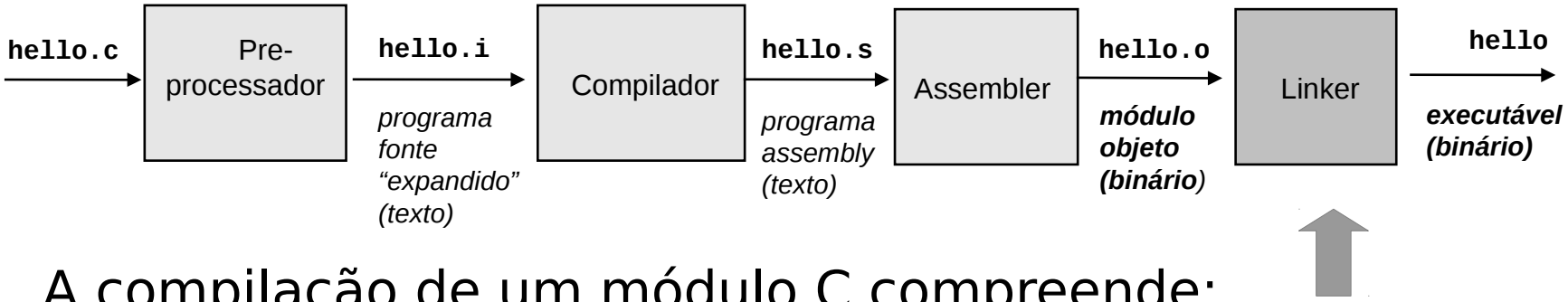
Ligação e Relocação

Noemi Rodriguez
Ana Lúcia de Moura

http://www.inf.puc-rio.br/~inf1018

Compilação e Ligação



A compilação de um módulo C compreende:

- pré-processamento
- geração de código *assembly*
- geração do módulo objeto

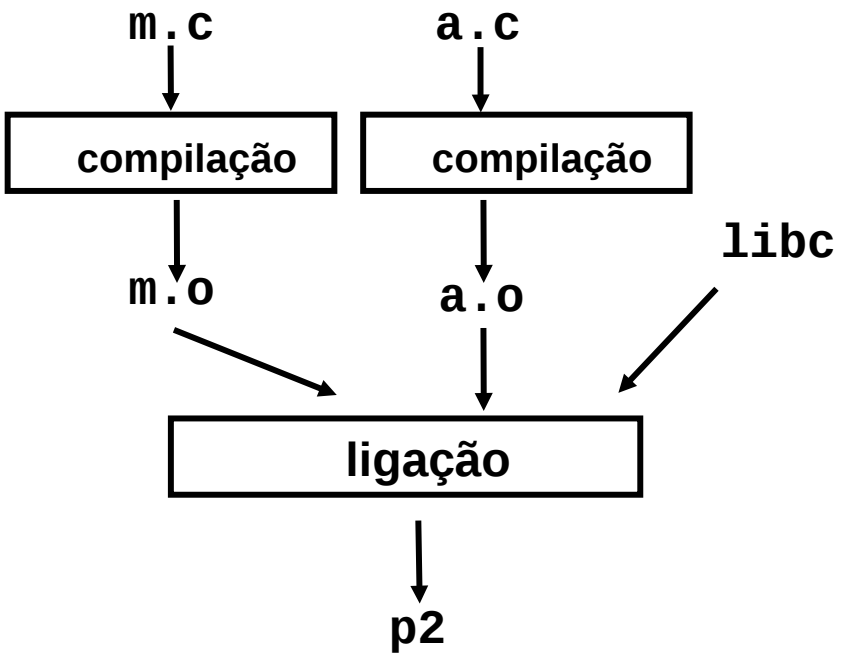
Um módulo objeto não pode ser executado:

- dependência de outros módulos (e bibliotecas): **ligação**
- endereços são relativos: **relocação**

Ligador (*linkeditor*)

Combina módulos objeto em arquivo executável

- união dos módulos (código e dados)
- resolução das referências externas
- relocação das referências à memória




```
gcc –o p2 m.c a.c
```

Módulos de bibliotecas estáticas são incluídos

- uma biblioteca reúne **módulos objeto** (libc, libm, ...)

Arquivo Objeto

Código, dados e informações para a “ligação” com outros módulos

- texto (código de máquina), dados (ro, inicializados e não inicializados/inicializados com 0)
- **Tabela de Símbolos**
  - símbolos **definidos** pelo módulo ("exportações")
  - símbolos **referenciados** pelo módulo ("importações")
  - símbolos “locais” ao módulo (static)
- **Dicionário de Relocação**
  - localização de referências a memória a serem preenchidas ou corrigidas (instruções e dados)

Exemplo Simplificado

```
extern int var1;
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var1);
  …
}
```

Exemplo Simplificado

```
extern int var1;
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var1);
  …
}
```

```
.data
.globl var2
var2: int 0

.text
.globl fun2
fun2: pushq %rbp
   …
   movl var1, %edi

   call fun1
```

Exemplo Simplificado

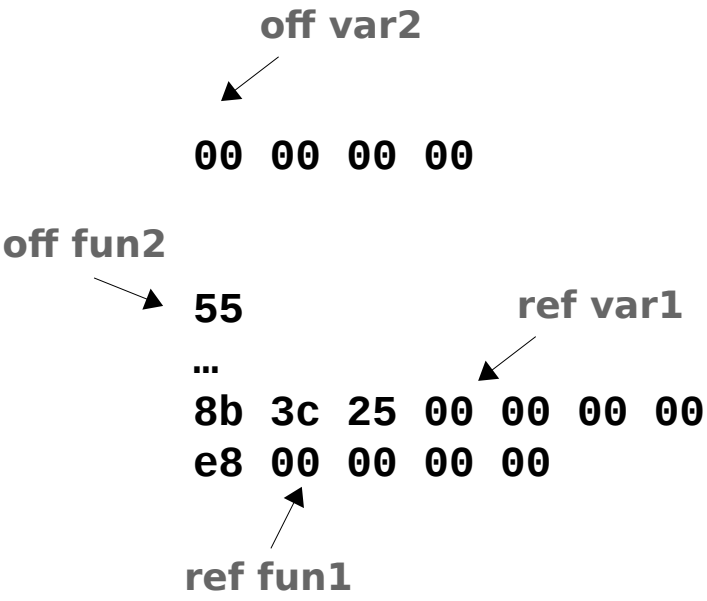
```
extern int var1;
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var1);
  …
}
```

```
.data
.globl var2
var2: int 0

.text
.globl fun2
fun2: pushq %rbp
   …
   movl var1, %edi

   call fun1
```



Exemplo Simplificado

```
extern int var1;
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var1);
  …
}
```

```
.data
.globl var2
var2: int 0

.text
.globl fun2
fun2: pushq %rbp
   …
   movl var1, %edi

   call fun1
```

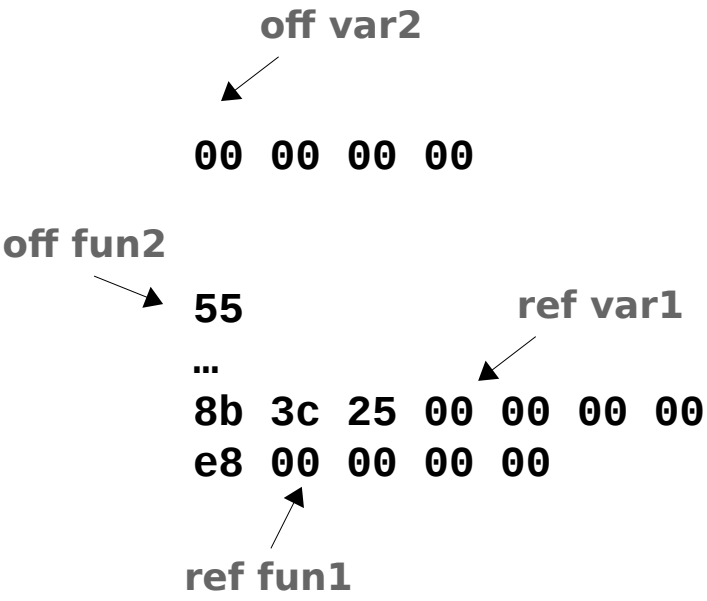


Tabela de Símbolos

var2	D	<off var2>
fun2	T	<off fun2>
var1	U	
fun1	U	

Exemplo Simplificado

```
extern int var1;
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var1);
  …
}
```

```
.data
.globl var2
var2: int 0

.text
.globl fun2
fun2: pushq %rbp
    …
    movl var1, %edi

    call fun1
```

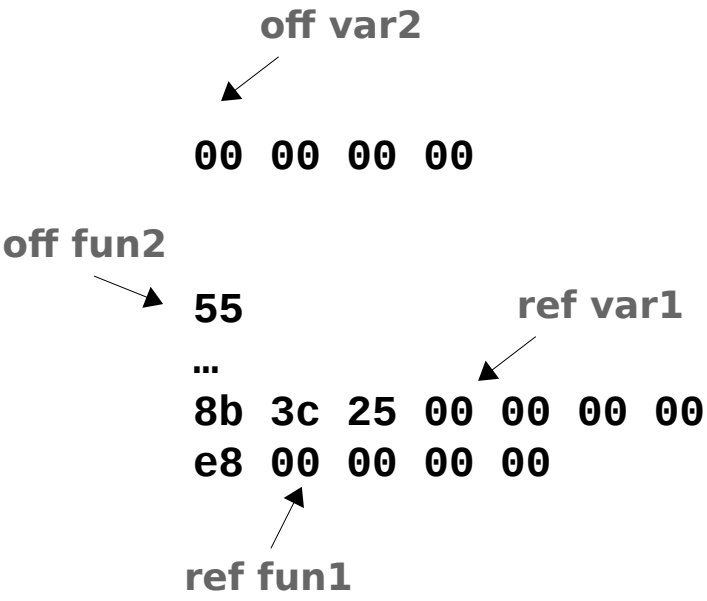


Tabela de Símbolos

var2	D	<off var2>
fun2	T	<off fun2>
var1	U	
fun1	U	

Dicionário de Relocação

<ref var1>	ref	var1
<ref fun1>	ref rel	fun1

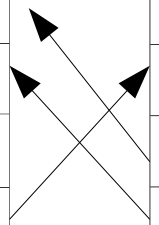
Resolução de Referências

Módulo 1

var1	D	<off var1>
fun1	T	<off fun1>
main	T	<off main>
fun2	U	

Módulo 2

var2	D	<off var2>
fun2	T	<off fun2>
var1	U	
fun1	U	



Cada referência externa deve ser associada a uma definição única (exportação) de um símbolo **com o mesmo nome**

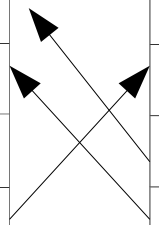
Resolução de Referências

Módulo 1

var1	D	<off var1>
fun1	T	<off fun1>
main	T	<off main>
fun2	U	

Módulo 2

var2	D	<off var2>
fun2	T	<off fun2>
var1	U	
fun1	U	



Cada referência externa deve ser associada a uma definição única (exportação) de um símbolo **com o mesmo nome**

- se nenhuma definição com esse nome é encontrada, o ligador termina com erro
- se há duas (ou mais) definições com o mesmo nome, o ligador também termina com erro

Erros de Ligação

```
void foo(void);

int main() {
  foo();
  return 0;
}
```

```
/tmp/cc0cCWXd.o: In function `main':
teste.c:(.text+0x7): undefined reference to
   `foo'
collect2: ld returned 1 exit status
```

```
int i = 1;

int foo(int j) {
  return i + j;
}
```

```
int foo(int);
int i = 0;

int main() {
  i = foo(3);
  return 0;
}
```

```
gcc -o t foo.o m.o
m.o:(.bss+0x0): multiple definition of `i'
foo.o:(.data+0x0): first defined here
collect2: ld returned 1 exit status
```

Referências em C

É importante distinguir **definição** e **referência**

- uma definição estabelece a representação na memória (localização, tamanho, ...)

Em ISO C:

- uma declaração **com inicialização** é uma **definição** (símbolo “forte”)

```
int i = 1024;
```

- uma declaração com classe **extern** é uma **referência**

```
extern int i;
```

- uma declaração sem inicialização (e sem *extern*) é uma tentativa de definição (símbolo “fraco”)
  - não deve ser usada!

```
int i;
```

Definições com mesmo nome: bug

```
#include <stdio.h>
void f(void);

int x = 15212;
int y = 15213;

int main() {
  f();
  printf("%d %d\n",x,y);
  return 0;
}
```

```
gcc -Wall -o f f.c f1.c
/usr/bin/ld: Warning: alignment 4 of
   symbol `x' in /tmp/ccP4WG2t.o is
   smaller than 8 in /tmp/ccobMeSD.o

./f
0 0
```

```
double x;

void f() {
    x = 0.0;
 }
```

Cuidados e Boas Práticas

modulo1.h

```
int foo(void);
```

modulo2.h

```
extern int contador;
```

modulo1.c

```
#include “modulo1.h”
#include “modulo2.h”

int foo() {
  contador++;
  …
}
```

modulo2.c

```
#include “modulo1.h”
#include “modulo2.h
int contador = 0;
int main (void) {
  int l = foo();
  …
}
```

Um símbolo global deve ser declarado **com o mesmo tipo** em todos os módulos

- a resolução de referências é feita com base apenas no **nome**!
- o compilador não tem como garantir consistência a não ser com o uso de arquivos de cabeçalho ("interface")

Relocação de Referências

O Dicionário de Relocação indica que referências à memória do módulo devem ser preenchidas/corrigidas

- referências externas ou a endereços que dependem da localização "final" do trecho de código/dados correspondente

```
void fun1 (int x);

int var2 = 0;
int fun2() {
  …
  fun1(var2);
  …
}
```

```
.data
.globl var2
var2: int 0
.text
.globl fun2
fun2: pushq %rbp
   …
   movl var2, %edi

   call fun1
```

```
55
…
8b 3c 25 00 00 00 00
e8 00 00 00 00
```

ref var2

ref fun1

Relocação de Referências

O ligador produz uma tabela com os endereços dos símbolos globais definidos por todos os módulos

Tabela de Símbolos

var1	D	<end var1>
var2	D	<end var2>
fun1	T	<end fun1>
main	T	<end main>
fun2	T	<end fun2>

```
55                ref var2
…                    ↓
8b 3c 25 00 00 00 00
e8 00 00 00 00
   ↑
ref fun1
```

Dicionário de Relocação

<ref var2>	ref	var2
<ref fun1>	ref rel	fun1

Carga e Relocação

Na ligação, as referências à memória são calculadas em relação ao endereço "virtual" do programa

- quando o programa é carregado na memória, essas referências devem ser relocadas em relação ao endereço "real"
- essa relocação pode ser feita através de uma tradução de endereços feita pelo hardware

Bibliotecas Dinâmicas

Carregadas na memória e "ligadas" em tempo de execução a um programa

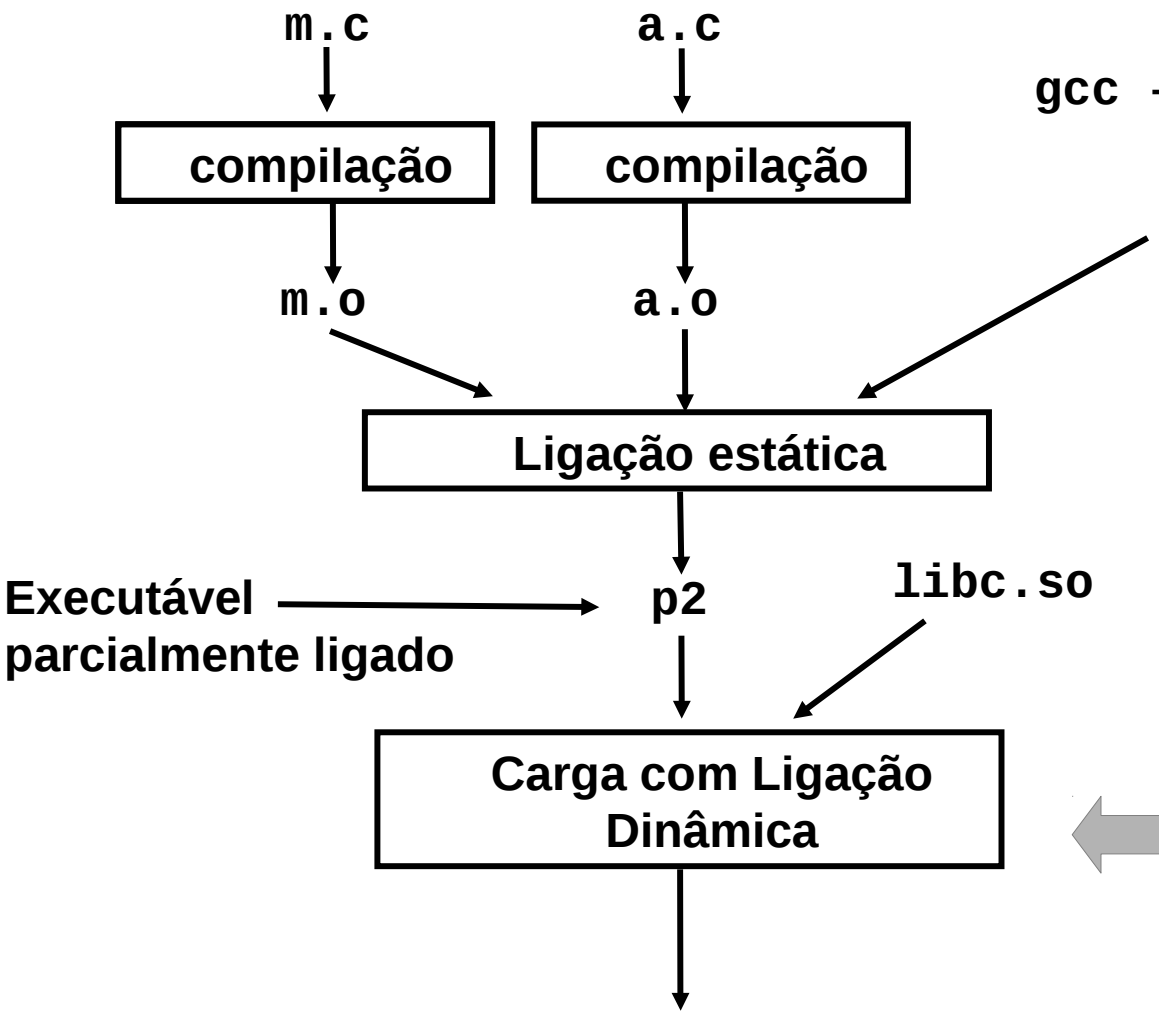
- *shared objects* (.so), *dynamic link libraries* (.dll)

Melhor aproveitamento de espaço

- bibliotecas estáticas são incorporadas no programa
- bibliotecas dinâmicas são compartilhadas

A carga e ligação da biblioteca é realizada por um *ligador dinâmico*

Ligação com Bibliotecas Dinâmicas



`gcc –o p2 m.c a.c`

`libc.so`

(tabela de símbolos e informações para relocação)

módulos de `libc.so` são carregados na memória

referências a símbolos de libc em p2 são relocadas

DEPARTAMENTO DE INFORMÁTICA
PUC·RIO