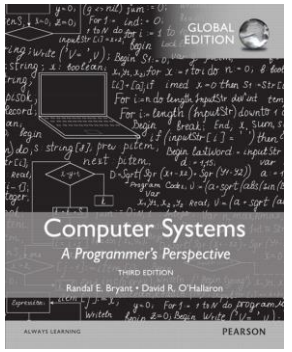


Processo de geração de um executável

Bib: Computer Systems: A Programmer's Perspective (cap. 7.1)

Programação em Sistemas Computacionais

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

ISEL, ADEETC, LEIC

Agenda

- Processo de geração de um executável:

    1. Pré processamento
    2. Compilação
    3. Tradução
    4. Ligação

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;
}
```

Fases dentro do processo de geração de um executável

Processo de geração de um executável

- Comando usado para gerar o executável até à data:

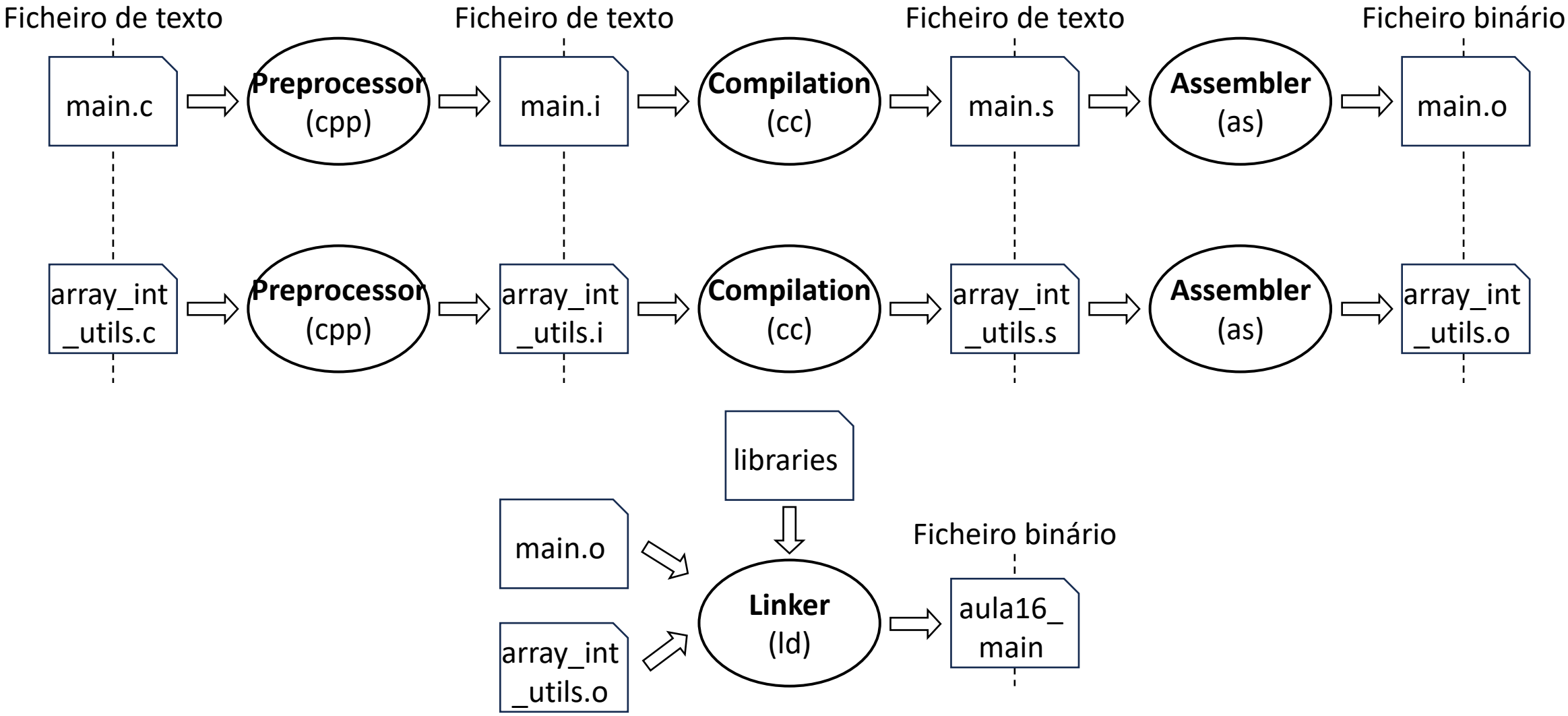
```
$ gcc -Wall -o aula16_main main.c array_int_utils.c
```

- A opção `-save-temps` deixa rasto de todos os ficheiros intermédios

```
$ gcc -Wall -save-temps -o aula16_main main.c array_int_utils.c
```

Fases (por ordem)	Ficheiros de entrada	Ficheiros de saída	Tipo de ficheiros produzidos
1. Pré processamento	`main.c` `array_int_utils.c`	`main.i` `array_int_utils.i`	Ficheiros pré processados
2. Compilação	`main.i` `array_int_utils.i`	`main.s` `array_int_utils.s`	Ficheiros em *assembly*
3. Tradução	`main.s` `array_int_utils.s`	`main.o` `array_int_utils.o`	Ficheiros objeto relocáveis
4. Ligação	`main.o` `array_int_utils.o`	`aula16_main`	Ficheiro objeto executável

Compilação separada – porquê?
Processo de geração de um executável



1. Pré processamento

Processo de geração de um executável

- Exemplos de diretivas de pré processamento
  - ***#include***: inclusão de ficheiro
  - ***#define***: definição de símbolo ou macro
  - ***#undef***: indefinição de símbolo ou macro
  - ***#ifdef*/*#ifndef***, ***#if*/*#elif*/*#else***, ***#endif***: compilação condicional
- Utilitário **`cpp`** para pré processar – por omissão o resultado é escrito no *standard output*

```
$ cpp array_int_utils.c > array_int_utils.i
```

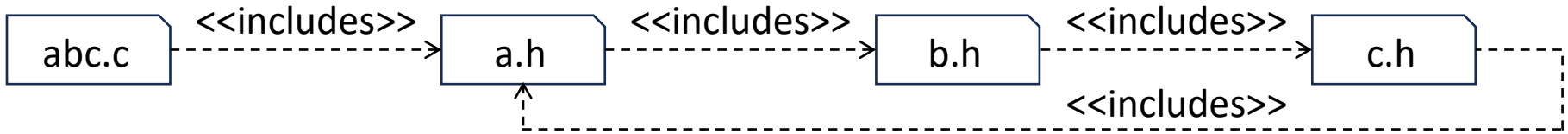
- Opção **`-E`** para parar após pré processamento usando o **`gcc`**

```
$ gcc -E array_int_utils.c -o array_int_utils.i
```

Problema de inclusão recursiva e solução

1. Pré processamento

- Problema de um grafo fechado de inclusões




```
$ gcc -E abc.c -o abc.i
```

```
a.h:1:15: error: #include nested too deeply
    1 | #include "b.h"
```

- Solução com inclusão condicional

a_no_rec.h

```
#ifndef _A_NO_REC_H
#define _A_NO_REC_H

#include "b_no_rec.h"

#endif/*_A_NO_REC_H*/
```

b_no_rec.h

```
#ifndef _B_NO_REC_H
#define _B_NO_REC_H

#include "c_no_rec.h"

#endif/*_B_NO_REC_H*/
```

c_no_rec.h

```
#ifndef _C_NO_REC_H
#define _C_NO_REC_H

#include "a_no_rec.h"

#endif/*_C_NO_REC_H*/
```

Definição de macros

1. Pré processamento

- A diretiva **`#define`** permite definir constantes e permite, igualmente, definir macros (espécie de funções sem que exista chamada mas antes substituição)

```
#define ARRAY_SIZE(a) (sizeof(a)/sizeof(a[0]))
```

- As boas práticas determinam
  - que os parâmetros da macro apareçam na definição da macro sempre envolvidos entre parêntesis
  - que quando a definição da macro corresponda a uma expressão, a própria definição esteja envolvida entre parêntesis
- O operador **#** aplicado a um argumento converto-o numa *string* (*stringizing operator*)
- O operador **##** concatena argumento a *token* ou concatena argumentos formando um novo *token* (*token-pasting operator*)

Exemplo com macros

Definição de macros

macros_main.c

```c
#define INC_BAD(x)        x+1
#define INC_GOOD(x)       ((x)+1)
#define DOUBLE_BAD(x)     (x * x)
#define DOUBLE_GOOD(x)    ((x) * (x))
#define PRINT_EXP(exp1, exp2)    \
    printf(#exp1" = %d\n", c##exp2)

int main() {
    int c1 = INC_BAD(2) * 3;
    int c2 = INC_GOOD(2) * 3;
    int c3 = DOUBLE_BAD(c2 + 1);
    int c4 = DOUBLE_GOOD(c2 + 1);
    PRINT_EXP(c1, 1);
    PRINT_EXP(c2, 2);
    PRINT_EXP(c3, 3);
    PRINT_EXP(c4, 4);
    return 0;
}
```

macros_main.i

```c
int main() {
    int c1 = 2 +1 * 3;
    int c2 = ((2)+1) * 3;
    int c3 = (c2 + 1 * c2 + 1);
    int c4 = ((c2 + 1) * (c2 + 1));
    printf("c1"" = %d\n", c1);
    printf("c2"" = %d\n", c2);
    printf("c3"" = %d\n", c3);
    printf("c4"" = %d\n", c4);
    return 0;
}
```

```
c1 = 5
c2 = 9
c3 = 19
c4 = 100
```

2. Compilação

Processo de geração de um executável

- O utilitário **`cc`** é o compilador de C do *toolchain* da GNU
- A opção **`-S`** termina o processo após gerar o ficheiro em *assembly*

```
$ cc array_int_utils.i -S
```

- Por omissão, o ficheiro produzido tem o mesmo nome do ficheiro fonte (`array_int_utils.s`)
- O utilitário **`cc`** é um link para o utilitário **`gcc`**

```
$ gcc array_int_utils.i -S
```

- Pode-se gerar o ficheiro em *assembly* diretamente a partir do código fonte

```
$ gcc array_int_utils.c -S
```

3. Tradução

Processo de geração de um executável

- O utilitário **`as`** é o compilador de *assembly* da GNU e produz um ficheiro objeto relocável

```
$ as array_int_utils.s –o array_int_utils.o
```

- Usando o **`gcc`**, a opção **`–c`** termina o processo após tradução. Por omissão, o nome do ficheiro de saída é igual ao nome do ficheiro de entrada

```
$ gcc array_int_utils.s –c
```

- Por norma, para compilação separada, gera-se o ficheiro objeto relocável a partir do código fonte. Usa-se, igualmente a opção **`-c`**

```
$ gcc array_int_utils.c –c
```

4. Ligação

Processo de geração de um executável

- O utilitário **`ld`** é o *linker* e produz o executável ligando os ficheiros objetos relocáveis e bibliotecas

```
$ ld main.o array_int_utils.o
-dynamic-linker /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
/usr/lib/x86_64-linux-gnu/crt1.o
/usr/lib/x86_64-linux-gnu/crti.o
/usr/lib/x86_64-linux-gnu/crtn.o -lc -o aula16_main
```

- Desaconselhável devido à promiscuidade das opções necessárias para conseguir gerar o executável; preferível usar o utilitário **`gcc`**

```
$ gcc main.o array_int_utils.o -o aula16_main
```

- A sua operação resume-se a duas tarefas principais
  - Resolução de símbolos
  - Relocação
- Curiosidade: usar a opção **`-v`** para ver as opções usadas pelo **`gcc`** no processo de ligação

Resumo

- Geração do executável por etapas com base nos dois módulos **`main.c`** e **`array_int_utils.c`**
  - Qualquer etapa pode ser realizada diretamente a partir dos módulos em C

- Pré processamento

```
$ gcc –E -o main.i main.c
$ gcc –E -o array_int_utils.i array_int_utils.c
```

- Compilação

```
$ gcc -S main.i array_int_utils.i
```

- Tradução

```
$ gcc -c main.s array_int_utils.s
```

- Ligação

```
$ gcc -o aula16_main main.o array_int_utils.o
```