

# IMD0030

# LINGUAGEM DE PROGRAMAÇÃO I

Aula 26 – *Namespaces* e Bibliotecas

---

# Objetivos da aula

- Expandir o conceito de modularidade de programas em C++
- Para isto estudaremos:
  - Agrupamentos lógicos baseados em namespaces
  - Criação e uso de bibliotecas
- Ao final da aula espera-se que o aluno seja capaz de:
  - Implementar programas modulares em C++ com o uso de namespaces e bibliotecas

# Namespace

- Mecanismo para expressar agrupamentos lógicos de modo a evitar confusões quando trabalhamos com diversas bibliotecas (que podem conter funções, estruturas ou classes com o mesmo nome)
- Bastante útil para criar registros, funções e classes com nomes já existentes em outras partes do código, sem causar conflito
- Sintaxe para a definição de um namespace:
  - `namespace nomeDoNamespace { corpoDoNamespace }`
- Exemplos:
  - Coleção de registros para geometria
  - Coleção de funções matemáticas

```
1 namespace geometria
2 {
3     struct Ponto { ... };
4     struct Circulo { ... };
5 }
6 namespace math
7 {
8     int max( int a, int b ) {...};
9     int min( int a, int b ) {...};
10 }
```

# Acesso ao namespace

- O acesso aos membros de um namespace pode ser feito assim:
  - Usando o operador `::`
  - Usando o comando `using namespace` nomeDoNamespace
- Por definição, a linguagem C++ utiliza o namespace `std` para definir todas as funções da biblioteca padrão
  - Exemplo: uso da namespace `std` (standard) da biblioteca padrão

```
1  #include <iostream>
2  #include <algorithm>
3
4  int main()
5  {
6      int x, y;
7      std::cin >> x >> y;
8      std::cout << std::max( x, y ) << std::endl;
9      std::cout << std::min( x, y ) << std::endl;
10     return 0;
11 }
```

```
1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4
5  int main()
6  {
7      int x, y;
8      cin >> x >> y;
9      cout << max( x, y ) << endl;
10     cout << min( x, y ) << endl;
11     return 0;
12 }
```

# Exemplo de definição e uso de namespaces

```
1 namespace mat1
2 {
3     template < class T >
4     T max( T a, T b )
5     {
6         return ( a > b ) ? a : b;
7     }
8 }
9
10 namespace mat2
11 {
12     template < class T >
13     T max( T a, T b )
14     {
15         return ( a > b ) ? a : b;
16     }
17 }
18
19 void calcular()
20 {
21     using namespace std;
22     max( 10, 5 );
23 }
```

O namespace **std**  
também inclui uma  
função **max ()**

```
1 #include <algorithm>
2
3 int main()
4 {
5     mat1::max( 10, 5 );
6     calcular();
7     mat2::max( 10, 5 );
8     std::max( 10, 5 );
9
10     return 0;
11 }
```

# Problema com acesso ao namespace (1)

- No exemplo anterior, os namespaces **mat1**, **mat2** e **std** contém a definição da mesma função **max**
- Se não usarmos esses namespaces com cuidado, poderemos gerar um conflito no código do programa

```
1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4  using namespace mat1;
5  using namespace mat2;
6
7  int main() {
8      int x, y;
9      cin >> x >> y;
10
11     // Quais funções serão chamadas? As do namespaces mat1, mat2 ou std?
12     cout << max( x, y ) << endl; // erro de conflito
13
14     return 0;
15 }
```

---

## Problema com acesso ao namespace (2)

- Para resolver este tipo de conflito, podemos ser mais específicos indicando que apenas alguns membros do namespace sejam visíveis

```
1  #include <iostream>
2  using std::cout;
3  using mat1::max;
4
5  int main()
6  {
7      int x, y;
8      std::cin >> x >> y;
9
10     // A função max do namespace mat1 será chamada
11     cout << max( x, y ) << std::endl;
12
13     return 0;
14 }
```

---

# Vantagens da modularidade

- **Código modular permite ser desenvolvido e testado uma só vez**, embora possa ser usado em várias partes de um programa
- **Permite a criação de bibliotecas** que podem ser usadas em diversos programas e por diversos programadores
- **Permite economizar memória** uma vez que o módulo utilizado é armazenado uma única vez, mesmo que utilizado em diferentes partes do programa
- **Permite ocultar código** uma vez que apenas a estrutura do código fica disponível para outros programadores



# Bibliotecas

Criação de bibliotecas em C++

Uso de bibliotecas externas em C++

---

# Bibliotecas

- Uma biblioteca é uma coleção de funções e definições escritas para um propósito definido
  - Qualquer programador C ou C++ já utilizou pelo menos uma biblioteca:
    - No C: a *stdio*, que define funções básicas como: *printf()*, *scanf()*, *fopen()*, *getchar()* e etc
    - No C++: a *iostream*, que define objetos básicos como: *cout*, *cin* e etc
- O uso de bibliotecas facilitam o desenvolvimento de grandes aplicações
  - Além de permitir o reuso de código entre diferentes executáveis e aplicações
- Aplicação e executável são coisas diferentes
  - Aplicação é um conjunto de coisas (executáveis, bibliotecas, arquivos de configuração, etc) e eventualmente pode ser representada somente por um executável quando é algo muito simples

---

# Bibliotecas

- No C/C++ há dois tipos básicos de bibliotecas: estáticas e dinâmicas
  - Estáticas (*static* ou *archive*)
    - Na unificação de código objeto com um programa usuário, o código da biblioteca é incluído no executável
    - Portanto, se houver cinco programas que incluam a mesma biblioteca estática no seu código executável, temos cinco cópias da biblioteca em memória, o que é claramente um desperdício em termos de ocupação da memória
    - Linux/Mac (.a) / Windows (.lib)

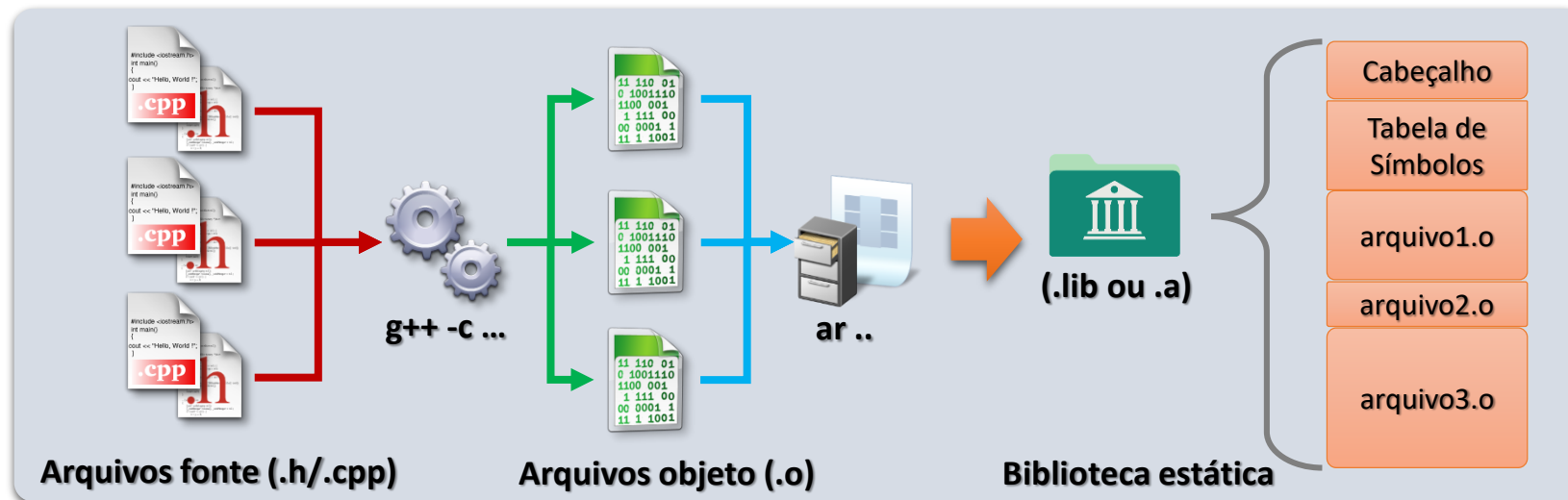
---

# Bibliotecas

- No C/C++ há dois tipos básicos de bibliotecas: estáticas e dinâmicas
  - Dinâmicas ou compartilhadas (*shared*)
    - Tal como o nome sugere, têm o seu código compartilhado pelos programas que as utilizam
    - Só existe uma cópia duma biblioteca dinâmica em memória, independentemente do número de programas usuários
    - Ao contrário das bibliotecas estáticas, estas não são inseridas em sua totalidade, elas são apenas referenciadas no binário final
    - Linux (.so) / Mac (.dylib) / Windows (.dll)

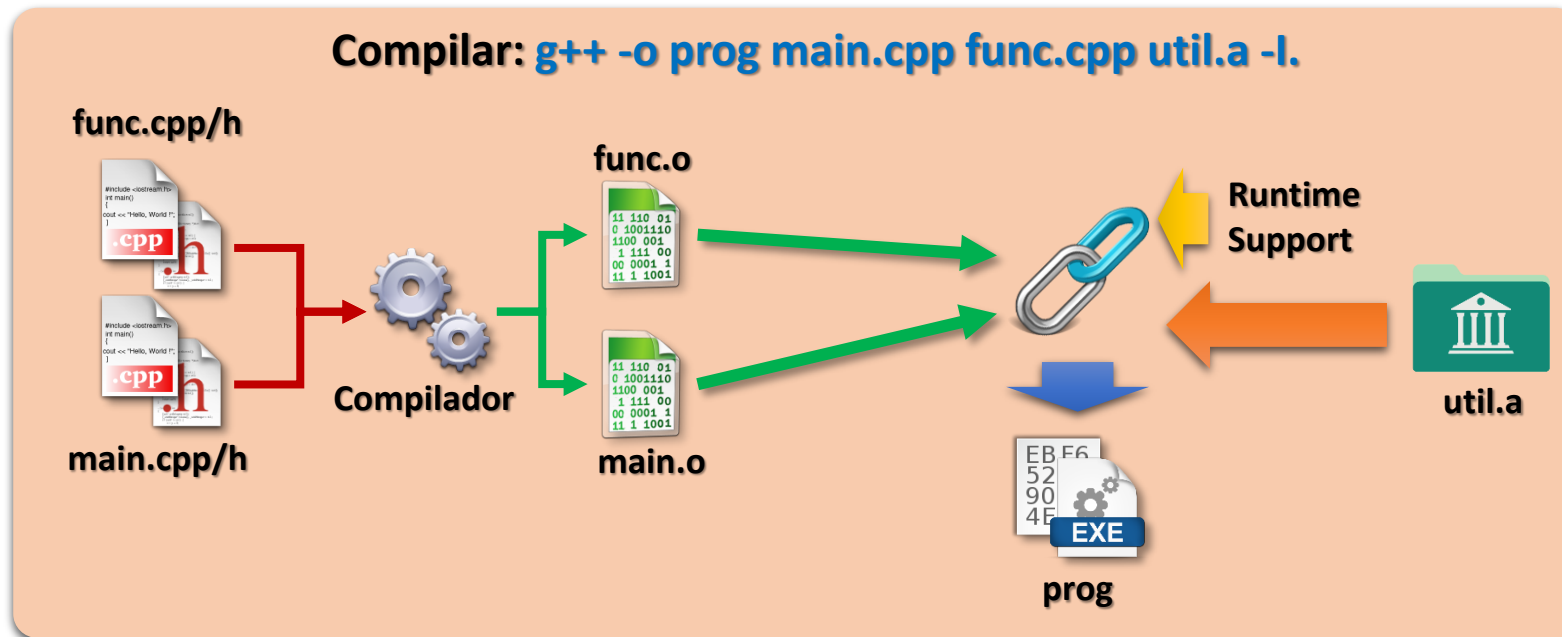
# Biblioteca Estática

- Há duas fases principais na criação de uma biblioteca estática:
  - Criação dos arquivos objeto (\*.o)
    - Nesta fase usa-se o compilador com a diretiva de compilação **-c** (apenas compilar)
  - Junção dos arquivos objeto em uma biblioteca estática (*archive library*)
    - Nesta fase usa-se o arquivador **ar**



# Biblioteca Estática

- Usando uma biblioteca estática para gerar um executável



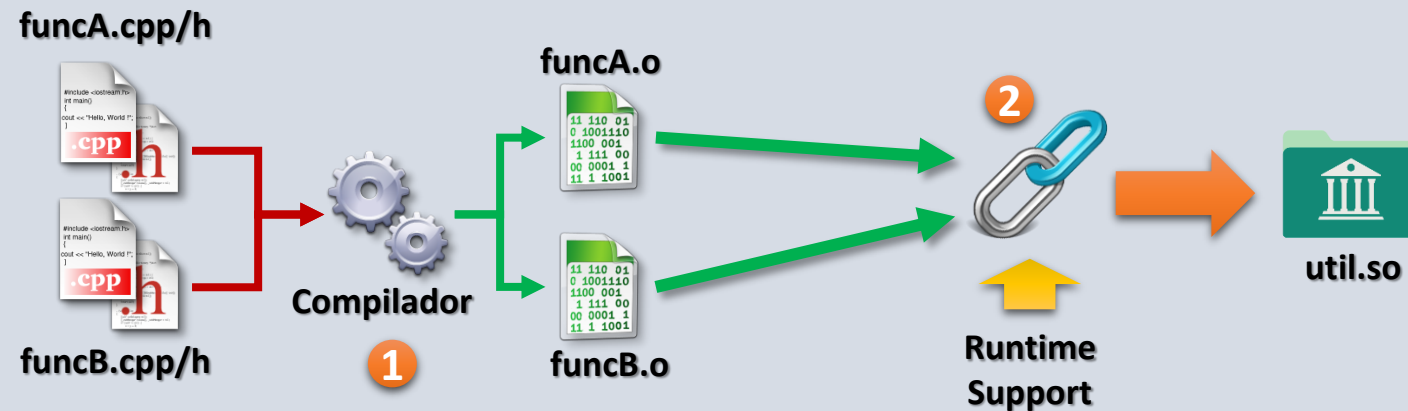
---

# Biblioteca Dinâmica

- Há duas fases principais:
  - Criação dos arquivos objeto (\*.o)
    - Nesta fase usa-se o compilador com a opção **-c**.
  - Junção dos arquivos objeto numa biblioteca compartilhada (*shared object library* ou *dynamic link library*)
    - Nesta fase usa-se o compilador com as opções **-shared -fPIC -o <nome\_biblioteca> arquivos\_objeto**
    - **Nota:** No Linux, esta opção **-fPIC** (*Position-Independent Code*) é necessária e indica que deve ser gerado um módulo objeto que possa ser carregado dinamicamente, e para tanto, o código gerado deve ser independente de posição
      - Neste exemplo, os **arquivos\_objeto** devem ser criados também com a opção **-fPIC**, ou seja, usando o **g++ -c -fPIC ...**

# Biblioteca Dinâmica

- 1 **Compilar:** `g++ -c -fPIC -o funcA.o funcA.cpp -I.`  
`g++ -c -fPIC -o funcB.o funcB.cpp -I.`
- 2 **Ligar:** `g++ -shared -fPIC -o util.so funcA.cpp funcB.cpp -I.`

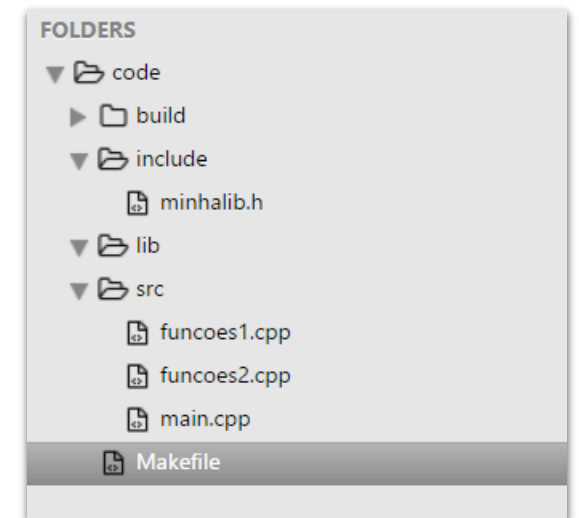




---

# Exemplo: Criando a biblioteca minhalib

- Organização dos arquivos em pastas:
  - ./build
    - Deverá armazenar os arquivos objeto e executáveis/binários
  - ./include
    - Deverá armazenar os arquivos de cabeçalho
  - ./lib
    - Deverá armazenar as bibliotecas (estáticas e dinâmicas)
  - ./src
    - Deverá armazenar os arquivos fonte (código)
  - ./Makefile
    - Arquivo de configuração para a compilação com uso do **make**



---

# Exemplo: Criando a biblioteca minhalib

- ./include/minhalib.h

```
1  #ifndef _MINHALIB_H_
2  #define _MINHALIB_H_
3
4  namespace exemplo
5  {
6
7      extern "C" void imprime(std::string frase);
8      extern "C" int soma(int valorA, int valorB);
9
10     template <typename T>
11     T max (T a, T b)
12     {
13         return (a > b) ? a : b;
14     }
15
16 }
17
18 #endif
```

---

# Exemplo: Criando a biblioteca **minhalib**

- ./src/funcões1.cpp

```
1  #include <iostream>
2  #include "minhalib.h"
3
4  using namespace std;
5
6  namespace exemplo
7  {
8      void imprime(std::string frase)
9      {
10         cout << frase << endl;
11     }
12 }
```

- ./src/funcões2.cpp

```
1  #include <iostream>
2  #include "minhalib.h"
3
4  using namespace std;
5
6  namespace exemplo
7  {
8      int soma(int valorA, int valorB)
9      {
10         return (valorA+valorB);
11     }
12 }
```

---

# Exemplo: Criando a biblioteca minhalib

- ./src/main.cpp

```
1  #include <iostream>
2  #include "minhalib.h"
3
4  using namespace std;
5  using namespace exemplo;
6
7  int main(void)
8  {
9      string frase = "Olá mundo!";
10
11     imprime(frase);
12     cout << "Resultado da soma 5+6: " << soma(5,6) << endl;
13     cout << "O maior valor entre 5 e 6: " << exemplo::max(5,6) << endl;
14 }
```

---

# Exemplo: Criando a biblioteca minhalib

- ./Makefile (Parte I)

```
1  INC_DIR = include
2  SRC_DIR = src
3  OBJ_DIR = build
4  LIB_DIR = lib
5
6  CC      = g++
7  CFLAGS  = -Wall -pedantic -std=c++11 -ansi -I. -I$(INC_DIR)
8  ARCHIVE = ar
9
10 linux: minhalib.a minhalib.so prog_estatico prog_dinamico
11
12 windows: minhalib.lib minhalib.dll prog_estatico.exe prog_dinamico.exe
13
```

# Exemplo: Criando a biblioteca minhalib

```
14 # LINUX
15
16 minhalib.a: $(SRC_DIR)/funcoes1.cpp $(SRC_DIR)/funcoes2.cpp $(INC_DIR)/minhalib.h
17     $(CC) $(CFLAGS) -c $(SRC_DIR)/funcoes1.cpp -o $(OBJ_DIR)/funcoes1.o
18     $(CC) $(CFLAGS) -c $(SRC_DIR)/funcoes2.cpp -o $(OBJ_DIR)/funcoes2.o
19     $(AR) rcs $(LIB_DIR)/$@ $(OBJ_DIR)/funcoes1.o $(OBJ_DIR)/funcoes2.o
20     @echo "+++ [Biblioteca estatica criada em $(LIB_DIR)/$@] +++"
21
22 minhalib.so: $(SRC_DIR)/funcoes1.cpp $(SRC_DIR)/funcoes2.cpp $(INC_DIR)/minhalib.h
23     $(CC) $(CFLAGS) -fPIC -c $(SRC_DIR)/funcoes1.cpp -o $(OBJ_DIR)/funcoes1.o
24     $(CC) $(CFLAGS) -fPIC -c $(SRC_DIR)/funcoes2.cpp -o $(OBJ_DIR)/funcoes2.o
25     $(CC) -shared -fPIC -o $(LIB_DIR)/$@ $(OBJ_DIR)/funcoes1.o $(OBJ_DIR)/funcoes2.o
26     @echo "+++ [Biblioteca dinamica criada em $(LIB_DIR)/$@] +++"
27
28 prog_estatico:
29     $(CC) $(CFLAGS) $(SRC_DIR)/main.cpp $(LIB_DIR)/minhalib.a -o $(OBJ_DIR)/$@
30
31 prog_dinamico:
32     $(CC) $(CFLAGS) $(SRC_DIR)/main.cpp $(LIB_DIR)/minhalib.so -o $(OBJ_DIR)/$@
```

./Makefile (Parte II)

# Exemplo: Criando a biblioteca minhalib

```
33
34 # WINDOWS
35
36 minhalib.lib: $(SRC_DIR)/funcoes1.cpp $(SRC_DIR)/funcoes2.cpp $(INC_DIR)/minhalib.h
37     $(CC) $(CFLAGS) -c $(SRC_DIR)/funcoes1.cpp -o $(OBJ_DIR)/funcoes1.o
38     $(CC) $(CFLAGS) -c $(SRC_DIR)/funcoes2.cpp -o $(OBJ_DIR)/funcoes2.o
39     $(AR) rcs $(LIB_DIR)/$@ $(OBJ_DIR)/funcoes1.o $(OBJ_DIR)/funcoes2.o
40     @echo "+++ [Biblioteca estatica criada em $(LIB_DIR)/$@] +++"
41
42 minhalib.dll: $(SRC_DIR)/funcoes1.cpp $(SRC_DIR)/funcoes2.cpp $(INC_DIR)/minhalib.h
43     $(CC) $(CFLAGS) -c $(SRC_DIR)/funcoes1.cpp -o $(OBJ_DIR)/funcoes1.o
44     $(CC) $(CFLAGS) -c $(SRC_DIR)/funcoes2.cpp -o $(OBJ_DIR)/funcoes2.o
45     $(CC) -shared -o $(LIB_DIR)/$@ $(OBJ_DIR)/funcoes1.o $(OBJ_DIR)/funcoes2.o
46     @echo "+++ [Biblioteca dinamica criada em $(LIB_DIR)/$@] +++"
47
48 prog_estatico.exe:
49     $(CC) $(CFLAGS) $(SRC_DIR)/main.cpp $(LIB_DIR)/minhalib.lib -o $(OBJ_DIR)/$@
50
51 prog_dinamico.exe:
52     $(CC) $(CFLAGS) $(SRC_DIR)/main.cpp $(LIB_DIR)/minhalib.dll -o $(OBJ_DIR)/$@
```

./Makefile (Parte III)

---

# Exemplo: Criando a biblioteca minhalib

- ./Makefile (Parte IV)

```
53  
54 clean:  
55     @echo "Removendo arquivos objeto e executaveis/binarios..."  
56     @rm -rf $(OBJ_DIR)/*
```



# Exemplo: Criando a biblioteca minhalib

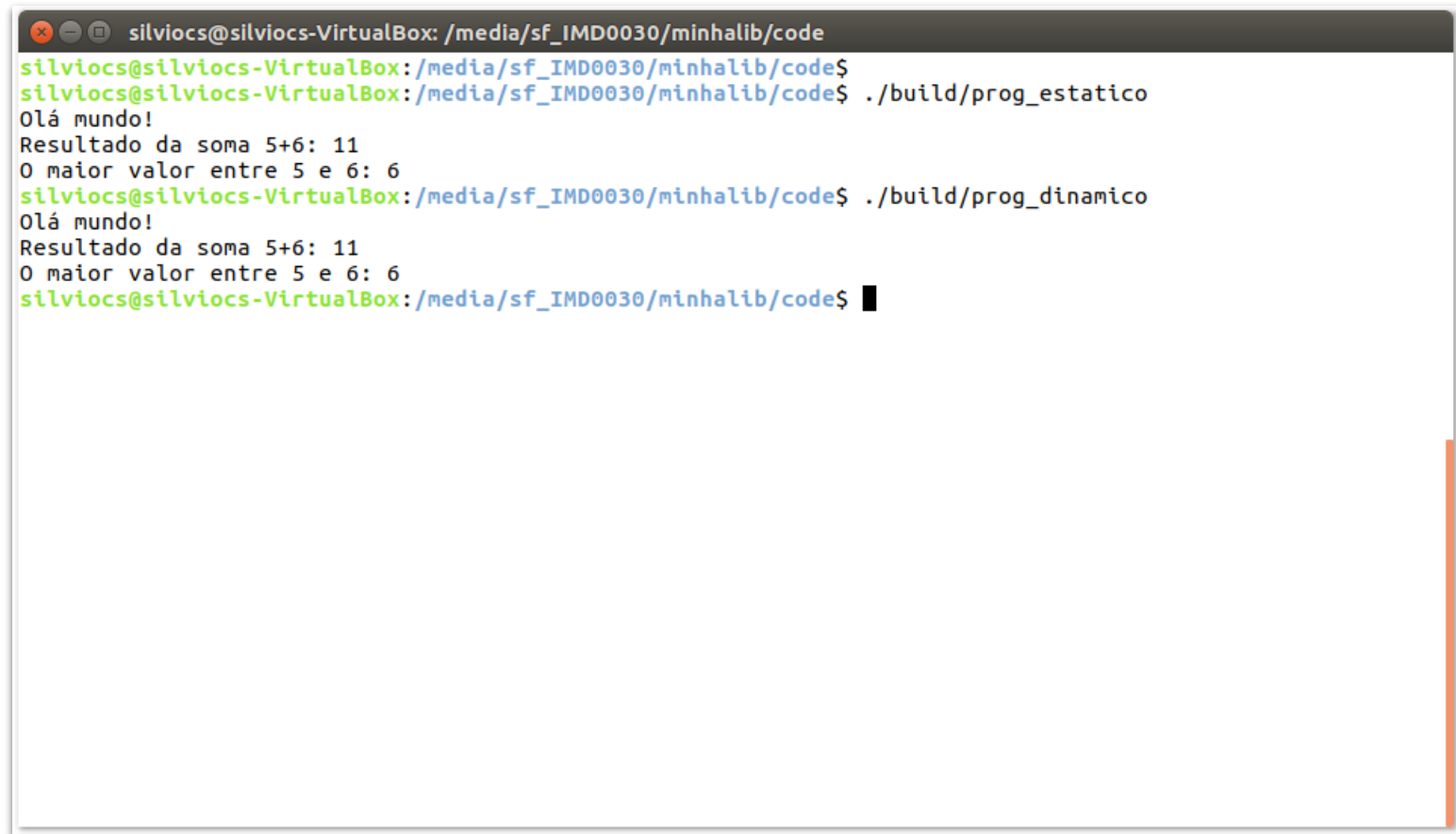
- Verificando
  - (Linux)

```
silviocs@silviocs-VirtualBox: /media/sf_IMD0030/minhalib/code
silviocs@silviocs-VirtualBox: /media/sf_IMD0030/minhalib/code$ du -a
2      ./build/funcoes1.o
2      ./build/funcoes2.o
14     ./build/prog_dinamico
1443   ./build/prog_dinamico.exe
15     ./build/prog_estatico
1444   ./build/prog_estatico.exe
2923   ./build
1      ./include/minhalib.h
1      ./include
6      ./lib/minhalib.a
1454   ./lib/minhalib.dll
4      ./lib/minhalib.lib
9      ./lib/minhalib.so
1477   ./lib
3      ./Makefile
1      ./src/funcoes1.cpp
1      ./src/funcoes2.cpp
1      ./src/main.cpp
2      ./src
4408   .
silviocs@silviocs-VirtualBox: /media/sf_IMD0030/minhalib/code$ ldd build/prog_dinamico
linux-vdso.so.1 => (0x00007ffff9bd2000)
lib/minhalib.so (0x00007f2b97092000)
libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f2b96ced000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f2b96ad6000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2b9670f000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f2b96406000)
/lib64/ld-linux-x86-64.so.2 (0x00005646c2a4e000)
silviocs@silviocs-VirtualBox: /media/sf_IMD0030/minhalib/code$
```

---

# Exemplo: Criando a biblioteca minhalib

- Executando
  - (Linux)



```
silviocs@silviocs-VirtualBox: /media/sf_IMD0030/minhalib/code
silviocs@silviocs-VirtualBox: /media/sf_IMD0030/minhalib/code$
silviocs@silviocs-VirtualBox: /media/sf_IMD0030/minhalib/code$ ./build/prog_estatico
Olá mundo!
Resultado da soma 5+6: 11
0 maior valor entre 5 e 6: 6
silviocs@silviocs-VirtualBox: /media/sf_IMD0030/minhalib/code$ ./build/prog_dinamico
Olá mundo!
Resultado da soma 5+6: 11
0 maior valor entre 5 e 6: 6
silviocs@silviocs-VirtualBox: /media/sf_IMD0030/minhalib/code$ █
```

# Alguma Questão?

