, etc...

Exercícios (para a próxima aula):

1. Identifique os atributos e métodos necessários para implementar um sistemas para controle de funcionários. Cada funcionário deve possuir CPF, Nome, Data de Nascimento, Função e Salário. O sistema deve prever a exclusão de funcionários, a emissão da folha de pagamento e uma opção que permita o reajuste do salário do funcionário.
2. Escreva um programa para controlar a agenda de um consultório médico. Observe os slides anteriores para visualizar os atributos e métodos dos pacientes. Seu programa deve controlar também os pagamentos recebidos pelo médico.

C++

- O C++ é uma evolução do C
- C é uma linguagem estruturada
- C++ é uma linguagem estruturada, que implementa POO



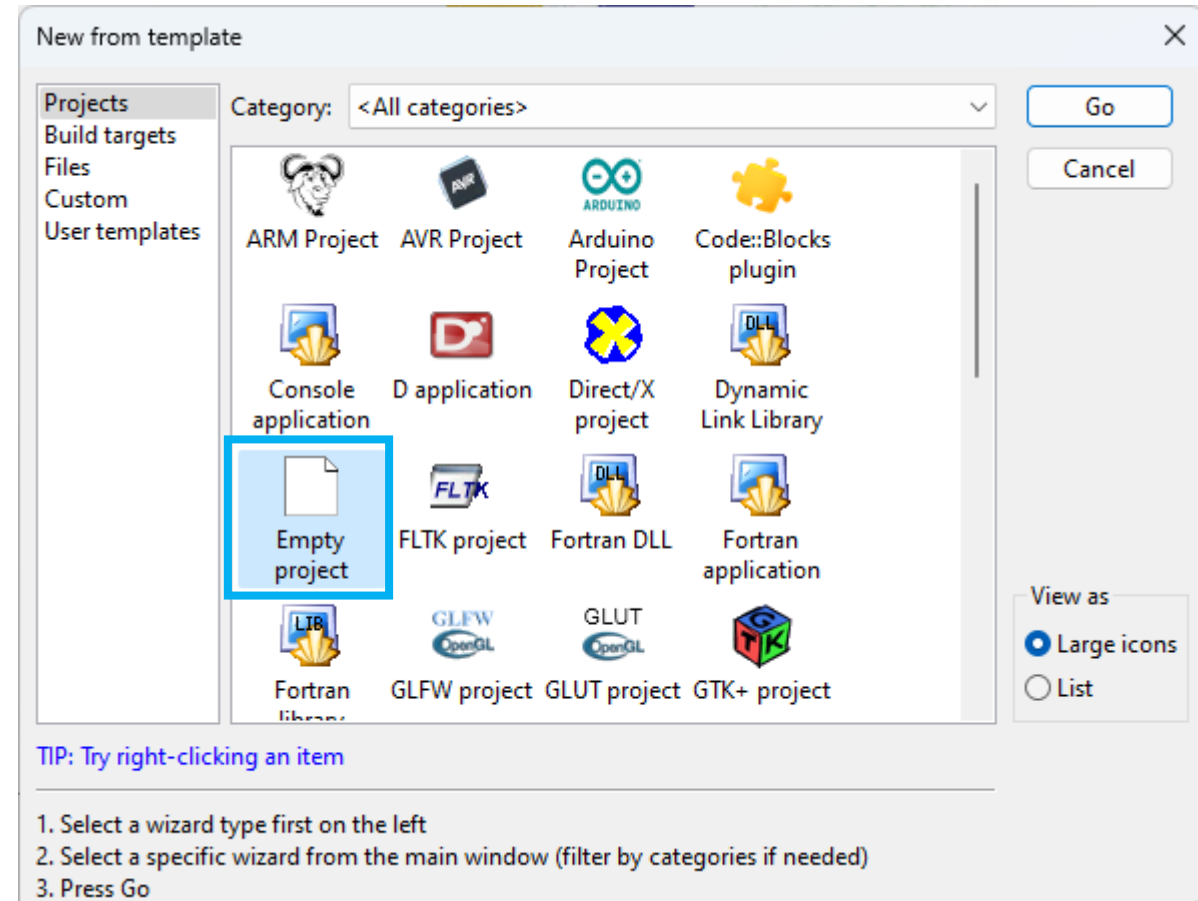
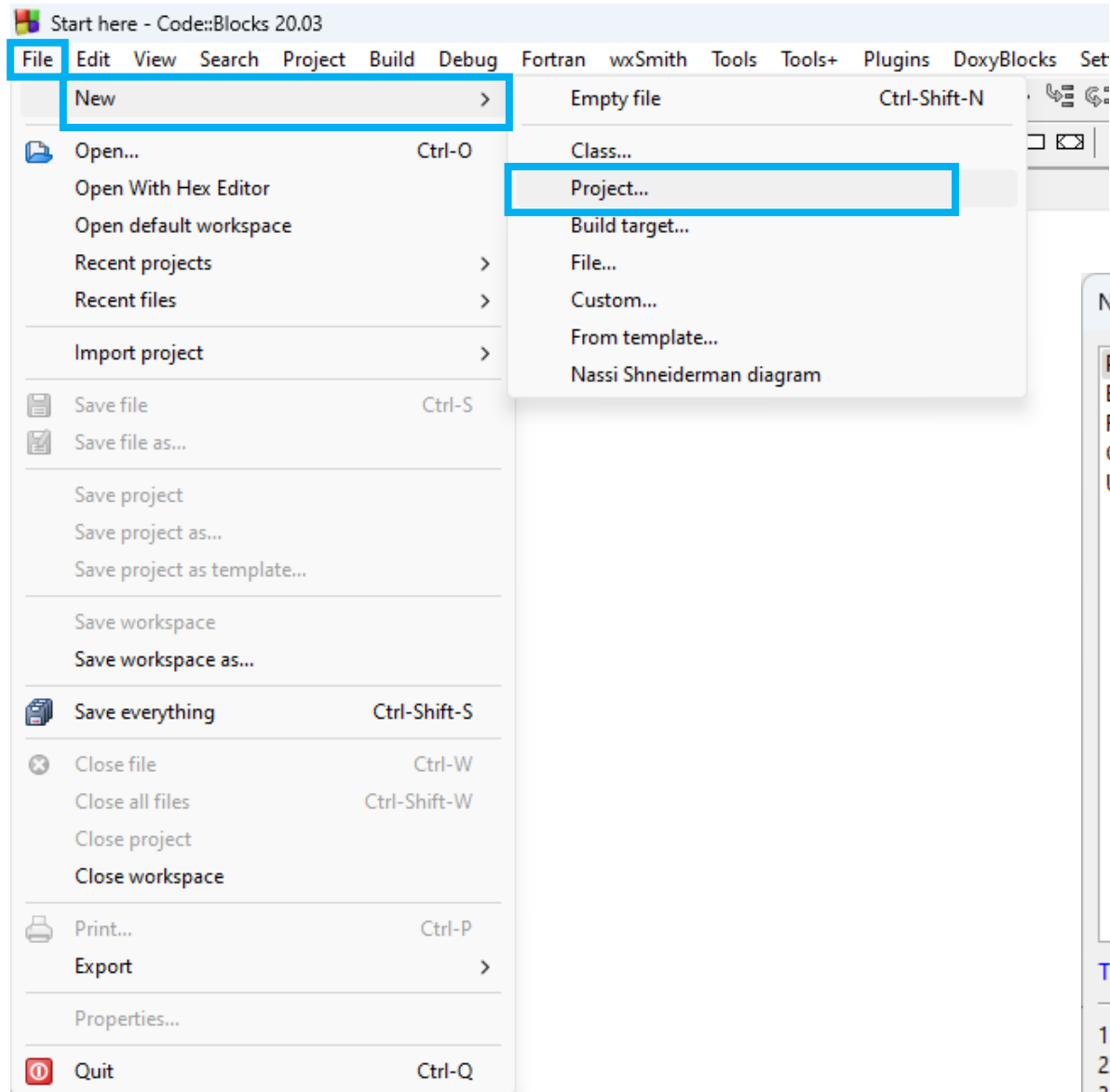
Como definir classes em C++?

- Antes de definir as classes, precisamos entender onde e como definir as classes.
- Até agora todos os nossos programas eram escritos em um único arquivo .c
- Entretanto, para definir novas classes, precisamos definir novos arquivos.
 - .h arquivo de cabeçalhos (headers)
 - .cpp arquivo de implementações (cpp = c plus plus = c++)
- O arquivo .h apresenta a estrutura
- O arquivo .cpp contém a implementação dos métodos (como eles devem funcionar) .

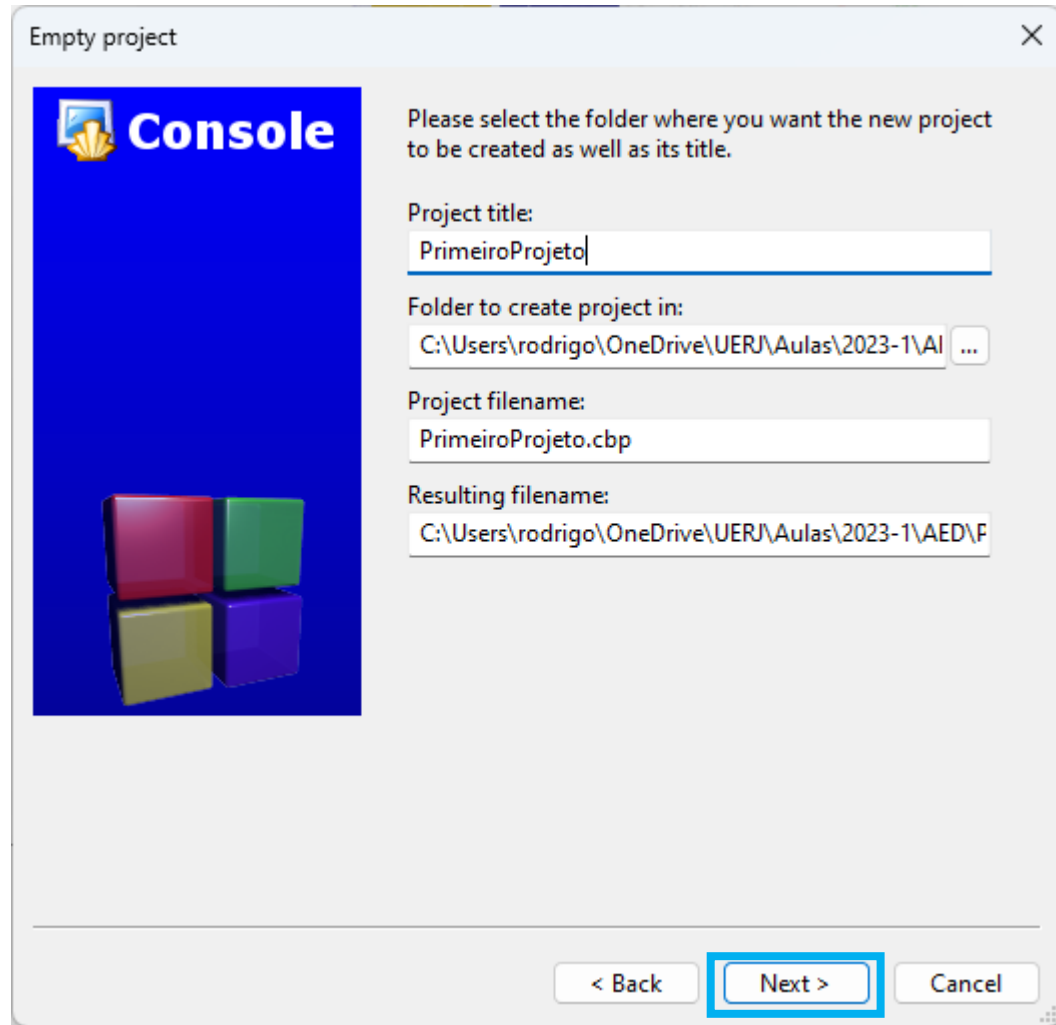
Como lidar com esses arquivos

- Existem duas formas de lidar com esses vários arquivos
 - Definir um projeto (vale para qualquer IDE)
 - Controlar a compilação manualmente ou usando um makefile
- Vamos usar a primeira opção
- Futuramente, vamos abordar as outras opções.

CodeBlocks



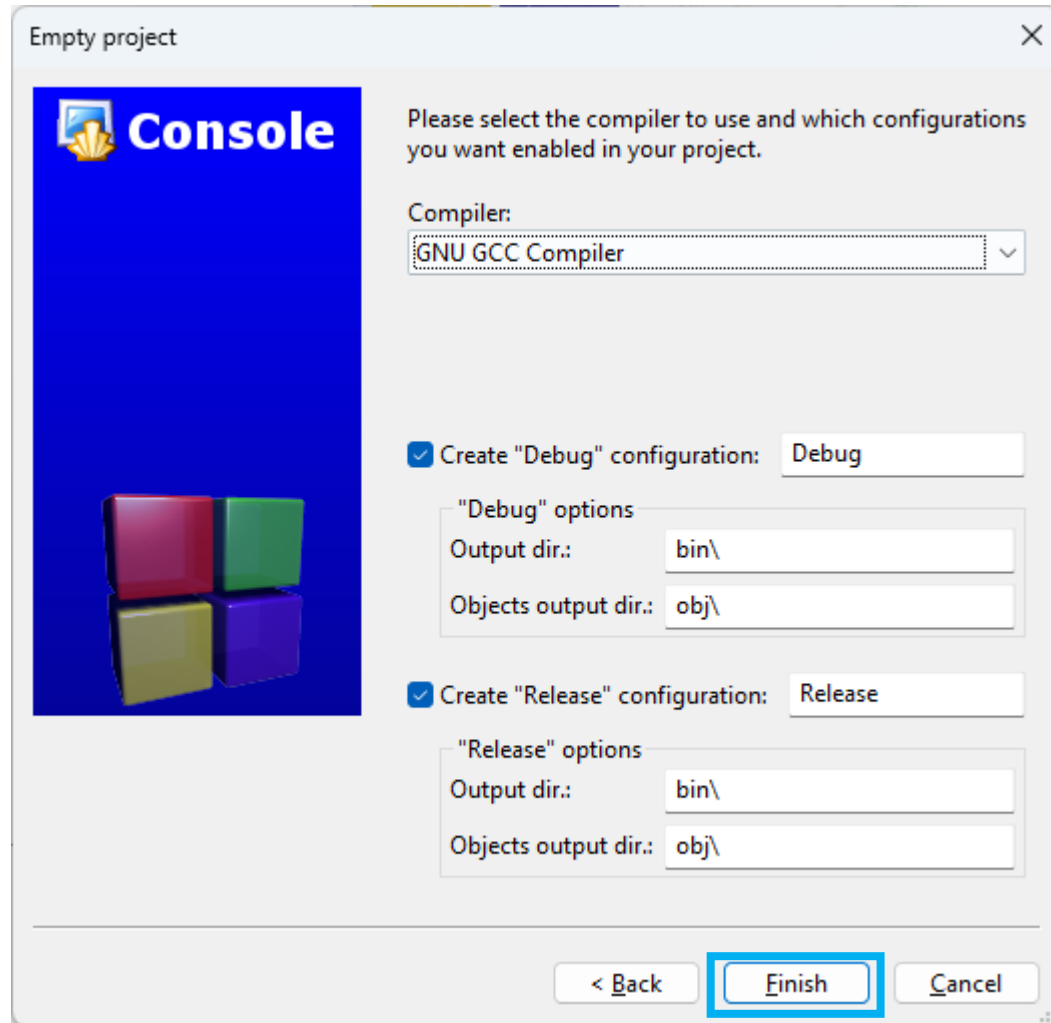
CodeBlocks



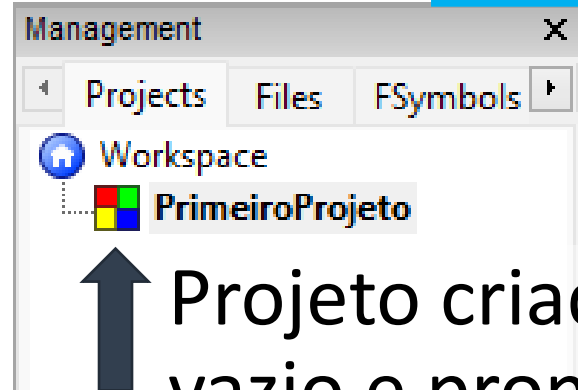
← Nome do projeto

← Pasta onde vamos salvar o projeto

CodeBlocks



- Escolha o compilador (em geral, a opção apresentada está correta).
- Mantenha as caixas debug e release clicadas.
- Versão debug: o compilador recebe instruções de que o programa ainda está em desenvolvimento e adiciona mais informações de erros e avisos
- Versão release: versão final, com otimizações para acelerar o código e reduzir o consumo de espaço (memória principal e o tamanho do executável)

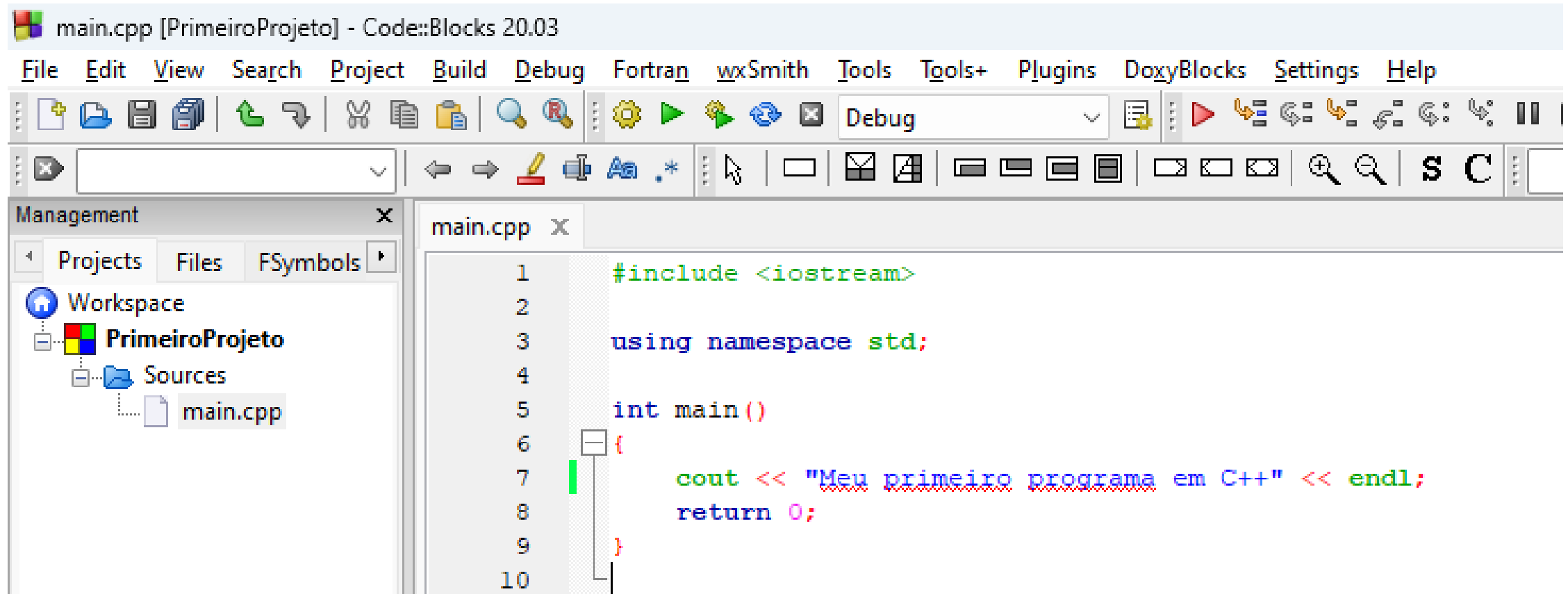


Codeblocks

- Vamos adicionar um arquivo ao projeto.
- Esse arquivo conterá a função main do projeto
- Importante:
 - Código em C: arquivo com extensão .c
 - Código em C++: arquivo com extensão cpp
 - Não misturar os dois.
- Para adicionar o novo arquivo, siga o processo normal de criar um arquivo em branco.
- Mas não se esqueça: salve como extensão .cpp

Codeblocks

- No exemplo, criei o arquivo main.cpp



The screenshot displays the Code::Blocks IDE interface. The title bar reads "main.cpp [PrimeiroProjeto] - Code::Blocks 20.03". The menu bar includes File, Edit, View, Search, Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DoxyBlocks, Settings, and Help. The toolbar contains icons for file operations, editing, and debugging. The left sidebar shows the "Management" pane with tabs for Projects, Files, and FSymbols. Under the "Projects" tab, a tree view shows the workspace structure: "Workspace" containing "PrimeiroProjeto", which in turn contains a "Sources" folder with the file "main.cpp". The main editor window displays the code in "main.cpp" with line numbers 1 through 10. The code is as follows:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      cout << "Meu primeiro programa em C++" << endl;
8      return 0;
9  }
10
```

Comandos de entrada e saída em C++

- Os comandos de entrada e saída em C++ estão na biblioteca `iostream`. Logo: `#include <iostream>`
- As bibliotecas nativas do C++ “agrupam” todas as funções dentro de um namespace (como um domínio separado) chamado `std` (de Standard ou padrão).
- Para usar esse namespace, precisamos declarar essa necessidade: `using namespace std;`

Comando de saída em C++

- O comando de saída em C++ é o `cout`.
- Para direcionar algum texto ou conteúdo para essa saída precisamos usar o operador `<<`
- `cout << "Primeiro comando de saída em C++\n";`
- Podemos substituir o `\n` nas saídas por `<< endl;`
- `cout << "Teste" << endl;`

Comando de saída em C++

- Para imprimir o conteúdo de uma variável, basta usar o comando de saída diretamente:

```
int a = 5;  
cout << a << endl;
```

- Ou podemos mesclar texto com variáveis:

```
cout << "O valor de a é " << a << endl;
```

Comando de entrada em C++

- O comando de entrada padrão em C++ é o `cin`

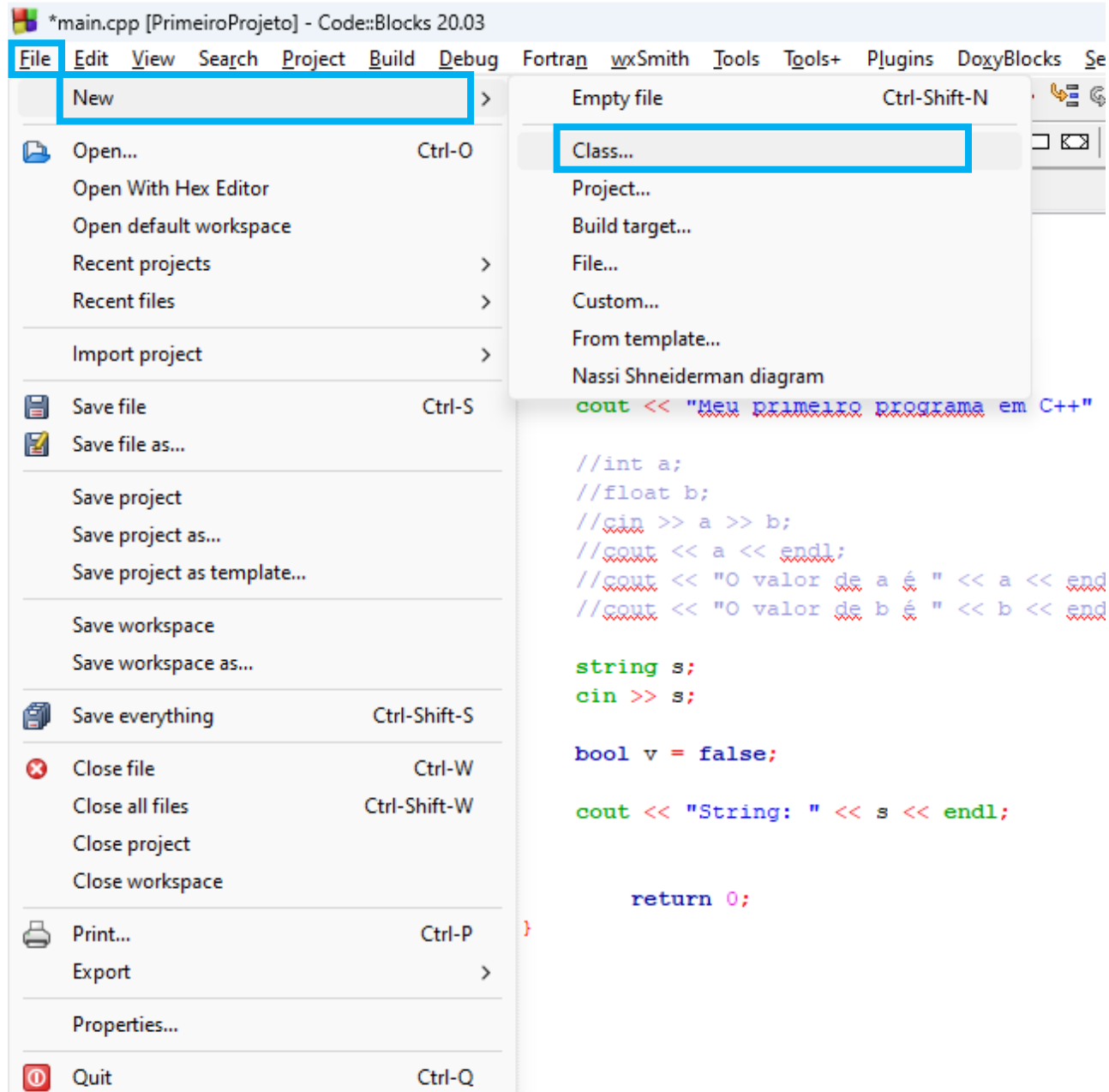
```
int a;  
cin >> a;  
cout << "O valor de a é " << a << endl;
```

```
int a;  
float b;  
cin >> a >> b;  
cout << "O valor de a é " << a << endl;  
cout << "O valor de b é " << b << endl;
```

Comandos C em C++

- Seguem idênticos em C++:
 - Operadores aritméticos, lógicos e relacionais
 - Estruturas de decisão
 - Estruturas de repetição
 - Definição de funções e procedimentos
 - Passagem de parâmetros por valor e por referência
 - Vetores e Matrizes
 - Structs
 - Ponteiros
- Apresentam diferenças:
 - Strings (possui um tipo próprio – string)
 - Tipo de dados booleano nativo (bool x = True)

Adicionar nova classe



Adicionar nova classe

Create new class

Class definition

Class name:

Arguments:

☐ Has destructor ☐ Has copy ctor
☒ Virtual destructor ☐ Has assignment op.

Inheritance

☐ Inherits another class

Ancestor:

Ancestor's include filename:

Scope:

Member variables

Add new:

Scope:

☐ Add "Getter" method
☐ Add "Setter" method
☐ Remove prefix:

Remove Add

Documentation

☐ Add documentation where appropriate

File policy

☒ Add paths to project ☒ Use relative path

☐ Header and implementation file shall be in same folder

Folder:

☐ Header and implementation file shall always be lower case

Header file

Folder:

Filename:

☒ Add guard block in header file

Guard block:

Implementation file

☒ Generate implementation file

Folder:

Filename:

Header include:

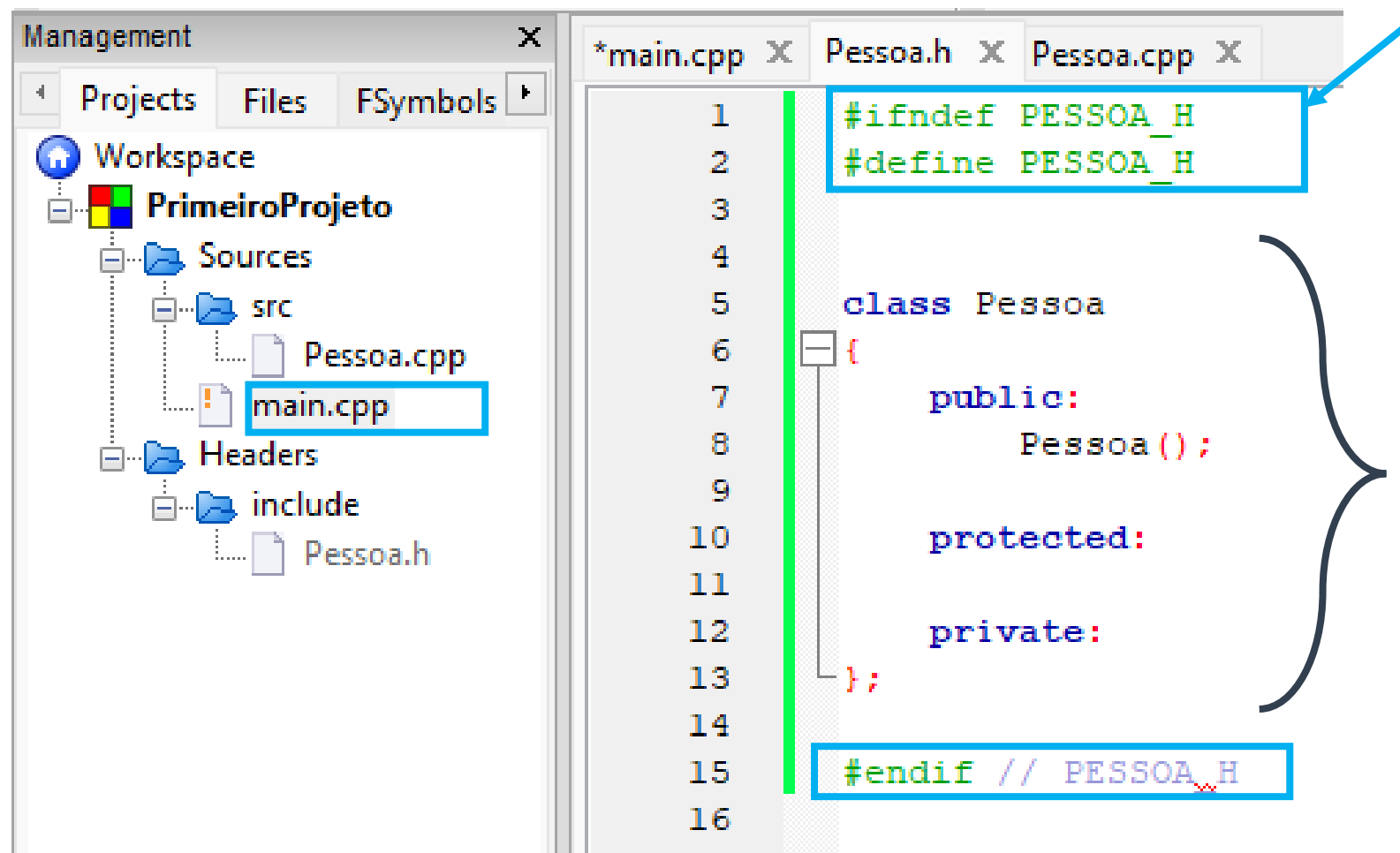
Create Cancel

Nome da classe

Informações do arquivo de cabeçalho (.h)
Informações do arquivo de implementação (.cpp)

Informações dos **Include Guards**

Arquivo Pessoa.h



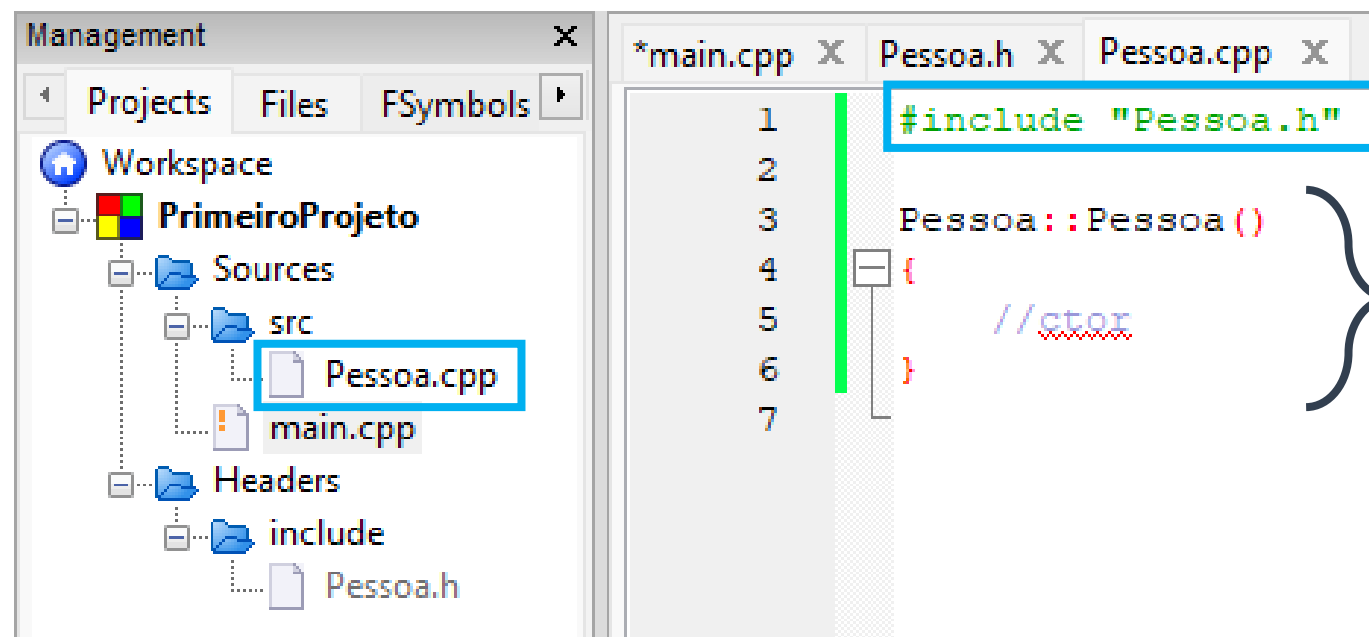
Include Guard

Definição de Classe:

- Atributos
- Métodos
- ...

Arquivo Pessoa.cpp

Importante: O arquivo .cpp precisa incluir o arquivo .h contendo a definição da classe.



Implementação dos métodos

- Construtores
- Destrutores
- Métodos
- Métodos estáticos
-

Include Guards (ou header guard)

- Cada classe deve ser descrita em um arquivo separado
 - Arquivo .h para definição – descrição dos atributos e métodos
 - Arquivo .cpp para o código – como cada método deve funcionar
- O problema:
 - Ao definir várias classes (arquivos), surge um problema:
 - Classe 1 usa a Classe 2 (requer o arquivo 2)
 - Classe 2 usa a Classe 3 (requer o arquivo 3)
 - Classe 3 usa a Classe 1 (requer o arquivo 1)
 - Isso cria um ciclo de importações infinito ($1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow \dots$)
- A solução: Include Guards
 - Comandos que evitam essa repetição: cada arquivo só é importado uma única vez. Na próxima, ele é ignorado (já foi importado antes).
- Usado somente nos arquivos de cabeçalho (extensão .h)

Include Guards (ou header guard)

`#ifndef <nome do arquivo>`

← Se esse arquivo não foi definido antes

`#define <nome do arquivo>`

← Defina esse arquivo

`class <Nome_Da_Classe>`

`{`

`public:`

`protected:`

`private:`

`};`

`#endif`

← Fecha o SE do início

Conceitos POO

- Existem quatro pilares básicos da Programação Orientada a Objetos:
 - Abstração
 - Encapsulamento
 - Herança
 - Polimorfismo
- Nas próximas aulas vamos ver cada item detalhadamente e como definir nossas classes

