Arquitetura de Computadores

MIEI – 2017/18
DI-FCT/UNL
Aula 3

Em C

- Exemplo **`printf(`***`format, X,...`***`)`**
  - *Format* – **é apenas uma string** que define o que escrever e como apresentar os restantes argumentos

  Exemplo: `printf("result: %d, %x, %f\n", 12, 12, 12.0);`
  escreve: **`result: 12, c, 12.00000`**↵
  - Exemplos de outros formatos:

  `%ld` = long em base 10 `%hd` = short em base 10
  `%u` = unsigned int base 10 `%lu` = unsigned long base 10
  `%e` = double em notação científica
- Em Java também existe `printf()`

Printf faz o indicado pela formatação

- Exemplo:

```
short i = -1;
unsigned short j = i;
int x = j+1;
printf("i=%hd, j=%hu, x=%d", i, j, x);
printf("i=%u, j=%d, x=%f ?=%d", i, j, x);
```

- O compilador pode avisar, mas produz um executável
- Resultado? (pode variar entre arquiteturas....)

```
i=-1, j=65535, x=65536
i=4294967295 j=65535 x=0.00000 ?=134518384
```


C vs Java

- Construções de controlo:

```
if (exp) {  inst;
} else {  inst;
}

while ( exp ) {
  inst;
}

for ( inst; exp; inst ) {
  inst;
}
```

```
switch (expr) {
  case val: inst; break;
  ....
  default: inst;
}

do {
   inst;
} while (exp);
```


C vs Java

- Comentários:

```
/* bloco
 */

// linha
```

- Blocos de código e escopo das declarações:

```
{ declarações;
  instruções;
}
```


C tem pre-processador

- Diretivas iniciadas por #
  - `#define XPTO 234` - no ficheiro, substitui XPTO por 234
  - Permite pseudo-constantes, e mais...

  `#include "file.h"` - inclui o file.h da diretoria corrente
  `#include <file.h>` - inclui o file.h, da diretoria standard
  - Permite introduzir as definições de funções e tipos fora do nosso ficheiro como das bibliotecas (interfaces)
  - Também chamados ficheiros de headers

Exemplo revisitado (hello.c)

- Escrever 5 linhas com "hello world":

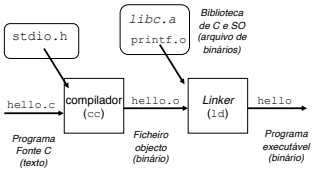
```
#include <stdio.h>
#define NVEZES  5

int main( ) {
    // escreve NVEZES
    for (int i=0; i<NVEZES; i++)
        printf("hello world\n");
    return 0;
}
```


Processo de compilação de C

- compilação do programa:

```
cc –o hello hello.c
```



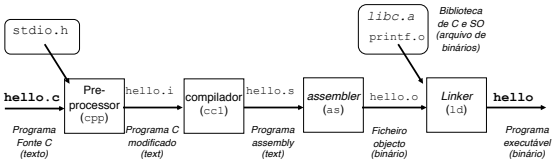
- Existem erros e *warnings*
  - de compilação e de ligação

Processo de compilação de C

- compilação do programa:

```
cc –o hello hello.c
```

- Os compiladores podem ter mais passos intermédios:



Funções C vs métodos Java

- As funções são semelhante aos métodos Java, mas:
  - Não pertencem a objetos. Não existe `this`
  - São como os métodos `static`
  - Não há polimorfismo (um nome = uma função)

Parâmetros de funções (Java e C)

- Os argumentos são passados por cópia do valor

```
int fun1( int x ) {
   x=x+1;
   return x;
}
...
int y=3;
int z=fun1( y );
```

- uma cópia do valor em y é passada para x de fun1
- y não é alterado!

Exemplo

- Exemplo de programa:

```
#include <stdio.h>
int proxpar( int x ) {
    if ( x%2 == 0 ) return x+2;
    else return x+1;
}
int main( ) {
    int par = proxpar( 3 );
    printf("Proximo par: %d\n", par);
    return 0;
}
```

Objetos como parâmetros no Java

- Os objetos em Java são passados por referência

```
class A { public int i; }
int fun1( int x, A z ) {
    x=x+1;
    z.i = z.i+1;
    return x;
}
```

```
...
int y1=1;
A y2=new A();
y2.i=1;
int w=fun1(y1, y2);
```

  - uma cópia do valor em y1 é passada para x de fun1
  - A referência de y2 é passada para z de fun1
  - y1 não é alterado por fun1!
  - y2 é alterado por fun1!

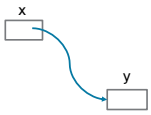
Apontadores

- Apontador = ponteiro = referência de memória = *pointer*
- Declaração de variável apontador:
  - *Tipo_apontado* ***** *var_ptr*;
- Obter apontador para uma variável:
  - *var_ptr* = **&***var*
- Obter valor apontado (valor na variável original):
  - *x* = ***** *var_ptr*

Parâmetros apontadores em C

- Em C podemos passar referências (apontadores):

```
    int fun1( int *x ) {
        *x=*x+1;
        return *x;
    }
    ...
int y=3;
int z=fun1( &y );   //passamos a referência de y
```



- O endereço de y é passado para x de fun1
- y é alterado via apontador x!

Arrays em C - vetores

- Os arrays consistem em sequências contíguas de elementos do mesmo tipo.
- Sintaxe (variantes):

```
int A[3];   // não iniciado
A[0]=1; A[1]=2; A[2]=3;

int A[3] = { 1, 2, 3};
int A[] = { 1, 2, 3};
```

- Tabela de símbolos do compilador:
  'A' representa o início do array (seu endereço)

Arrays em C - matrizes

- As matrizes são dispostas por linhas.
- Sintaxe (variantes):

```
short A[3][2];
A[0][0]=1; A[0][1]=2;
A[1][0]=3; A[1][1]=4;
A[2][0]=5; A[2][1]=6;

short A[][]={ {1, 2}, {3, 4}, {5, 6} };
```

- Tabela de símbolos do compilador:
  'A' representa o início do array (seu endereço)

Arrays em C e apontadores

- O nome do array vale por um apontador constante:

```
int v[3];     // v tem tipo int*

v[0] = 1   ⇔   *v = 1

v[2] = 5;  ⇔   *(v+2) = 5;
```

Criar variáveis dinamicamente

- Criando dinamicamente variáveis:

```
void * malloc( int size )
```

- Criar um vetor de 3 inteiros:

```
int *v = (int*)malloc(3*sizeof(int));
```

- Usando com um vetor:

```
v[2] = 5;
```