

Linguagem C

Tipos de Dados

void; escalares; sizeof


Vectores

Strings em C

Estruturas

Introdução ao pré-processador

Funções void

 **void** pode ser usado em lugar de um tipo, para indicar a ausência de valor

 Funções que não retornam valores (void) são por vezes designadas por **procedimentos**

 Exemplo – tratamento de erros:

```
void error( char msg[], int fatal )
{    // escrever mensagem de erro
    printf("Error: %s\n", msg );
    if( fatal ) exit(1); // terminar programa
    return;
}
```

Tipos de dados escalares (revisão)



Essencialmente os mesmos que em Java, no entanto:

- ✓ Em C não existe **boolean**
 - ✓ temos que usar inteiros: 0 significa falso; qualquer outro valor significa verdade
- ✓ **char** em C corresponde a **byte** em Java
 - ✓ Os caracteres correspondem à representação ASCII dos valores armazenados
 - ✓ Por exemplo: '0' = 48, 'A' = 65, '\n' = 10
 - ✓ Logo podemos misturar livremente caracteres e inteiros em expressões e atribuir inteiros a caracteres e vice-versa

Constantes

Caracteres — delimitados por “

- ✓ 'a', 'b', etc
- ✓ '\n' — mudança de linha
- ✓ '\0' — carácter nulo - o mesmo que 0
- ✓ '\\' — a barra → \

Representação de constantes numéricas:

- ✓ Inteiros em decimal: 0, 11, -23
- ✓ Inteiros em hexadecimal: 0xfe, 0x00, 0x2af0
- ✓ Inteiros em octal: 0644, 0755
- ✓ Inteiros em binário: não existe
- ✓ Reais: 0.0, 3.14, -2.5
- ✓ Reais em notação científica: 2e5, 1.3e-4, -1.0

Modificadores

 **signed** e **unsigned** – podemos especificar se os números têm ou não sinal

```
signed int c = -1;
```

```
unsigned int d;
```

 **long** e **short** - podemos requerer mais ou menos precisão nos inteiros

```
long int a;
```

```
short int b;
```

 Pode-se omitir o **int**

```
long a; unsigned b;    // o mesmo que:
```

```
long int a; unsigned int b;
```

 **double** - é o mesmo que **long float**

Exemplos do tamanho dos números

Inteiros - gcc - Linux i386 (IA-32)

- ✓ short - 16 bits
- ✓ int - 32 bits
- ✓ long - 32 bits

nbits (short) <=
 <= nbits (int) <=
 <= nbits (long)


Inteiros - gcc - Linux x86-64 (AMD64/Intel64)

- ✓ short - 16 bits
- ✓ int - 32 bits
- ✓ long - 64 bits


Reais

- ✓ float - 32 bits - precisão simples IEEE 754
- ✓ double - 64 bits - precisão dupla IEEE 754
- ✓ long double - 80 bits - extensão a 80 bits do IEEE 754

sizeof

 **sizeof** permite-nos saber o número de bytes ocupado por uma variável

 Também pode ser usado para tipos

 Por definição, temos sempre:

`sizeof(char) = 1`

 Podemos usar **sizeof** nos nossos programas para saber o tamanho dos dados que estamos a usar!

✓ Saber o tamanho da palavra da máquina física

```
if (sizeof(long) == 8)
```

```
    //estamos num sistema de 64 bits!
```

Precisão dos inteiros — Exemplos


 **int** - gcc - Linux i386:

- ✓ 32 bits
- ✓ `sizeof(int) = 4`
- ✓ Valor máximo: $2^{31}-1$
- ✓ Valor mínimo: -2^{31}

 **unsigned long** - gcc - no Linux x86-64

- ✓ 64 bits
- ✓ `sizeof(unsigned long) = 8`
- ✓ Valor máximo: $2^{64}-1$
- ✓ Valor mínimo: 0

Vectores em C

 Declaração de vectores:


```
int d[10]; // vector de 10 inteiros
           // primeiro elemento: d[0]
           // último elemento:   d[9]
char c[100]; // vector de 100 caracteres
```

 Os índices começam sempre em 0!

 Exemplo – somar todos os elementos de d:

```
int s, i;
s = 0;
for( i = 0; i < 10; i++ )
    s += d[i];
```

Vectores em C (cont.)

 Não há verificação dos limites do vector

- ✓ No Java é emitida uma excepção sempre que se acede a uma posição fora dos limites do vector
- ✓ No C não existe tal verificação
- ✓ Implica consequências imprevisíveis quando o programa acede a uma posição inválida do vector!

 Não existe o **length** do Java!

- ✓ Temos que usar outras formas de saber o tamanho

 Não se podem atribuir vectores

- ✓ É preciso copiar os elementos um a um

Vectores em C (cont)



As funções não podem retornar vectores

- ✓ Será preciso retornar o endereço do primeiro elemento do vector, ou seja, onde o vector começa
- ✓ Falaremos disso na próxima aula



Os vectores são sempre passados por “referência” para as funções

- ✓ As alterações realizadas dentro da função reflectem-se no vector original
- ✓ Equivalente aos objectos em Java

Strings

- 🖥️ Em C não existe o tipo de dados *string*!
- 🖥️ Temos que usar vectores de caracteres
- 🖥️ Uma string é um vector que contém os caracteres da string, seguido do carácter '\0'
- 🖥️ Tudo o que vier a seguir ao terminador não faz parte da string – é considerado “lixo”
- 🖥️ As constantes de string em C são delimitadas por "..."
 - ✓ diferente da ' dos caracteres!

🖥️ "abc" - corresponde a:

'a'	'b'	'c'	'\0'
-----	-----	-----	------

🖥️ "\n\n" - corresponde a:

'\n'	'\n'	'\0'
------	------	------

Strings — Funções de biblioteca

 **strlen(s)** - comprimento útil de uma string


- ✓ Equivalente, grosso modo, ao **length** do Java
- ✓ Não conta com o terminador ('\0')

 **strcmp(s1, s2)** — comparar s1 com s2 — retorna:


- ✓ < 0 se $s1 < s2$ — por exemplo — `strcmp("cat", "dog")`
- ✓ 0 se $s1 == s2$ — por exemplo — `strcmp("cat", "cat")`
- ✓ > 0 se $s1 > s2$ — por exemplo — `strcmp("dog", "cat")`
- ✓ **Cuidado!** O compilador permite usar `==`, `!=`, `<=`, etc mas não funciona!
 - ✓ O mesmo se passa no Java, têm de usar o *equals*

```
if ( strcmp(nome, "Tiago") == 0)
    // estamos a tratar do Tiago...
```

Strings — Funções de biblioteca (cont)

 **strcpy(s1,s2)** - copia s2 para s1

- ✓ Podemos ler como "s1 = s2"
- ✓ É necessário usar esta função para copiar strings, porque em C não se podem atribuir vectores — é preciso copiar os valores dos elementos!
- ✓ **Cuidado!**
 - ✓ Se s1 não tiver posições suficientes para todos os caracteres de s2 (incluindo o terminador '\0'), strcpy vai aceder a posições fora do vector s1!
 - ✓ As consequências são imprevisíveis!

 Os protótipos destas funções estão definidos no ficheiro de cabeçalho **<string.h>**

Strings – Inicialização

 Podemos inicializar uma string quando declaramos o respectivo vector:

```
char s[100] = "ola\n";
```

✓ `strlen(s)` → 4

✓ `sizeof(s)` → 100


✓ `s[0]` → 'o'


✓ `s[4]` → '\0'

✓ As restantes 95 posições são preenchidas com valores indefinidos ("lixo")

 Temos que prever espaço suficiente para se depois quisermos copiar strings maiores para esta variável!

Strings – inicialização (2)

 Podemos omitir o tamanho do vector – ele é criado com o tamanho mínimo necessário

 O tamanho será suficiente para todos os caracteres da string inicial

✓ Incluindo o terminador ('\\0')

```
char ola[] = "ola\\n";
```

✓ `strlen(ola)` → 4;

✓ `sizeof(ola)` → 5;

```
strcpy(ola, "xpto123");
```

 Problema! "xpto123" não vai caber em ola!

✓ resultados imprevisíveis!

Strings em C – Exemplo

```
#include <stdio.h>
```




```
#include <string.h>
```

```
void cifra_cesar( char s[], int n ) {  
    int i;  
    for ( i = 0; i < strlen(s); i++ )  
        if ( s[i] >= 'a' && s[i] <= 'z' ) {  
            s[i] += n;  
            if ( s[i] > 'z' )  
                s[i] -= 26; // 26 = 'z'-'a'+1  
        }  
}
```

Strings em C - Exemplo (cont)

```
int main() {  
    char s[100];  
    int flag = 1;  
    while(flag) {  
        printf("Introduza a string para cifrar ou fim\n");  
        scanf("%s", s); // reparar que não se faz &s  
        if(strcmp(s, "fim") == 0)  
            flag = 0;  
        else {  
            cifra_cesar(s, 3);  
            printf("%s\n", s);  
        }  
    }  
    return 0;  
}
```

Estruturas

-  Uma estrutura (**struct**) em C corresponde, em grosso modo, a uma **classe sem métodos** em Java
-  Permite agregar dados de diferentes tipos
-  Os **campos** da estrutura são acedidos como os membros de uma classe

```
struct data { int dia, mes, ano; };  
struct funcionario {  
    unsigned long num_interno;  
    char nome[100];  
    struct data data_admissao;  
};
```

Estruturas - Exemplo de uso

```
struct data d;  
struct funcionario f;  
  
d.dia = 5; d.mes = 10; d.ano = 1910;  
  
f.num_interno = 1024;  
f.data_admissao = d;  
strcpy(f.nome, "Belmiro");  
  
f.data_admissao.dia = 6;  
  
d = f.data_admissao;
```

sizeof — Vectors e estruturas

Tamanho de um vector

- ✓ Número de elementos * tamanho do elemento

```
int d[100];
```

- ✓ `sizeof(d[0]) == sizeof(int)`

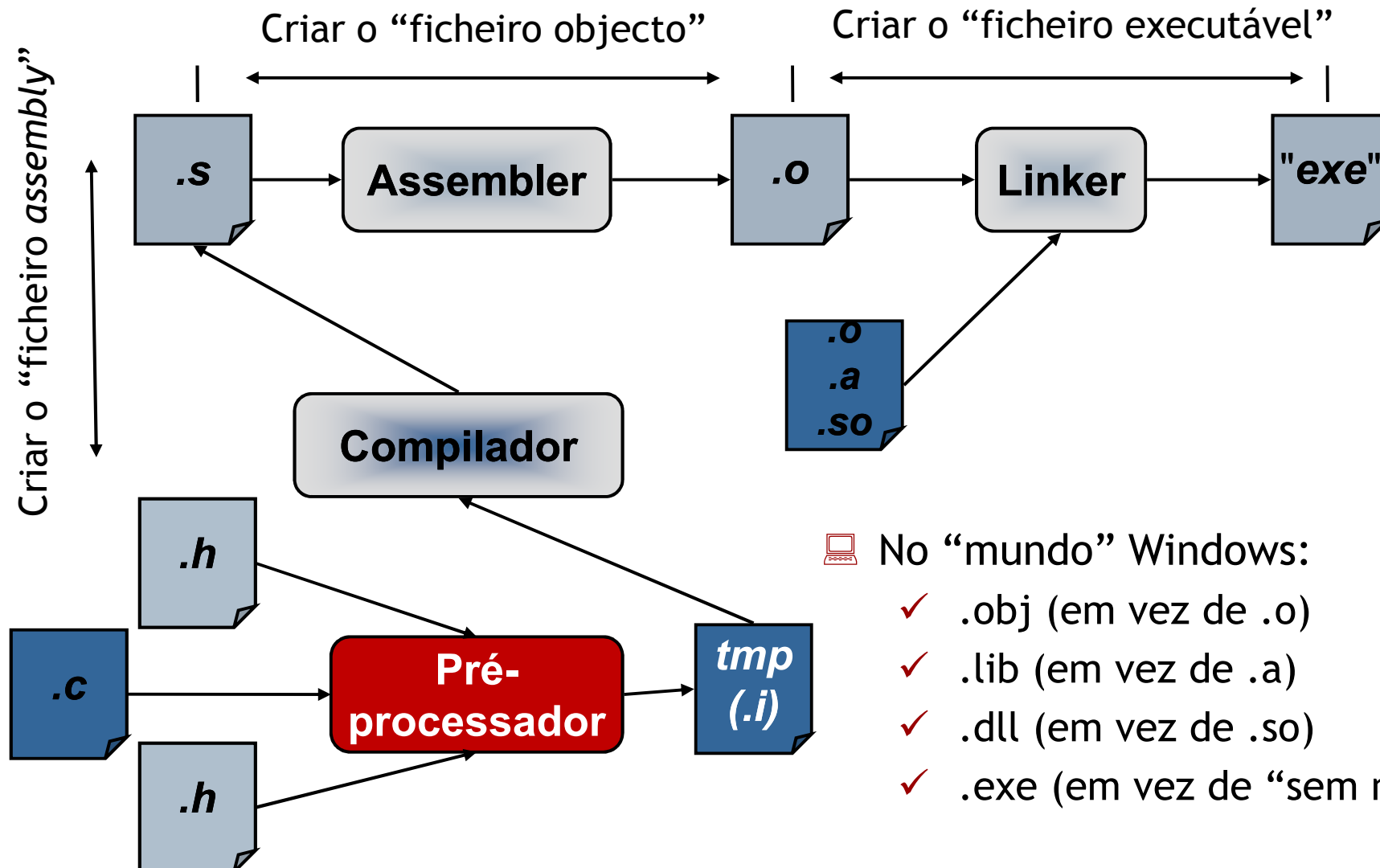
- ✓ `sizeof(d) == 100 * sizeof(int)`

O tamanho de uma estrutura é maior ou igual que a soma dos tamanhos dos seus campos


- ✓ `sizeof(struct data) ≥ 3 * sizeof(int)`

- ✓ `sizeof(struct funcionario) ≥
sizeof(unsigned long) + 100 +
sizeof(struct data)`

Compilar um programa C em Linux



Pré-processador — Directivas

 Fase de substituição de texto executada antes da compilação

 Usa directivas em linhas iniciadas por #

 **#include** - incluir outros ficheiros no programa

✓ Por exemplo, cabeçalhos de bibliotecas:

```
#include <string.h>
```

```
#include <stdlib.h>
```

 **#define** - substituição de texto

✓ Muito usada para definir constantes

```
#define TRUE 1
```

```
#define FALSE 0
```

Programa antes de pré-processado

```
#include <stdio.h>
// colocar sempre números e expressões entre parêntesis
#define PI (3.1416)

float graus_para_rad(int x) {
    return x*PI/180;
}

int main() {
    int x = 10;
    float res = graus_para_rad(x);
    printf("O valor em radianos de %d e' %f\n", x, res);
    return 0;
}
```


Programa depois de pré-processado

```
...  
int printf(const char format, ...);  
...  
  
float graus_para_rad(int x) {  
    return x*(3.1416)/180; ←  
}  
  
int main() {  
    int x = 10;  
    float res = graus_para_rad(x);  
    printf("O valor em radianos de %d e' %f\n", x, res);  
    return 0;  
}
```

Código do
ficheiro stdio.h.
Note que não
contém a
implementação
do printf. Só o
seu protótipo

Susbtituição
de PI
pelo valor
definido