PROGRAMAÇÃO E ALGORITMOS (LEII)

Universidade da Beira Interior, Departamento de Informática
Hugo Pedro Proença, 2016/2017

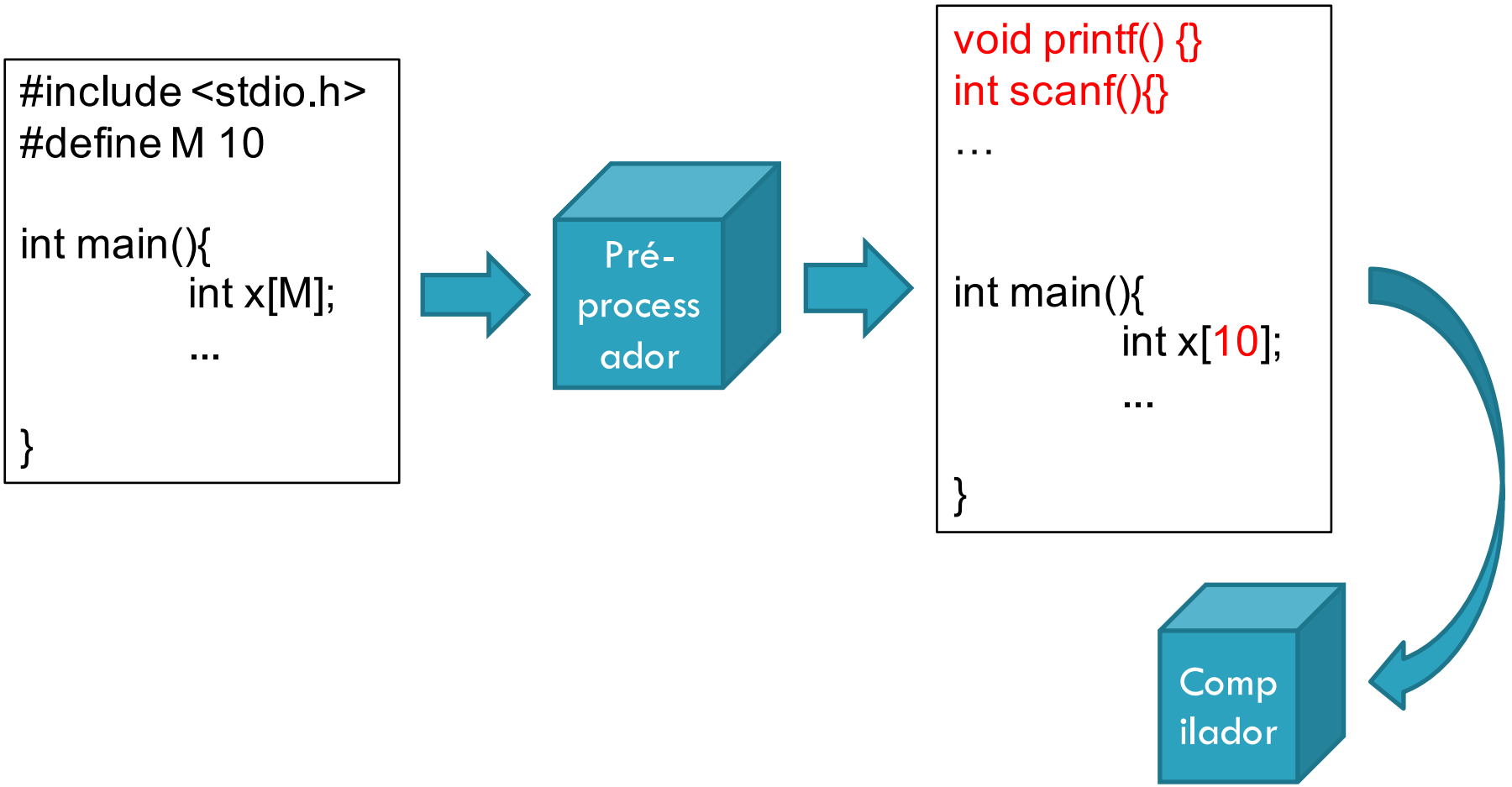
Resumo

- Pré-processador
  - Directivas
- Macros
- Exemplos
- Exercícios

Pré-Processador

- O pré-processador da linguagem C é um processo automaticamente executado pelo compilador da linguagem antes do processo de compilação. Em termos gerais, disponibiliza quatro tipos de funcionalidades:
  - Inclusão de ficheiros cabeçalho (header)
  - Definição de Macros
  - Compilação condicional
  - Controlo de código

Pré-Processador



Pré-Processador: Transformações Gerais

- As directivas do pré-processador são normalmente indicadas pelo caractér "#". No entanto, existem 3 transformações feitas ainda antes da procura de directivas:
  - Todos os comentários são substituídos por espaços em branco.
  - Mudanças de linha através do caractér "\" são removidas.
  - As macros pré-definidas são expandidas.

Pré-Processador:Directivas

- Inclusão
  - Os diferentes ficheiros de código-fonte são incluídos no programa a compilar através da directiva "#include".
  - Pode incluir tanto ficheiros criados pelo programador, como bibliotecas nativas da linguagem.
- #include <ficheiro>
  - Procura por "ficheiro" numa lista de directorias pré-definidas. É normalmente utilizado para incluir as bibliotecas de sistema.
- #include "ficheiro"
  - Procura por "ficheiro" na directoria actual. É utilizado para incluir bibliotecas de funções criadas pelo programador.

Pré-Processador: #include

- Exemplo:

F1.h

```
void funcao1(){

}
```

F2.h

```
void funcao2(){

}
```

F3.h

```
typedef struct{
        int bi;
}Pessoa;
```

```
#include "F3.h"
#include "F1.h"
#include "F2.h"

int main(){

}
```

```
typedef struct{
        int bi;
}Pessoa;
void funcao1(){

}
void funcao2(){

}
int main(){

}
```

Pré-Processador: Definição

- A directiva "#define" serve para criar "*alias*" para uma determinada sequência de caracteres.
  - Quero que "ABC DDEEE" seja designado por "AB".
- É bastante comum para definir constantes.
- Exemplo:

```
#define TOTAL 100

int main(){
    int x[TOTAL];
}
```

Pré-Processador: Condicional

- É comum que diferentes ficheiros necessitem de incluir bibliotecas comuns.
  - Suponha-se que um nosso programa necessitava de funções das bibliotecas "bib1.h" e "bib2.h".
  - No entanto, por hipótese, tanto "bib1.h" como "bib2.h" inluíam "bib3.h".
  - Ao fazer a inclusão de "bib1.h" e "bib2.h" estava-se a incluir uma biblioteca repetida.

```
#include "bib1.h"
#include "bib2.h"
int main(){
    ...
}
```

Pré-Processador: Condições

- A solução será definir constantes identificadoras no ínicio de cada biblioteca e posteriormante só fazer a sua inclusão caso a respectiva constante não esteja definida.

  Exemplo:

  ```
  #define __BIBLIOTECA1
  ```

- Inclusão condicional:

  ```
  #ifndef __BIBLIOTECA1
      #include “bib1.h”
  #endif
  ```

Pré-Processador: Macros

- Uma macro pode ser entendida como um alias (#define é uma macro). Na sua forma mais simples, servem para substituir blocos de código extenso por abreviaturas:

```
#define ST1 “Esta é uma string escrita no programa”


int main(){
    printf(“%s”,ST1);
}
```

Pré-Processador

- A utilização de macros é especialmente vantajosa quando elas permitem a entrada de argumentos. Este tipo de macros designa-se por "funções macro". Para definir uma função macro utiliza-se também a directiva "#define", seguida imediatamente dos parêntesis e da lista de parâmetros.
- Exemplo:
  - `#define min(X, Y)  ((X) < (Y) ? (X) : (Y))`

Pré-Processador

- Para utilizar a macro no programa, basta colocar o seu nome e os respectivos parâmetros, semelhante à utilização de 1 função:
  - "min(a,b);" vai utilizar a macro "min" e fazer a subtituição de "X-->a" e "Y-->b".
  - de igual forma, `min (x + 28, *p)' expande-se para:
    - ((x + 28) < (*p) ? (x + 28) : (*p))

Pré-Processador

- Deve-se ter em atenção que o uso de parêntesis imediatamente a seguir ao nome da macro é o que determina se esta tem ou não parâmetros.
- Por exemplo, a seguinte macro recebe 1 argumento (x) e substitui-o por -1/x:
  - #define BAR(x) - 1 / (x)
- No entanto, a seguinte macro apenas substitui a ocorrência de "BAR" pela expressão "(x) - 1/(x)“
  - #define BAR (x) - 1 / (x)
- A maior vantagem para o uso de macros é o desempenho, em comparação com a utilização de funções. Não são gastos recursos a passar o controlo de execução para a nova função.