Linguagem de Programação I

Aula 20 - Modularidade: criação e uso de bibliotecas

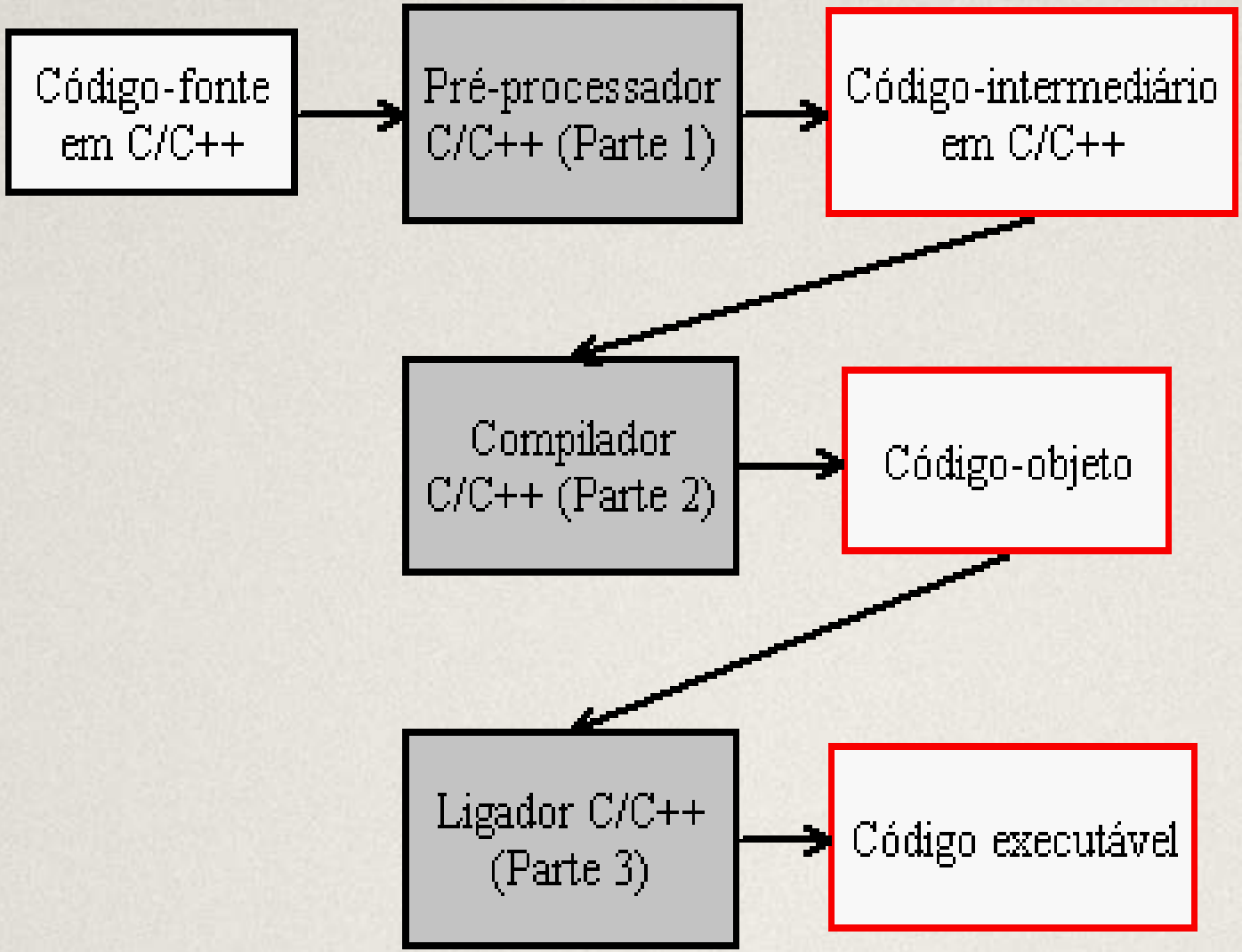
Objetivos da aula

- Desenvolver e manipular bibliotecas estáticas e dinâmicas em C++
- Para isto estudaremos:
  - Criação de bibliotecas
  - Uso de bibliotecas em programas
- Ao final da aula espera-se que o aluno saiba tirar proveito das vantagens do uso de bibliotecas em seus programas

Recapitulativo da compilação em C++

- A compilação de um programa em C++ envolve 3 passos principais:
  - Pré-processamento
    - Executa as diretivas #ifdef, #include, substitui macros, etc.
    - Gera arquivos temporários com código pré-processado
  - Compilação
    - Processa os arquivos temporários com código fonte pré-processado seguindo a gramática da linguagem
    - Gera arquivos objeto (.obj / .o) caso não ocorram erros
  - Linkedição
    - Faz a ligação de todos os arquivos de código-objeto gerados
    - Gera executável (.exe / .out) ou biblioteca dinâmica (.dll / .so)

Recapitulativo da compilação em C++



Bibliotecas

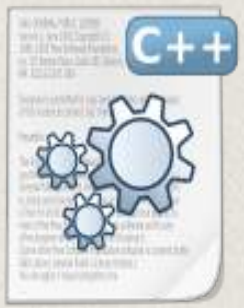
- As bibliotecas são classificadas da seguinte maneira:
  - Bibliotecas Estáticas (união)
    - Código objeto que deve ser inserido no programa
    - Contém os seguintes tipos de arquivos:
      - Protótipos (ou cabeçalhos .h)
      - Métodos e/ou sub-rotinas pré-compiladas (.lib ou .a)
  - Bibliotecas dinâmicas (compartilhamento)
    - Código executável que é separado do programa
    - Contém os seguintes tipos de arquivos:
      - Protótipos (ou cabeçalhos .h)
      - Referências para os métodos e/ou sub-rotinas (.lib ou .a)
      - Executáveis (.dll ou .so)

Bibliotecas estáticas

- Para serem geradas, as bibliotecas estáticas necessitam das seguintes etapas:
  - Pré-processamento
  - Compilação
- Arquivos de cabeçalho e de corpo são utilizados da mesma forma que em um programa comum
- OBS: Biblioteca estática não contém o método main

Criando uma biblioteca estática

- Exemplo:
  - Biblioteca estática que efetua cálculos geométricos
- Passos:
  - Criar um projeto do tipo biblioteca estática
  - Criar dois arquivos de cabeçalho e inserí-los no projeto
    - Ponto.h
    - Circulo.h
  - Criar dois arquivos de corpo e inserí-los no projeto
    - Ponto.cpp
    - Circulo.cpp
  - Compilar o projeto para gerar o código objeto (.lib ou .a)



Arquivo de cabeçalho Ponto.h

```
1.  #ifndef _PONTO_H_
2.  #define _PONTO_H_
3.
4.  namespace geometria
5.  {
6.
7.      // Definição dos dados
8.      typedef struct
9.      {
10.         int x;
11.         int y;
12.     } Ponto;
13.
14.     // Definição das sub-rotinas
15.     int distancia( const Ponto& ponto1, const Ponto& ponto2 );
16.
17.     void imprimir( const Ponto& ponto );
18.
19. }
20.
21. #endif
```

Arquivo de corpo Ponto.cpp

```
1.  #include <iostream>
2.  #include <cmath>
3.
4.  #include "Ponto.h"
5.
6.  namespace geometria
7.  {
8.
9.      int distancia( const Ponto& ponto1, const Ponto& ponto2 )
10.     {
11.         return std::sqrt( std::pow( ponto1.x - ponto2.x, 2 ) +
12.                           std::pow( ponto1.y - ponto2.y, 2 ) );
13.     }
14.
15.     void imprimir( const Ponto& ponto )
16.     {
17.         std::cout << ponto.x << ", " << ponto.y << std::endl;
18.     }
19.
20. }
```

Arquivo de cabeçalho Circulo.h

```
1.  #ifndef _CIRCULO_H_
2.  #define _CIRCULO_H_
3.
4.  #include "Ponto.h"
5.
6.  namespace geometria
7.  {
8.
9.      // Definição dos dados
10.     typedef struct
11.     {
12.         Ponto ponto;
13.         int raio;
14.     } Circulo;
15.
16.     // Definição das sub-rotinas
17.     int area( const Circulo& circulo );
18.
19.     void imprimir( const Circulo& circulo );
20.
21. }
22.
23. #endif
```

Arquivo de corpo Circulo.cpp

```
1.  #include <iostream>
2.  #include <cmath>
3.
4.  #include "Circulo.h"
5.
6.  #define PI 3.14159
7.
8.  namespace geometria
9.  {
10.
11.     int area( const Circulo& circulo )
12.     {
13.         return PI * circulo.raio * circulo.raio;
14.     }
15.
16.     void imprimir( const Circulo& circulo )
17.     {
18.         imprimir( circulo.ponto );
19.         std::cout << circulo.raio << std::endl;
20.     }
21.
22. }
```

Usando uma biblioteca estática

- Exemplo:
  - Programa que efetua cálculos geométricos com a biblioteca estática criada anteriormente
- Passos:
  - Criar um projeto do tipo aplicação em console
  - Criar um arquivo de corpo contendo o método principal e inserí-lo no projeto
    - O arquivo deve fazer uso das funções definidas na biblioteca estática
  - Configurar o projeto para ter acesso aos protótipos e códigos objeto da biblioteca a fim de linkeditá-la juntamente com o programa
    - Inserir os diretórios dos arquivos de cabeçalho e arquivo de código objeto
    - inserir o arquivo de código objeto (.lib ou .a)
  - Compilar o projeto para gerar o código executável (.exe ou .out)
  - Executar o programa

Arquivo de corpo do método principal

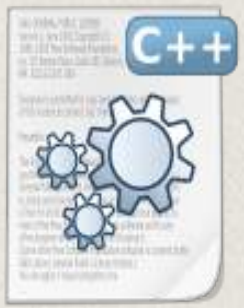
```
1.  #include <iostream>
2.  #include <Ponto.h>
3.  #include <Circulo.h>
4.
5.  using namespace geometria;
6.
7.  int main()
8.  {
9.      Ponto ponto1; ponto1.x = 0; ponto1.y = 0;
10.     Ponto ponto2; ponto2.x = 10; ponto2.y = 5;
11.     Ponto ponto3 = ponto2;
12.     Circulo circulo1; circulo1.ponto.x = 100; circulo1.ponto.y = 50; circulo1.raio = 10;
13.     Circulo circulo2; circulo2.ponto = circulo1.ponto; circulo2.raio = 20;
14.
15.     imprimir( ponto1 );
16.     imprimir( ponto2 );
17.     imprimir( ponto3 );
18.     imprimir( circulo1 );
19.     imprimir( circulo2 );
20.
21.     std::cout << distancia( ponto1, ponto2 ) << std::endl;
22.     std::cout << distancia( circulo1.ponto, ponto3 ) << std::endl;
23.     std::cout << area( circulo2 ) << std::endl;
24.     return 0;
25. }
```

Bibliotecas dinâmicas

- Para serem geradas, as bibliotecas dinâmicas necessitam das seguintes etapas:
  - Pré-processamento
  - Compilação
  - Linkedição
- Arquivos de corpo são utilizados da mesma forma que em um programa comum
- Arquivos de cabeçalho necessitam de macros para explicitar a exportação ou importação do conteúdo da biblioteca
- OBS: Biblioteca dinâmica não contém o método main

Criando uma biblioteca dinâmica

- Exemplo:
  - Biblioteca dinâmica que efetua cálculos geométricos
- Passos:
  - Criar um projeto do tipo biblioteca dinâmica
  - Criar dois arquivos de cabeçalho e inseri-los no projeto
    - Ponto.h (arquivo anterior com macros)
    - Circulo.h (arquivo anterior com macros)
  - Criar dois arquivos de corpo e inseri-los no projeto
    - Ponto.cpp (arquivo anterior com macros)
    - Circulo.cpp (arquivo anterior com macros)
  - Compilar o projeto para gerar o código executável (.dll ou .so)



Arquivo de cabeçalho Ponto.h

```
1.  #ifndef _PONTO_H_
2.  #define _PONTO_H_
3.
4.  #if defined _WIN32 || defined __CYGWIN__  // Macros para Windows
5.      #ifdef BUILDING_DLL  // Identificador a ser definido para o pré-processador
6.          #define DYNAMIC_MODE __declspec(dllexport) // Quando gerando a biblioteca
7.      #else
8.          #define DYNAMIC_MODE __declspec(dllimport) // Quando usando a biblioteca
9.      #endif
10. #else  // Macros para Linux
11.     #define DYNAMIC_MODE __attribute__((visibility("default")))
12. #endif
13.
14. namespace geometria
15. {
16.     typedef struct
17.     {
18.         int x;
19.         int y;
20.     } Ponto;
21.     // sempre inserir o identificador DYNAMIC_MODE nos protótipos de funções
22.     int  DYNAMIC_MODE distancia( const Ponto& ponto1, const Ponto& ponto2 );
23.     void DYNAMIC_MODE imprimir( const Ponto& ponto );
24. }
25. #endif
```

Arquivo de corpo Ponto.cpp

```
1.  #include <iostream>
2.  #include <cmath>
3.
4.  #include "Ponto.h"
5.
6.  namespace geometria
7.  {
8.
9.      int DYNAMIC_MODE distancia( const Ponto& ponto1, const Ponto& ponto2 )
10.     {
11.         return std::sqrt( std::pow( ponto1.x - ponto2.x, 2 ) +
12.                           std::pow( ponto1.y - ponto2.y, 2 ) );
13.     }
14.
15.     void DYNAMIC_MODE imprimir( const Ponto& ponto )
16.     {
17.         std::cout << ponto.x << ", " << ponto.y << std::endl;
18.     }
19.
20. }
```

Arquivo de cabeçalho Circulo.h

```
1.  #ifndef _CIRCULO_H_
2.  #define _CIRCULO_H_
3.  #include "Ponto.h"
4.
5.  #if defined _WIN32 || defined __CYGWIN__  // Macros para Windows
6.      #ifdef BUILDING_DLL  // Identificador a ser definido para o pré-processador
7.          #define DYNAMIC_MODE __declspec(dllexport) // Quando gerando a biblioteca
8.      #else
9.          #define DYNAMIC_MODE __declspec(dllimport) // Quando usando a biblioteca
10.     #endif
11. #else  // Macros para Linux
12.     #define DYNAMIC_MODE __attribute__((visibility("default")))
13. #endif
14.
15. namespace geometria
16. {
17.     typedef struct
18.     {
19.         Ponto ponto;
20.         int raio;
21.     } Circulo;
22.     // sempre inserir o identificador DYNAMIC_MODE nos protótipos de funções
23.     int  DYNAMIC_MODE area( const Circulo& circulo );
24.     void DYNAMIC_MODE imprimir( const Circulo& circulo );
25. }
26. #endif
```

Arquivo de corpo Circulo.cpp

```
1.  #include <iostream>
2.  #include <cmath>
3.
4.  #include "Circulo.h"
5.
6.  #define PI 3.14159
7.
8.  namespace geometria
9.  {
10.
11.     int DYNAMIC_MODE area( const Circulo& circulo )
12.     {
13.         return PI * circulo.raio * circulo.raio;
14.     }
15.
16.     void DYNAMIC_MODE imprimir( const Circulo& circulo )
17.     {
18.         imprimir( circulo.ponto );
19.         std::cout << circulo.raio << std::endl;
20.     }
21.
22. }
```

Usando uma biblioteca dinâmica

- Exemplo:
  - Programa que efetua cálculos geométricos com a biblioteca dinâmica criada anteriormente
- Passos:
  - Criar um projeto do tipo aplicação em console
  - Criar um arquivo de corpo contendo o método principal e inserí-lo no projeto
    - O arquivo deve fazer uso das funções definidas na biblioteca dinâmica
  - Configurar o projeto para ter acesso aos protótipos e referências da biblioteca a fim de linkeditar o programa
    - Inserir os diretórios dos arquivos de cabeçalho e arquivo de referências
    - inserir o arquivo de referências do código objeto (.lib ou .a)
  - Compilar o projeto para gerar o código executável (.exe ou .out)
  - Executar o programa juntamente com o executável da biblioteca (.dll ou .so)

Resumo da aula

- Bibliotecas estáticas necessitam pré-processamento e compilação
- Bibliotecas dinâmicas necessitam também linkedição
- Nas bibliotecas estáticas, arquivos de cabeçalho e de corpo são utilizados da mesma forma que em um programa comum
- Nas bibliotecas dinâmicas, arquivos de cabeçalho necessitam de macros para explicitar a exportação ou importação do conteúdo
- O uso de bibliotecas em um programa requer configurações para que os arquivos necessários possam ser acessados

Exercite-se

- Modificar as bibliotecas apresentadas na aula e os programas que as utilizam para que as funções para Ponto e Circulo passem a ser genéricas através do uso de templates
  - Lembrar que a implementação de funções templates não pode ser feita em arquivos de corpo .cpp, separada da estrutura das classes situadas em arquivos de cabeçalho .h
  - Ao invés disto, devemos implementar as funções templates em arquivos .inl (inline) que devem ser incluídos no final dos arquivos de cabeçalho