Programação Orientada a Objetos

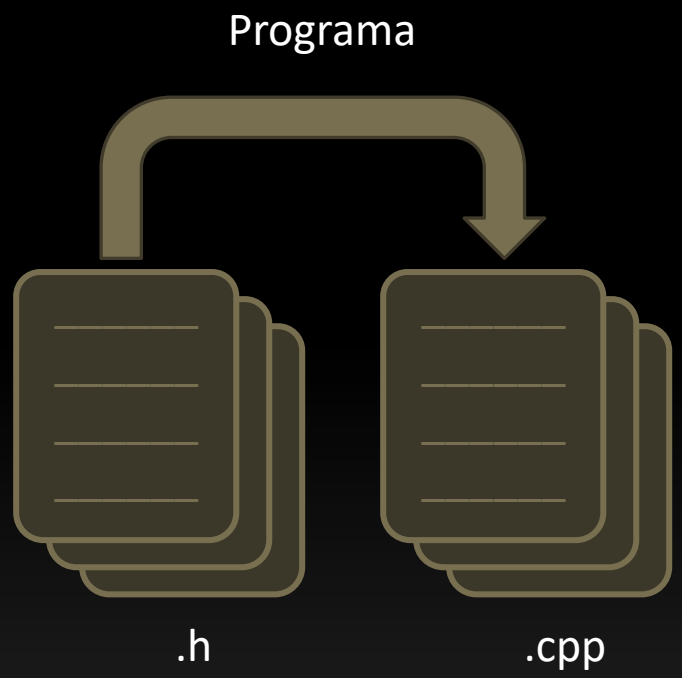
ORGANIZAÇÃO DE CÓDIGO

em C++

Introdução

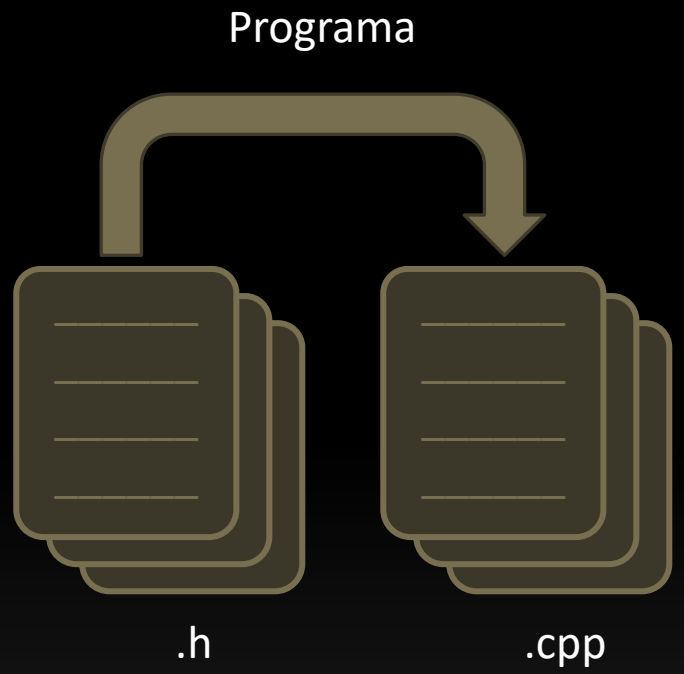
- Programas em C++ são compostos por vários arquivos
  - Arquivos de inclusão (.h)
  - Código fonte (.cpp)

Esses arquivos frequentemente compartilham dados, funções, tipos, etc.



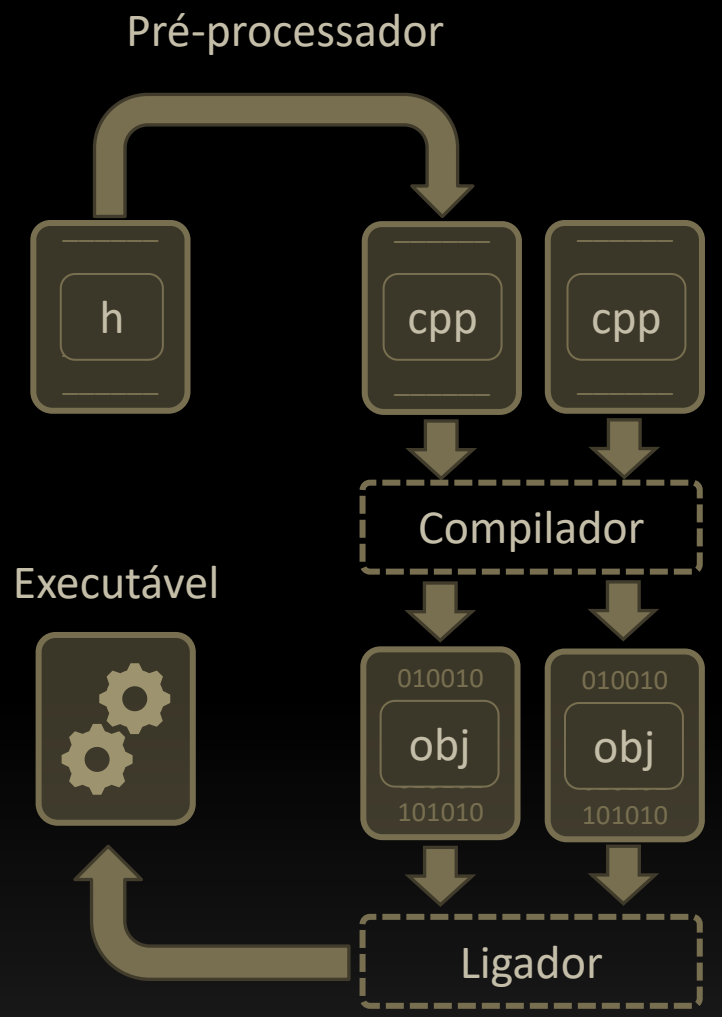
Introdução

- Como organizar o código?
  - Onde vai cada coisa?
    - Arquivos de inclusão
    - Arquivos de código fonte
  - Quem pode incluir o que?
  - Quais os erros mais comuns?
  - Como evitá-los?



Compilação e Ligação

- Arquivos de código fonte (.cpp)
  - Compilados de forma independente
    gerando arquivos objeto (.obj)
  - Ligados ao final do processo
    gerando o executável (.exe)
- Arquivos de inclusão (.h)
  - Adicionados ao código fonte
    pelo pré-processador



Compilação e Ligação

- Os arquivos cpp's devem ser autocontidos
  - Não depender de outros cpp's para a compilação

Jogo.h

```
class Jogo
{
private:
   string nome;
   float preco;
   int horas;

   ...
};
```

Principal.cpp

```
#include "Jogo.h"

int main()
{
   Jogo gow
   {
     "God of War",
     100.0f,
     40
   };
}
```

A declaração do tipo "Jogo" deve estar no arquivo cpp.

Compilação e Ligação

- Os arquivos .h não são compilados
  - Usados para compartilhar declarações

Jogo.h

```
class Jogo
{
private:
   string nome;
   float preco;
   int horas;

public:
   void Exibir();
   ...
};
```

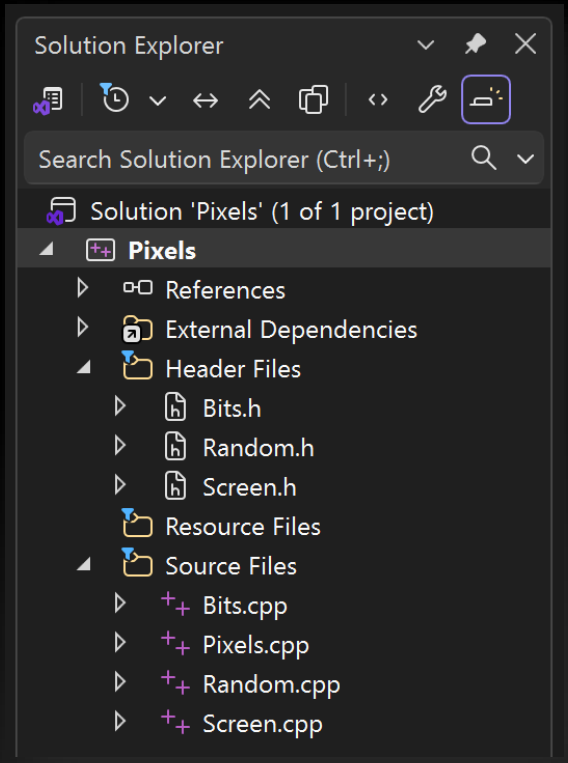
Jogo.cpp

```
#include "Jogo.h"

void Jogo::Exibir()
{

    cout << nome  << " R$"
         << preco << " "
         << horas << "h\n";

}
...
```

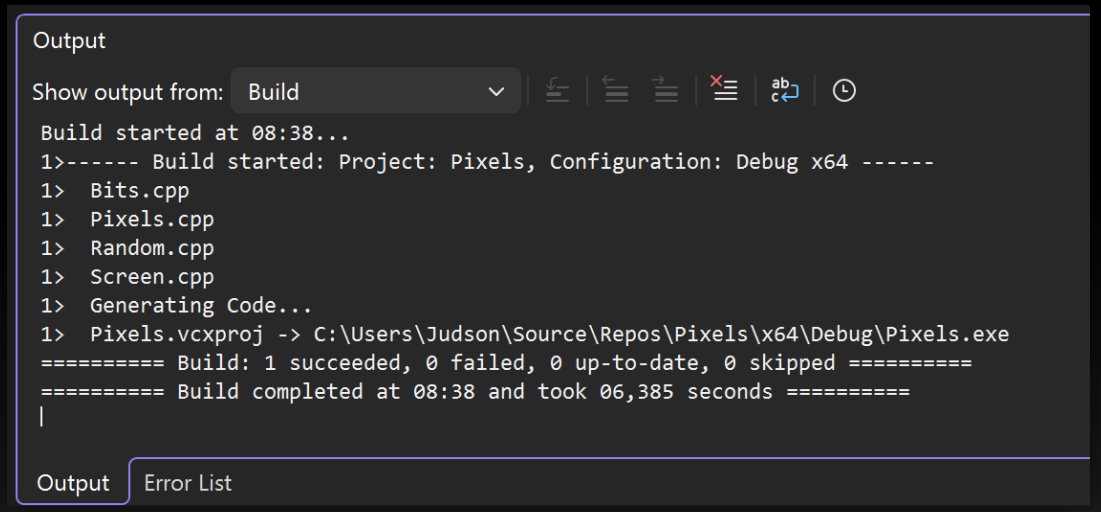
Outro cpp's também precisam da declaração do tipo "Jogo".

Compilação e Ligação

- Arquivos de inclusão não são compilados

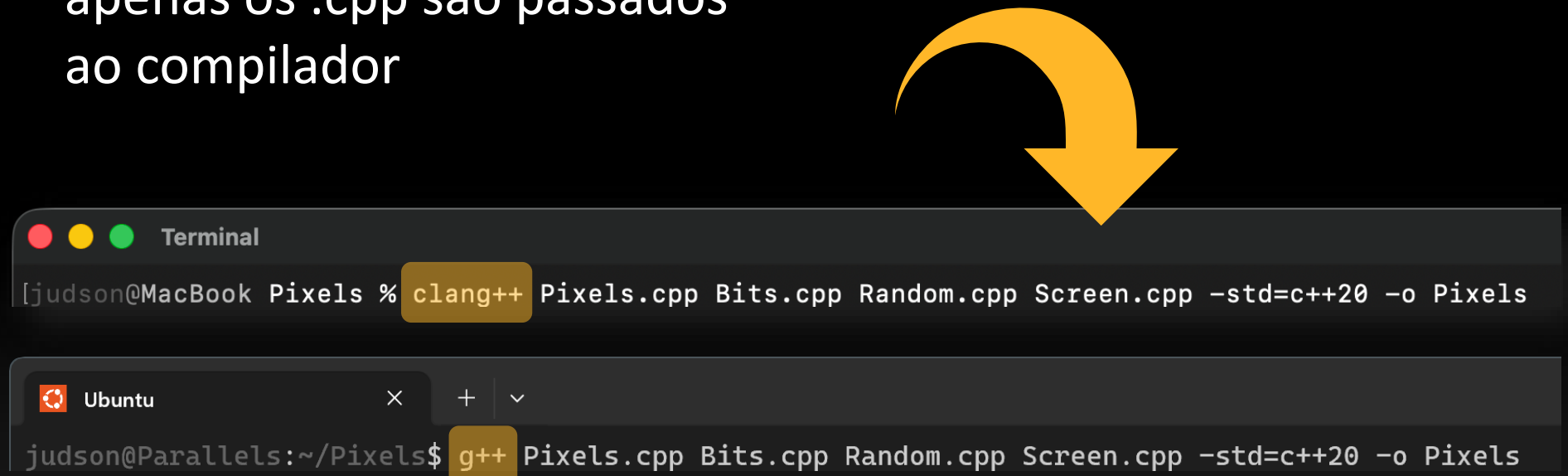


O pré-processador inclui o conteúdo do .h no .cpp ao achar um #include



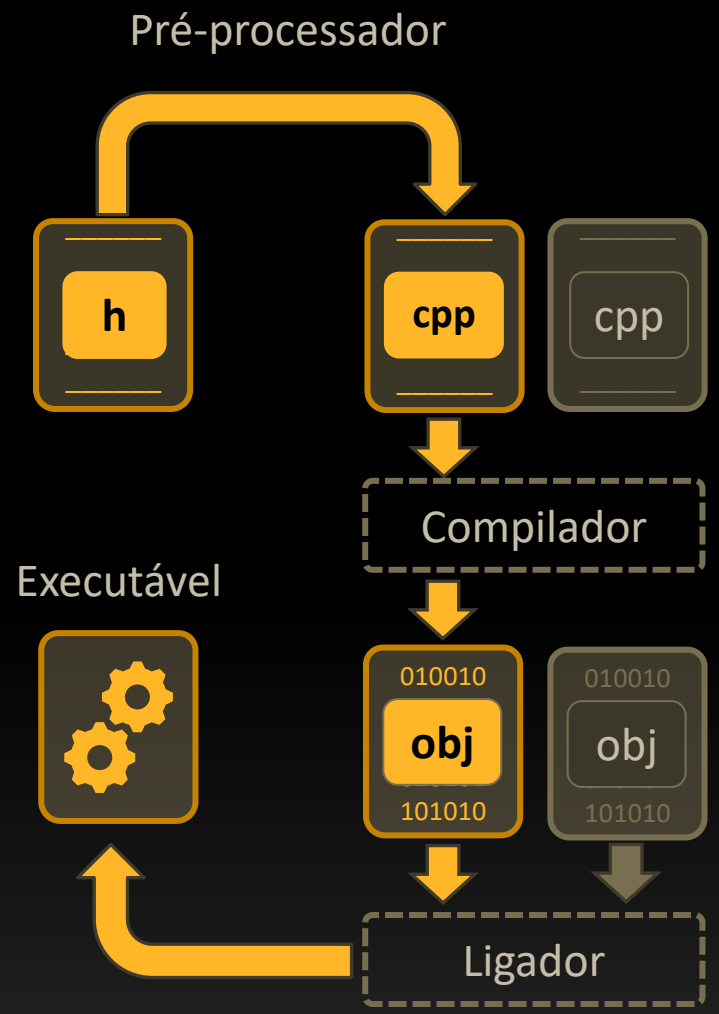
Compilação e Ligação

- Arquivos de inclusão não são compilados
  - Na linha de comandos,
    apenas os .cpp são passados
    ao compilador



Compilação e Ligação

- **Recompilação seletiva**
  - Visual Studio ou CMake
  - Apenas arquivos modificados precisam ser recompilados:
    - Código fonte
      novos arquivos objeto
    - Arquivos de inclusão
      novos arquivos de código fonte
      novos arquivos objeto



Arquivos de Inclusão

- São usados para concentrar declarações
  - Evitam a duplicação manual

Jogo.h

```
class Jogo
{
private:

   string nome;
   float preco;
   int horas;

   ...

};
```

Principal.cpp

```
#include "Jogo.h"

int main()
{
   Jogo gow
   {
     "God of War",
     100.0f,
     40
   };
}
```

Jogo.cpp

```
#include "Jogo.h"

void Jogo::Exibir()
{
    cout << nome  << " R$"
         << preco << " "
         << horas << "h\n";
}

...
```

Arquivos de Inclusão

- Podem conter declarações de:
  - Constantes
  - Funções
  - Registros
  - Classes
  - Templates
- Não devem conter:
  - Variáveis

Arquivo .h

```
const float Pi = 3.14;

void Exibir(char ch);

struct Ponto
{
    int x;
    int y;
};

int val;
```

❌

```
class Jogo
{
private:

   string nome;
   float preco;
   int horas;
   ...
};

Jogo gears;
```

❌

Arquivos de Inclusão

- Isso se deve a *One Definition Rule* (ODR)
  - As variáveis, funções, objetos, classes, etc. só podem ser definidas uma vez
    - Declaração é diferente de definição
    - Mas a declaração de uma variável é uma definição

Jogo.h

```
// variável
int val;
```

Principal.cpp

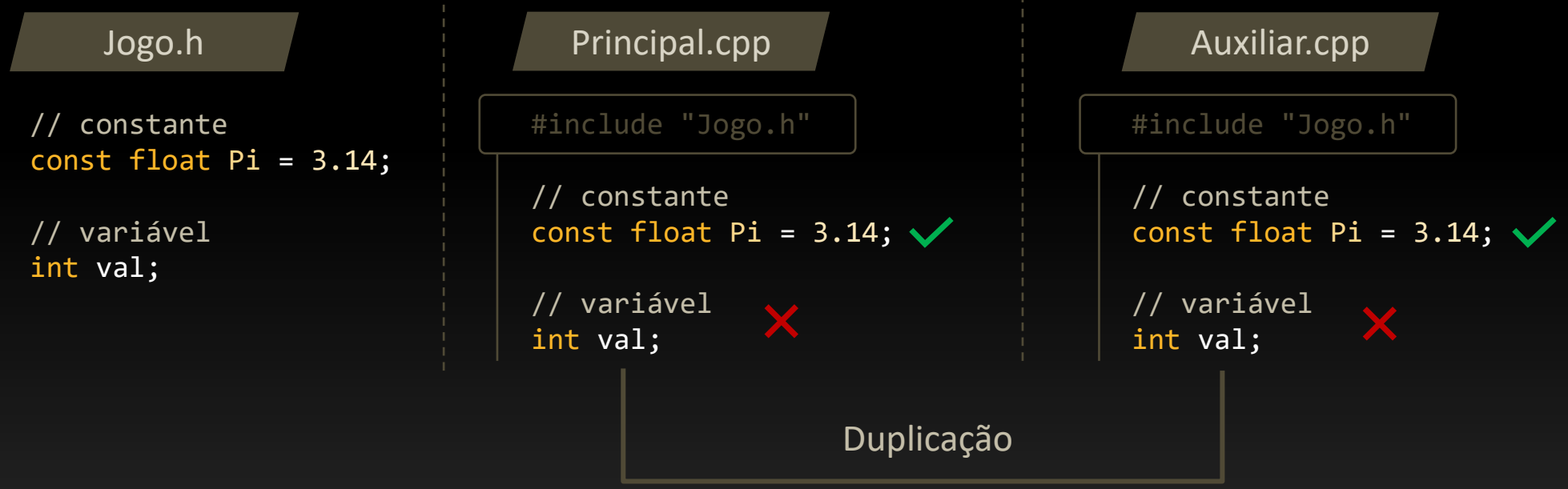
```
#include "Jogo.h"
int val;
```

Auxiliar.cpp

```
#include "Jogo.h"
int val;
```

Arquivos de Inclusão

- Declarações de constantes e variáveis
  - Constantes tem ligação interna
  - Variáveis tem ligação externa



Arquivos de Inclusão

- Declarações de registros e classes
  - Definem apenas um modelo
  - Não criam variáveis

Jogo.h

```
class Jogo
{
private:
   string nome;
   float preco;
   int horas;
   ...
};
```

Principal.cpp

```
#include "Jogo.h"

class Jogo
{
   ...
};

// variável
Jogo gears;
```

Auxiliar.cpp

```
#include "Jogo.h"

class Jogo
{
   ...
};

// variável
Jogo gow;
```

Arquivos de Inclusão

- Declarações de funções
  - O protótipo não cria funções

Jogo.h

```
// funções
void Exibir(char ch);
```

Principal.cpp

```
#include "Jogo.h"

void Exibir(char ch);


int main()
{
    Exibir('A');
}
```

Auxiliar.cpp

```
#include "Jogo.h"

void Exibir(char ch);


void Exibir(char ch)
{
    ...
}
```

Arquivos de Inclusão

- Declarações de templates de funções
  - O compilador cria a função a partir do template

Jogo.h

```
// templates
template <typename T>
void Exibir(T val)
{
   ...
}
```

Principal.cpp

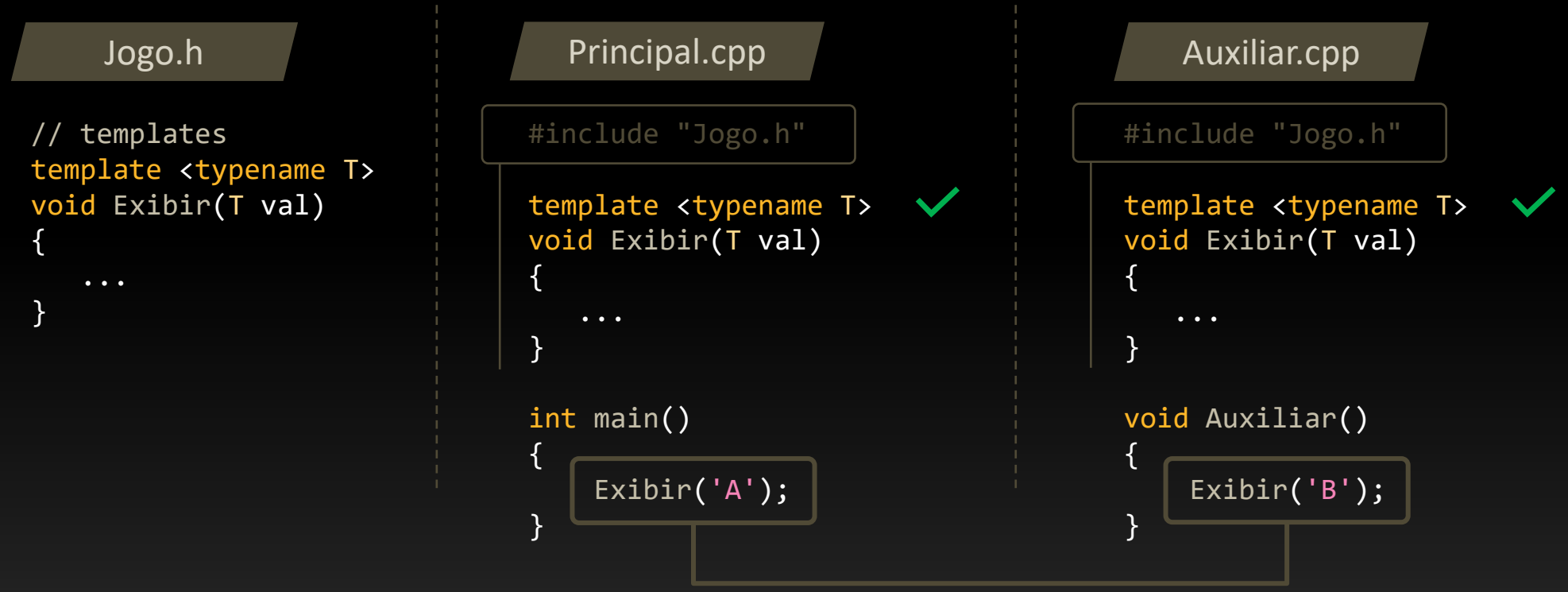
```
#include "Jogo.h"

template <typename T>
void Exibir(T val)
{
   ...
}

int main()
{
    Exibir('A');
}
```

Auxiliar.cpp

```
#include "Jogo.h"

template <typename T>
void Exibir(T val)
{
   ...
}

void Auxiliar()
{
    Exibir(50);
}
```

Arquivos de Inclusão

- Templates podem gerar funções duplicadas
  - O ligador identifica a duplicação e usa apenas uma



Arquivos de Inclusão

- Funções não podem ser duplicadas
  - Duplicação é tratada apenas em templates

Principal.cpp

```
void Exibir(char val)
{
   ...
}

int main()
{
    Exibir('A');
}
```

Auxiliar.cpp

```
void Exibir(char val)
{
   ...
}

void Auxiliar()
{
    Exibir('B');
}
```

A compilação funcionaria, mas teríamos um erro de ligação.

Erros Comuns

- Arquivos de código fonte (.cpp)
  - Devem conter:
    - Definição (implementação) de funções e métodos
    - Criação de variáveis
  - Não devem ser incluídos
    - Cria um único .cpp
    - Quebra a recompilação seletiva: compilação mais lenta

Principal.cpp

```
#include "Auxiliar.cpp"

int main()
{
    ...
}
```

Erros Comuns

- Inconsistência entre declaração e definição
  - Erro de ligação

Mensagem.h

```
class Msg
{
private:
   string texto;

public:
   Msg();

   void Exibir();
   void Exibir(int n);
};
```

Mensagem.cpp

```
#include <iostream>
#include "Mensagem.h"

Msg::Msg()
{
    ...
}

void Msg::Exibir()
{
    ...
}
```

Programa.cpp

```
#include "Mensagem.h"

int main()
{
   Msg msg;

   msg.Exibir();
   msg.Exibir(2);
}
```

Erros Comuns

- Criar definição no .h
  - Erro de ligação: definições múltiplas

Mensagem.h

```
void Exibir(int num)
{
   cout << num << '\n';
}
```

Principal.cpp

```
#include "Mensagem.h"
```

Auxiliar.cpp

```
#include "Mensagem.h"
```

Code	Description
LNK2005	"void __cdecl Exibir(int)" (?Exibir@@YAXH@Z) already defined in Auxiliar.obj
LNK1169	one or more multiply defined symbols found

Erros Comuns

- Inclusões duplicadas
  - Declarações repetidas no mesmo arquivo .cpp

Palavra.h

```
class Palavra
{
private:
   char palavra[20];

public:
   Palavra();
   ~Palavra();
};
```

Texto.h

```
#include "Palavra.h"

class Texto
{
private:
   Palavra texto[10];

public:
   Texto();
   ~Texto();
};
```

Programa.cpp

```
#include "Palavra.h"
#include "Texto.h"

int main()
{
   Palavra palavra;
   Texto texto;

   ...
}
```

Erros Comuns

- É preciso proteger os arquivos de inclusão
  - Usando *header guards*

Palavra.h

```
#ifndef PALAVRA_H
#define PALAVRA_H

class Palavra
{
   ...
};

#endif
```

Texto.h

```
#ifndef TEXTO_H
#define TEXTO_H

#include "Palavra.h"

class Texto
{
   ...
};

#endif
```

Programa.cpp

```
#include "Palavra.h"
#include "Texto.h"

int main()
{
   Palavra palavra;
   Texto texto;

   ...
}
```

Erros Comuns

- É preciso proteger os arquivos de inclusão
  - #pragma once não é padronizado

Palavra.h

```
#pragma once

class Palavra
{
   ...
};
```

Texto.h

```
#pragma once

#include "Palavra.h"

class Texto
{
   ...
};
```

Programa.cpp

```
#include "Palavra.h"
#include "Texto.h"

int main()
{
   Palavra palavra;
   Texto texto;

   ...
}
```

Erros Comuns

- Inclusões desnecessárias
  - Prefira sempre inclusões no .cpp

Audio.h

```
#ifndef AUDIO_H
#define AUDIO_H

#include <iostream>
#include "Sound.h"

class Audio
{

};

#endif
```

Audio.cpp

```
#include <iostream>
#include "Sound.h"
#include "Audio.h"

Audio::Audio()
{
    Sound snd;
}

void Audio::Erro()
{
    cout << "Falha no áudio\n";
}
```

Erros Comuns

- Inclusões circulares são um problema
  - Mesmo com *header guards*

Sound.h

```
#ifndef SOUND_H
#define SOUND_H

#include "Audio.h"

class Sound
{
    Audio * aud;
};

#endif
```

Audio.h

```
#ifndef AUDIO_H
#define AUDIO_H

#include "Sound.h"

class Audio
{
    Sound * snd;
};

#endif
```

```
#ifndef SOUND_H
#define SOUND_H

#include "Audio.h"

#ifndef AUDIO_H
#define AUDIO_H

#include "Sound.h"

class Audio
{
   Sound  * snd;
};

#endif

class Sound
{
   Audio  * aud;
};

#endif
```

Erros Comuns

- A solução para inclusões circulares:
  - Usar forward declarations

Sound.h

```
#ifndef SOUND_H
#define SOUND_H

#include "Audio.h"

class Audio;

class Sound
{
    Audio * aud;
};

#endif
```

Audio.h

```
#ifndef AUDIO_H
#define AUDIO_H

#include "Sound.h"

class Sound;

class Audio
{
    Sound * snd;
};

#endif
```

```
#ifndef SOUND_H
#define SOUND_H

#include "Audio.h"

#ifndef AUDIO_H
#define AUDIO_H

#include "Sound.h"

class Sound;

class Audio
{
   Sound  * snd;
};

#endif

class Audio;

class Sound
{
   Audio  * aud;
};

#endif
```

Resumo

- Declare no .h
- Defina no .cpp
- Utilize:
  - *Pragma once* (`#pragma`)
  - *Header guards* (`#ifndef #define #endif`)
  - *Forward declaraction* (inclusões circulares)
- Inclua arquivos .h, nunca .cpp
- Prefira inclusões no .cpp