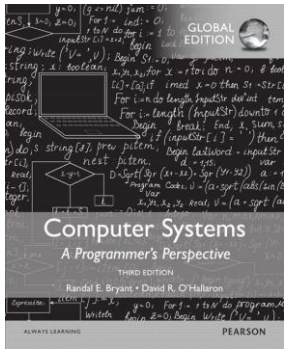


Ficheiros objeto relocáveis

Bib: Computer Systems: A Programmer's Perspective (cap. 7, .2-.5)

Programação em Sistemas Computacionais

João Pedro Patriarca (joao.patriarca@isel.pt), Gabinete F.0.23 do edifício F

ISEL, ADEETC, LEIC

Agenda

- Formato ELF (*Executable and Linkable Format*) para ficheiros objeto relocáveis

Código base exemplo

Processo de geração de um executável

main.c

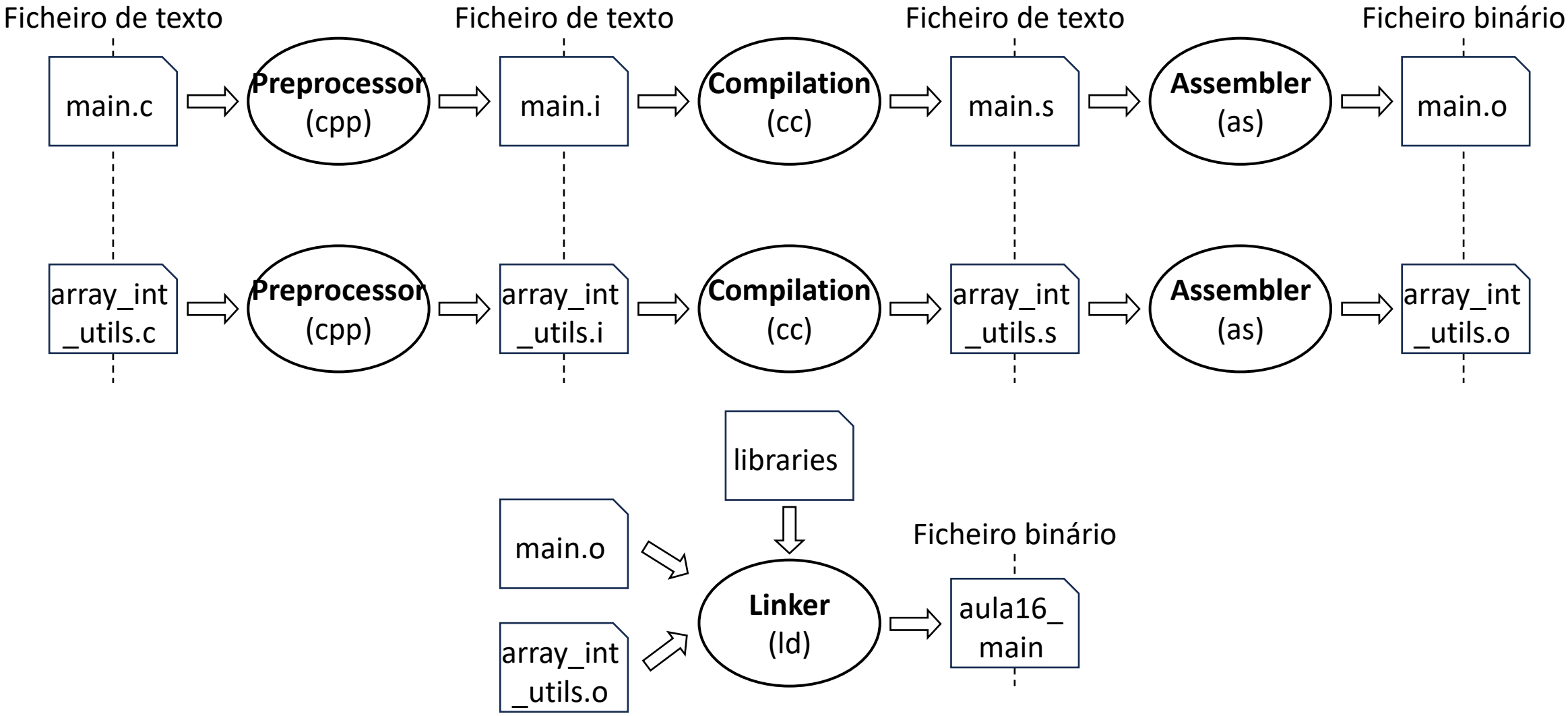
```c
#include "../common.h"
void aprint(int a[], int size);
int  asum(int a[], int size);

int main() {
    int a1[] = {1, 2, 3, 4};
    int sum = asum(
        a1, ARRAY_SIZE(a1)
    );
    aprint(a1, ARRAY_SIZE(a1));
    printf("Sum of a1=%d\n", sum);
    return 0;
}
```

array_int_utils.c

```c
#include <stdio.h>
void aprint(int a[], int size) {
    for (int i = 0; i < size-1; i++)
        printf("a[%d]=%d, ",
                i, a[i]);
    if (size > 0)
        printf("a[%d]=%d\n",
                size-1, a[size-1]);
}
int asum(int a[], int size) {
    int r = 0;
    for (int i = 0; i < size; i++)
        r += a[i];
    return r;
}
```

Compilação separada – porquê?
Processo de geração de um executável

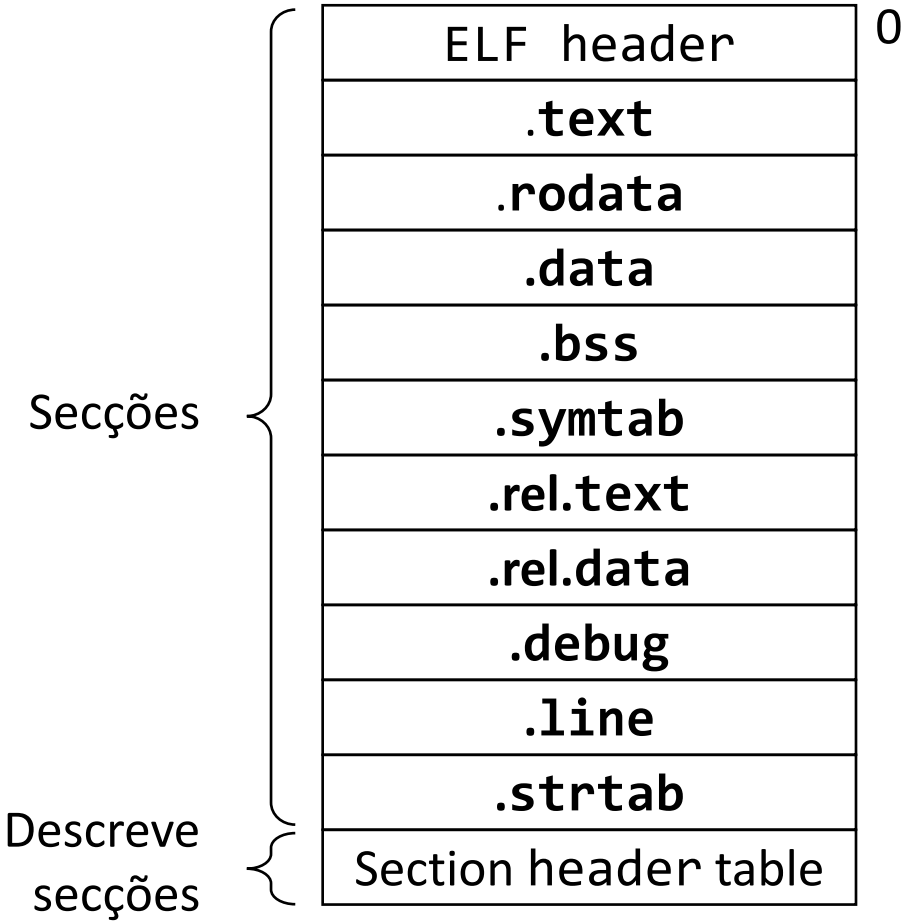


Ficheiros objeto

- Ficheiros objeto correspondem a ficheiros binários que obedecem a um determinado formato
  - Nos sistemas Linux atuais, formato ELF (*Executable and Linkable Format*)
  - Nos sistemas Windows, formato PE (*Portable Executable*)

- Tipos de ficheiros objeto
  - Ficheiros objeto relocáveis: gerados pelo compilador de C ou compilador de *assembly*;
  - Ficheiros objeto executáveis: gerados pelo *linker*;
  - Ficheiros objeto partilháveis: gerados pelo *linker* na forma de bibliotecas de carregamento dinâmico (tema estudado em próximas lições).

Estrutura típica de um ficheiro objeto relocável



`.text`: código

`.rodata`: dados apenas com acesso de leitura (ex: *strings* com os operadores de controlo da função `printf`)

`.data`: dados globais e variáveis estáticas iniciadas

`.bss`: dados globais e variáveis estáticas não iniciadas ou iniciados com 0. Esta secção não ocupa espaço no ficheiro

`.symtab`: tabela de símbolos com nomes de funções e de variáveis globais definidas e referenciadas neste módulo

`.rel.text`: lista de localizações na secção **`.text`** que precisam ser modificadas na fase de ligação. Tipicamente, corresponde a chamadas diretas a funções definidas noutros módulos e acesso a variáveis globais

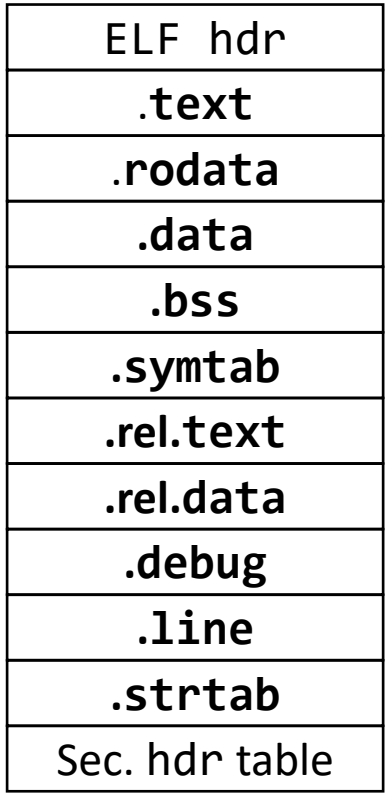
`.rel.data`: o mesmo que a secção **`.rel.text`** mas aplicado à secção **`.data`**. Tipicamente, quando uma variável global é iniciada com o endereço de outra variável global

`.debug`, **`.line`**, **`.strtab`**: menos relevantes para o estudo. As duas primeiras relacionadas com informação de *debug* e apenas existem se o módulo foi compilado com a opção **`-g`** e a terceira para dar suporte a outras secções

Ferramenta `objdump`

- A ferramenta **`objdump`** permite interpretar os dados de um ficheiro objeto (**`$ objdump <options> <object_file>`**)

Opção	Descrição
`-x`	Informação disponível de todos os *headers* (eq: **`-a -f -h -p -r -t`**)
`-h`	Informação do *section header table*
`-s`	Conteúdo completo de todas as secções, por omissão, ou da secção especificada (depende da opção **`-j`**)
`-j <section_name>`	Especifica uma secção (implica a opção **`-s`**)
`-d`	*Disassembly* da secção **`.text`**
`-S`	Apresenta código fonte (em C e compilado com a opção −g) intercalado com o *disassembly* (implica a opção **`-d`**)
`-r`	Apresenta as entradas de relocação. Juntamente com a opção **`-d`**, as entradas são apresentadas junto do código onde o *linker* precisa intervir
`-t`	Tabela de símbolos (preferível usar a ferramenta `nm`)



Ferramenta nm

- A ferramenta **nm** permite interpretar a tabela de símbolos de um ficheiro objeto (**$ nm <options> <object_file>**)

Opção	Descrição
-n	Apresenta símbolos por ordem crescente de endereços
-S	Imprime igualmente a dimensão associada ao símbolo definido
-h	Imprime um resumo das opções
Outras	**$ man nm**

```
$ nm aula16_main.o
         U aprint
         U asum
0...0000 T main
         U printf
         U __stack_chk_fail
```

```
$ nm array_int_utils.o
0...0000 T aprint
0...0097 T asum
         U printf
```

Formato do *output* produzido por **nm**

- Formato apresentado pelo **nm** com a opção **-S**
  <endereço> <dimensão> <tipo de símbolo> <símbolo>
- Tipo de símbolo:
  - Apresentado com letra maiúscula: símbolo com visibilidade global
  - Apresentado com letra minúscula: símbolo com visibilidade local ao módulo

Tipo de símbolo	Descrição
T ou t	Símbolo presente na secção `.text`
D ou d	Símbolo presente na secção `.data`
B ou b	Símbolo presente na secção `.bss`
C	~~Símbolo comum (COMMON) com dados não iniciados~~ (a partir do GCC versão 10, por omissão, deixam de existir símbolos fracos (*weak symbols*))
U	Símbolo indefinido
Outros	**$ man nm**

Associação de símbolos a secções versus tipos de símbolos

Símbolo	Secção	Tipo
Definição de variáveis globais iniciadas com valores diferentes de 0	`.data`	D
Definição de variáveis globais iniciadas com valores diferentes de 0 e marcadas com `static`	`.data`	d
Definição de variáveis globais não iniciadas ou iniciadas com 0	`.bss`	B
Definição de variáveis globais não iniciadas ou iniciadas com 0 e marcadas com `static`	`.bss`	b
Definição de variáveis locais a uma função iniciadas com valores diferentes de 0 e marcadas com `static`	`.data`	d
Definição de variáveis locais a uma função não iniciadas ou iniciadas com 0 e marcadas com `static`	`.bss`	b
Definição de funções	`.text`	T
Definição de funções marcadas com `static`	`.text`	t
Declaração de variáveis e funções como externas (`extern`)	`--`	U

Variáveis locais a uma função, não estando marcadas com `static`, não produzem informação simbólica

Exemplo com a produção de vários tipos de símbolos

Para versões do compilador >= 10, usam, por omissão a opção `-fno-common`

```c
int sym_data_public = 1;
static int sym_data_local = 2;
short sym_bss_public1;
long sym_bss_public2 = 0;
static int sym_bss_local;
extern int sym_undef_public1;
extern int sym_undef_public2();
int sym_undef_public3();
int sym_undef_public4();
static long sym_text_local() {
    static long sym_bss_local;
    static long sym_data_local = 3;
    int sym_stack = 4;
    sym_bss_local += 1;
    sym_data_local += 1;
    return sym_bss_public1 + sym_bss_local + sym_data_local + sym_stack;
}
long sym_text_public() {
    const char * s = "Read only section";
    return sym_text_local() + *s + sym_data_local +
           sym_bss_local + sym_undef_public1 +
           sym_undef_public2() + sym_undef_public3();
}
```

```
$ gcc -c sym1.c
$ nm sym1.o
0000000000000010 b sym_bss_local
0000000000000018 b sym_bss_local.1
0000000000000000 B sym_bss_public1
0000000000000008 B sym_bss_public2
0000000000000004 d sym_data_local
0000000000000008 d sym_data_local.0
0000000000000000 D sym_data_public
0000000000000000 t sym_text_local
000000000000005c T sym_text_public
                 U sym_undef_public1
                 U sym_undef_public2
                 U sym_undef_public3
```

Exemplo com a produção de vários tipos de símbolos

Para versões do compilador < 10, usam, por omissão a opção `-fcommon`

```c
int sym_data_public = 1;
static int sym_data_local = 2;
short sym_bss_public1;
long sym_bss_public2 = 0;
static int sym_bss_local;
extern int sym_undef_public1;
extern int sym_undef_public2();
int sym_undef_public3();
int sym_undef_public4();
static long sym_text_local() {
    static long sym_bss_local;
    static long sym_data_local = 3;
    int sym_stack = 4;
    sym_bss_local += 1;
    sym_data_local += 1;
    return sym_bss_public1 + sym_bss_local + sym_data_local + sym_stack;
}
long sym_text_public() {
    const char * s = "Read only section";
    return sym_text_local() + *s + sym_data_local +
           sym_bss_local + sym_undef_public1 +
           sym_undef_public2() + sym_undef_public3();
}
```

```
$ gcc -c -fcommon sym1.c
$ nm sym1.o
0000000000000008 b sym_bss_local
0000000000000010 b sym_bss_local.1
0000000000000002 C sym_bss_public1
0000000000000000 B sym_bss_public2
0000000000000004 d sym_data_local
0000000000000008 d sym_data_local.0
0000000000000000 D sym_data_public
0000000000000000 t sym_text_local
000000000000005c T sym_text_public
                 U sym_undef_public1
                 U sym_undef_public2
                 U sym_undef_public3
```

Exercício 1

- Indique o conteúdo das tabelas de símbolos dos ficheiros objeto relocáveis, resultantes da compilação de **`ex1_m.c`** e **`ex1_swap.c`**. Para cada símbolo, indique o nome, a secção e o respetivo âmbito (local ou global). Apresente a resposta segundo a convenção **`nm`**.

ex1_m.c

```c
void swap();

int buf[2] = {1, 2};

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

ex1_swap.c

```c
extern int buf[];
int *bufp0 = &buf[0];
int *bufp1;

void swap() {
    int temp;
    bufp1 = &buf[1];
    temp = *bufp0;
    *bufp0 = *bufp1;
    *bufp1 = temp;
}
```

Exercício 2

- Apresente uma definição possível para cada um dos símbolos com base no resultado produzido pela ferramenta (**`$ nm -S ex2.o`**)

```
nm -S ex2.o
0000000000000000 0000000000000042 T f1
0000000000000042 0000000000000011 t f2
                                   U f3
0000000000000000 0000000000000014 D v1
0000000000000000 0000000000000008 B v2
0000000000000008 0000000000000008 B v3
0000000000000014 0000000000000002 D v4
                                   U v5
0000000000000000 0000000000000008 d v6
0000000000000010 0000000000000004 b v7
0000000000000018 0000000000000008 d v8
```