



Programação em C

Prof. Inês Dutra

ines@dcc.fc.up.pt

Gab: 1.31



Sintaxe

- Básico: estrutura de programa, sintaxe
- Interface com linha de comando
- Preprocessamento e compilação
- Makefiles
- Ponteiros e estruturas
- Bibliotecas, ficheiros “include” e funções
- Referências



Básico

- Estrutura de um programa:

```
main()  
{  
    <corpo do programa>;  
}
```



Básico

- Exemplo simples

```
#include <stdio.h>
main()
{
    printf("Hello, world!\n");
    return 0;
}
```



Interface com linha de comando

```
#include <stdio.h>
main(int argc, char **argv)
{
    if (argc > 0)
        printf("Hello %s!\n", argv[1]);
    else
        printf("Hello World!\n");
    return 0;
}
```

```
$ ./prog Artur
Hello Artur!
$
```



Preprocessamento e compilação

- Preprocessador: cpp
 - Compilador: cc (gcc)
 - `man gcc`
 - `gcc prog.c -o prog`
- ```
$./prog
Hello World!
$
```



# Makefiles

- `make -f <makefile>`
- `make`
  - default filename: `makefile` or `Makefile`



# Makefiles

```
CC=gcc
all: hello
hello: main.o factorial.o hello.o
 $(CC) main.o factorial.o hello.o -o hello
main.o: main.c
 $(CC) -c main.c
factorial.o: factorial.c
 $(CC) -c factorial.c
hello.o: hello.c
 $(CC) -c hello.c
clean:
 rm -rf *.o hello
```





# Makefiles

```
CC=gcc
```

```
CFLAGS= -c -Wall
```

```
LDFLAGS=
```

```
SOURCES=main.c hello.c factorial.c
```

```
OBJECTS=$(SOURCES:.c=.o)
```

```
EXECUTABLE=hello
```

```
all: $(SOURCES) $(EXECUTABLE)
```

```
$(EXECUTABLE): $(OBJECTS)
```

```
 $(CC) $(LDFLAGS) $(OBJECTS) -o $@
```

```
.c.o:
```

```
 $(CC) $(CFLAGS) $< -o $@
```



# Comandos

- Atribuições:  $x=2$
- Condições:

```
if (<condição>) {
 <ramo true>
} else {
 <ramo false>
}
```



# Comandos

## ■ Iterações

```
for (var=<valor inicial>; var < limite; var++) {
 <corpo do for>
}
```

```
while (<condição>) {
 <corpo do while>
}
```



# Tipos de dados

- int, long int, float, double, char

```
int i;
```

```
float x = 0.5;
```

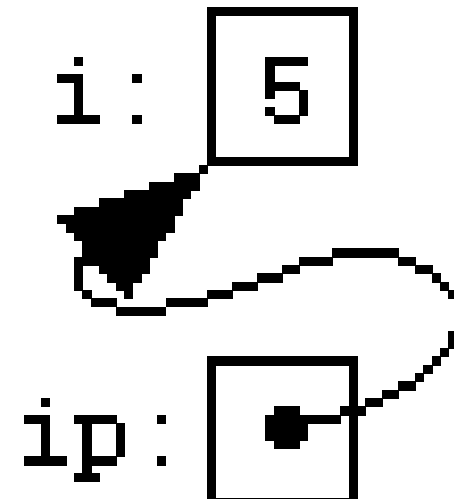
```
char nome[20]; // vetor com max 20 cars
```

- Arrays, pointers, strings, structs

# Tipos de dados

## ■ Pointers

```
int *ip;
int i = 5;
ip = &i;
printf("%d\n", *ip);
```



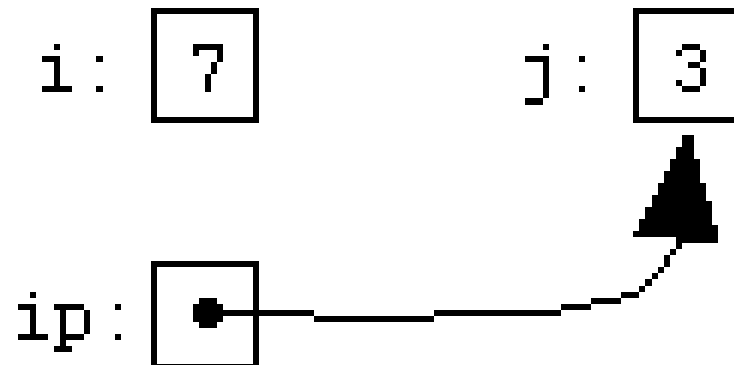
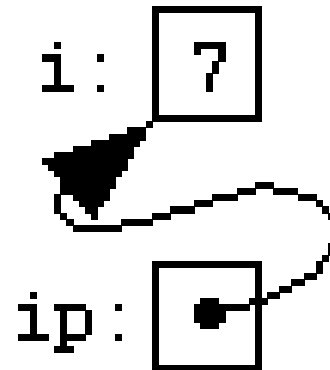
# Tipos de dados

## ■ Pointers

```
*ip = 7;
```

```
int j = 3;
```

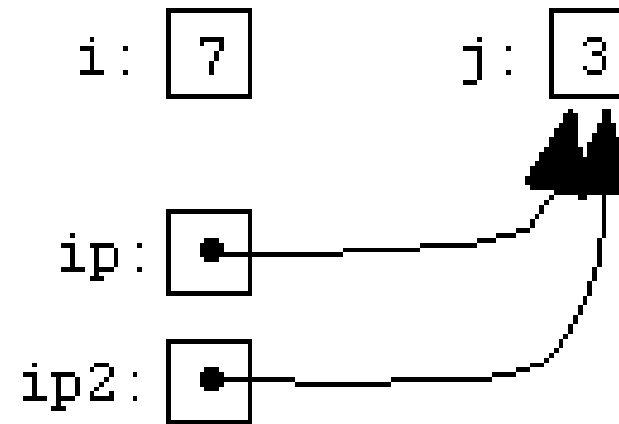
```
ip = &j;
```



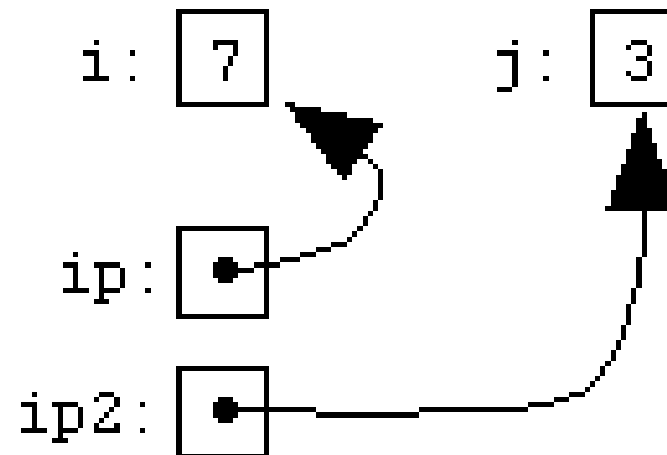
# Tipos de dados

## ■ Pointers

```
int *ip2;
ip2 = ip;
```



```
ip = &i;
```



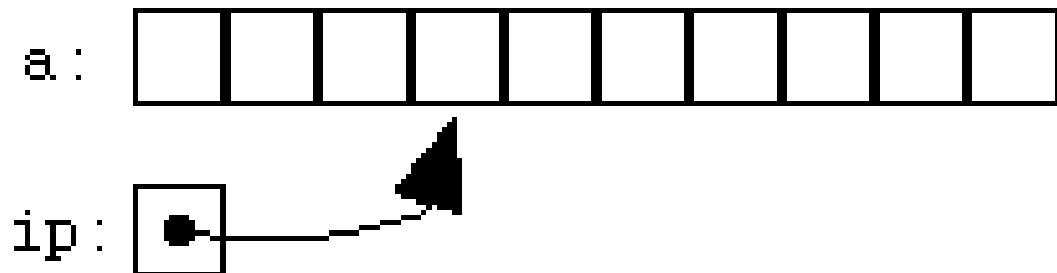
# Tipos de dados

## ■ Pointers

```
int *ip;
```

```
int a[10];
```

```
ip = &a[3];
```

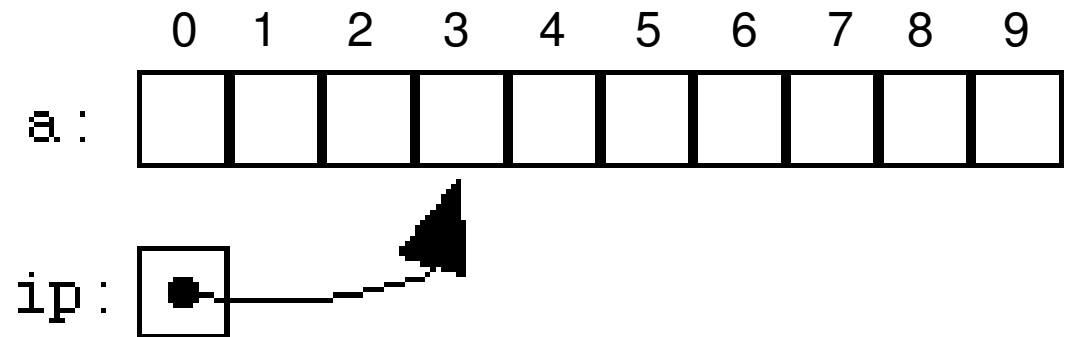




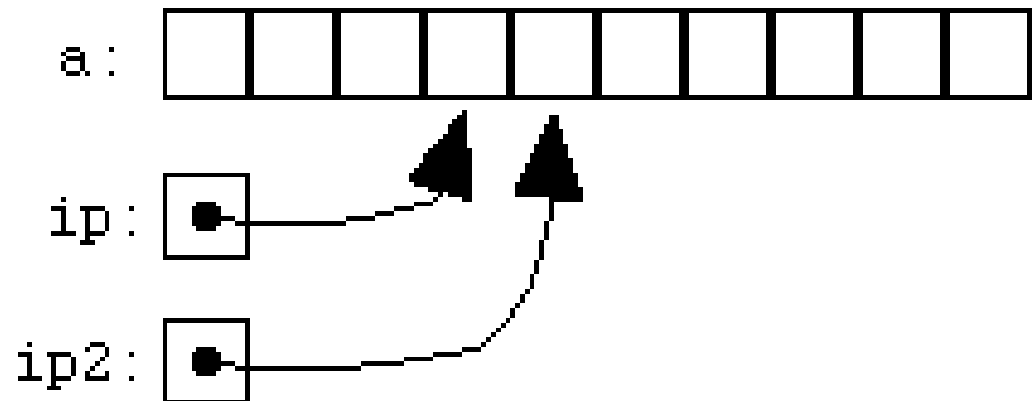
# Tipos de dados

## ■ Pointers

```
int *ip;
int a[10];
ip = &a[3];
```



```
int *ip2;
ip2 = ip + 1;
```





# Tipos de dados: pointers

- Exemplo: comparação de strings

```
char *p1 = &str1[0], *p2 = &str2[0];
while(1) {
 if(*p1 != *p2) return *p1 - *p2;
 if(*p1 == '\0' || *p2 == '\0')
 return 0;
 p1++;
 p2++;
}
```



# Tipos de dados: structs

```
typedef struct directory_entry {
 char type;
 char name[MAX_NAME_LENGTH];
 unsigned char day;
 unsigned char month;
 unsigned char year;
 int size;
 int first_block;
} dir_entry;
```



# Chamada indireta de funções

```
main()
```

```
{
```

```
 int (*pf) (); // declaração de apontador para função,
 // nenhuma função existe ainda
```

```
}
```

# Chamada indireta de funções

```
main ()
{
 int (*pf) (); // declaração de apontador para função,
 // nenhuma função existe ainda
```

```
// declaração de uma função
void escreve(int n)
{
 int i = 0;
 while (i++ < n) {
 printf("Hello world!\n");
 }
}
```

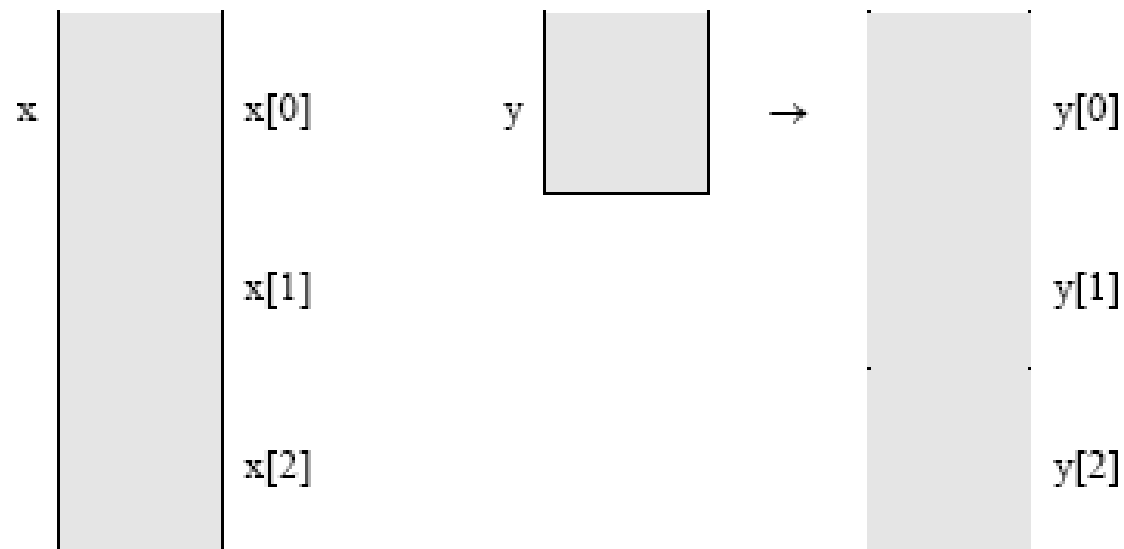
# Chamada indireta de funções

```
main ()
{
 int (*pf) (); // declaração de apontador para função,
 // nenhuma função existe ainda
 pf = &escreve; // apontador recebe o endereço da função
 escreve()
 (*pf) (10); // chamada da função com argumento = 10
 exit (0);
}
// declaração de uma função
void escreve(int n)
{
 int i = 0;
 while (i++ < n) {
 printf("Hello world!\n");
 }
}
```

# Alocação dinâmica de memória

```
int x[3];
```

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





# Fonte de problema

```
// test.c: seg fault!
#include <stdio.h>
#include <string.h>

char *x; // declaração de ponteiro: ocupa 4 bytes
main()
{
 strcpy(x, "Hello world!");
 printf("Valor de x = %s\n", x);
}
```

Inicialização do apontador????





# Fonte de problema

```
// test1.c: seg fault!
#include <stdio.h>
#include <string.h>

char *x; // declaração de ponteiro: ocupa 4 bytes
main()
{
 strcpy(x, "Hello world!");
 printf("Valor de x = %s\n", x);
}
```

Inicialização do apontador????



# Fonte de problema

```
// test2.c: funciona, mas está errado!
#include <stdio.h>
#include <string.h>

main()
{
 char *x;

 strcpy(x, "Hello world!");
 printf("Valor de x = %s\n", x);
}
```



# Fonte de problema

```
// test3.c: parou de funcionar!
#include <stdio.h>
#include <string.h>

main()
{

 int y;
 char *x;

 strcpy(x, "Hello world!");
 printf("Valor de x = %s\n", x);
}
```



# Fonte de problema

```
// test4.c: voltou a funcionar! Continua errado!
#include <stdio.h>
#include <string.h>

main()
{

 int y;
 int z;
 char *x;
 y = 10;

 printf("Valor de y = %d\n", y);
 printf("Valor de z = %d\n", z);
 strcpy(x, "Hello world!");
 printf("Valor de x = %s\n", x);
}
```