

## História

Linguagem C

- Criada por Dennis M.Ritchie em 1972, nos laboratórios Bell
- Usada para desenhar SISTEMAS OPERATIVOS
  - O Unix, que apareceu na mesma altura nos laboratórios Bell, foi re-escrito em C, e desde então Unix e C têm-se mantido ligados
- Definida por Kernigham e Ritchie (norma K&R) em 1977
- Definida por uma norma ANSI em 1988
- Muito utilizada actualmente por programadores profissionais e em programação de baixo nível.

1

V.Lobo@EN

## OBJECTIVOS E CARACTERÍSTICAS

Linguagem C

- Fácil de implementar (compiladores eficientes)
- Proximidade à linguagem máquina, para tirar o máximo rendimento da máquina
- Estruturas de alto nível que facilitam a programação
- ENORME flexibilidade
- Utilizada principalmente por programadores profissionais
- Delega muita da responsabilidade pela correcção no programador -> muito sujeita a erros

2

V.Lobo@EN

## ESTRUTURA DE UM PROGRAMA C

Linguagem C

- Conjunto de funções
- O programa principal é uma função chamada "main()"
- O programa pode fazer referência a entidades externas (ex. rotinas), através das *directivas include* e *extern*

```
#include <stdio.h>

main()
{
    int x,y;
    x=3;
    y=Metade(x);
    printf("metade de %d é %d",x,y);
}

int Metade(int z)
{
    int a;
    a=z/2;
    return(a);
}
```

3

## ESTRUTURA DE UM PROGRAMA C

Linguagem C

- Tudo o que é definido dentro de uma função é **LOCAL** a essa função
  - Várias funções podem ter variáveis com o mesmo nome
- Fora das funções podem-se definir tipos e variáveis **GLOBAIS**
  - Devem-se evitar variáveis globais
  - São uma fonte de erros !
- As funções podem ser definidas em, ficheiros separados, chamados **MÓDULOS**

4

## ESTRUTURA DE UM PROGRAMA C

Linguagem C

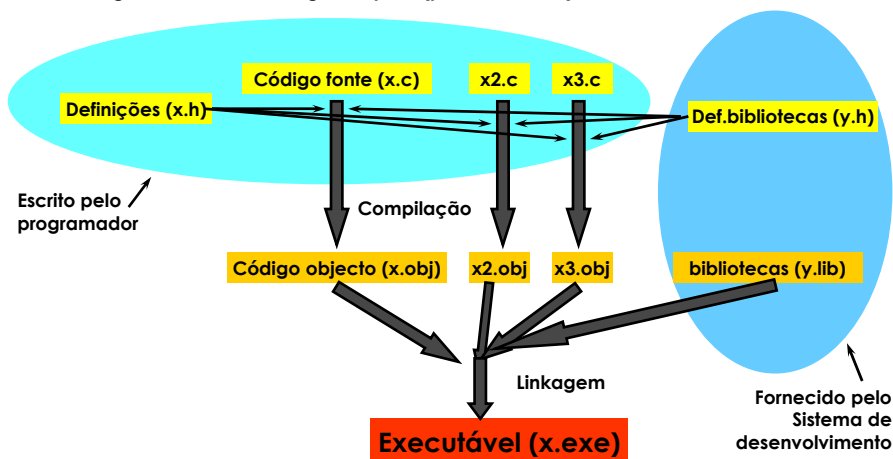
- Os ficheiros que contêm **código** (funções, variáveis, etc) têm normalmente a extensão **“.C”**
  - Dão origem a código máquina, ou à reserva de espaço de memória
- Os ficheiros que apenas têm **definições** têm normalmente a extensão **“.H”**
  - Não geram código, mas são necessários para que seja possível com compilar os outros módulos

5

## Ficheiros, Compilação, e Linkagem

Linguagem C

- Código fonte = texto
- Código objecto = código máquina + referências a outros módulos
- Código executável = código máquina (pronto a correr)

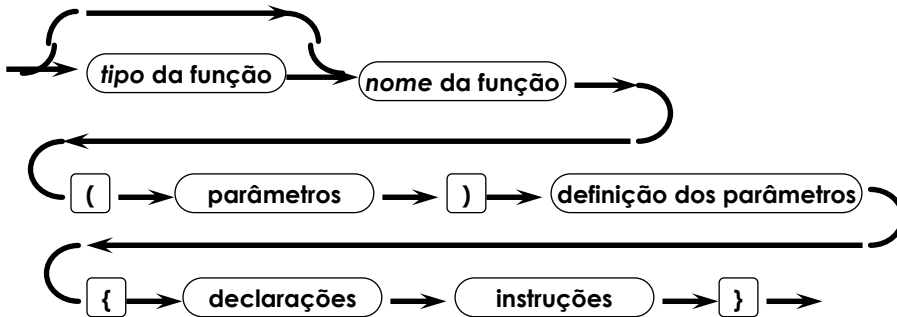


6

## ESTRUTURA DE UMA FUNÇÃO

Linguagem C

- **Sintaxe moderna**
  - Igual ao Pascal
- **Sintaxe antiga**
  - No cabeçalho indicam-se os nomes dos parâmetros, e mais tarde definem-se os tipos



7

## Instruções

Linguagem C

- **Terminam sempre com ponto e vírgula ( ; )**
- **Um conjunto de instruções envolvido em chavetas ( { } ) chama-se um BLOCO e comporta-se como uma instrução**
- **Instrução de atribuição**
  - *variável* = expressão;
  - exemplos
 

```
a = 3;          x = 3+4/2;      y = sin( 3*pi );
```
- **Chamada de uma função**
  - *nome\_da\_função* ( *parametros\_reais* );
  - Exemplos
 

```
printf("ola");  calc_max( vect1);  fopen(f1, "fich", "r");
```

8

## INSTRUÇÕES DE CONTROLO DE FLUXO

Linguagem C

### • Blocos condicionais

- **if** ( expressão ) instrução [ **else** instrução ]
- Se expressão  $\neq 0$   $\Rightarrow$  o bloco principal é executado
- Se expressão = 0  $\Rightarrow$  o bloco else é executado (se existir)
- Exemplo

```

a=3;
if( a )
    printf("certo");
else
    printf("asneira");
if( a==2 )
    printf("a é 2");
else
    {
        printf("a vai ser 2");
        a=2;
    }

```

### Operadores relacionais

==	igual
<=	menor ou igual
>=	maior ou igual
!=	diferente
<	menor
>	maior

**Cuidado ! ==  $\neq$  =**

9

V.Lobo@EN

## INSTRUÇÕES DE CONTROLO DE FLUXO

Linguagem C

### • Blocos de seleção múltipla

- **switch** ( expressão ) { < **case** constante : instrução > [ **default** : instrução ] }
- a partir do momento em que é encontrado o caso aplicável, todas as instruções daí para a frente são executadas
- Exemplo

```

switch( num )
{
    case 3 : printf("era 3 ");
    case 4: printf("era <5");break;
    default: printf("era grande");
}

```

### INSTRUÇÃO break

- aborta a execução do bloco
- muito usado no switch

10

V.Lobo@EN

## INSTRUÇÕES DE CONTROLO DE FLUXO

Linguagem C

### ● Ciclo while

– **while** ( expressão ) instrução

– Enquanto a expressão for verdadeira (i.e.  $\neq 0$ ) as instruções do bloco são executadas, sendo o teste feito ANTES de iniciar novo ciclo.

– Exemplo

```
a = 1;
while( a<5)
{
    printf("ola");
    a = a+2;
}
```

### ● Ciclo do while

– **do** instrução **while** ( expressão )

– Enquanto a expressão for verdadeira (i.e.  $\neq 0$ ) as instruções do bloco são executadas, sendo o teste feito DEPOIS de ter acabado um ciclo.

– Exemplo

```
do
{
    printf("...");
    a = a+2;
}while( a<1)
```

V.Lobo@EN

11

## INSTRUÇÕES DE CONTROLO DE FLUXO

Linguagem C

### ● Ciclo for

– **for** ( inicialização ; condição ; incremento )  
instrução

– O código de inicialização é executado antes de se iniciar o ciclo.

– Seguidamente a condição é verificada. Se ela for falsa ( i.e.  $= 0$ ) o ciclo termina. Se não, são executadas as instruções do ciclo

– Por último, é executado o código do incremento, após o qual a condição é novamente testada.

– Exemplo

```
for( i=1; i<11; i=i+1 )
    printf("ola\n");

for( j=0; j<27; j++ )
    printf("adeus");
```

V.Lobo@EN

12

## TIPOS DE DADOS ELEMENTARES

Linguagem C

- **Inteiros**

- int - Inteiro
- long - Inteiro (com maior gama de variação) (4bytes)
- short - Inteiro (com menor gama de variação) (1 byte)
- unsigned - Inteiro (só positivos)

- **Reais**

- float - Vírgula flutuante
- double - Vírgula flutuante (dupla precisão)

- **Caracteres**

- char - Caracter (1 byte)

- **Enumerados**

- enum - Enumerados

13

V.Lobo # 08

## TIPOS DE DADOS ELEMENTARES

Linguagem C

- **As variáveis são declaradas indicando primeiro o tipo, depois o nome da variável.**

- Exemplo: 

```
int    x,y;
char  a,b,c;
```

- **Uma variável pode ser inicializada assim que é declarada.**

- Exemplo: 

```
int    z=27;
char  a='F',b='E';
```

- **Muitas vezes, pretende-se saber o ENDEREÇO (em memória) de uma variável. O endereço pode ser obtido através do operador `&nome_da_variável`**

- Exemplo: 

```
x = &y;
scanf("%d", &a );
```

14

V.Lobo # 08

## ESTRUTURAS DE DADOS - MATRIZES

Linguagem C

- As matrizes são declaradas indicando o nome e, entre parêntesis rectos, o tamanho
  - Exemplo: `int x[20]` /\* Define uma matriz de 20 inteiros \*/  
`float y[9]` /\* Define uma matriz de 9 reais \*/
- Os índices começam sempre em 0
  - A matriz definida com `x[20]` tem os elementos `x[0],x[1]...x[19]`
- Os elementos da matriz são referidos indicando o seu índice entre parêntesis rectos
  - `y[6]` é o elemento número 6 da matriz `y` (ou seja o 7º elemento)

**y**

						2.5		
0	1	2	3	4	5	6	7	8

V.Lobo@EN

15

## ESTRUTURAS DE DADOS - MATRIZES

Linguagem C

- As matrizes de dimensão superior a 1 são matrizes de matrizes
  - `int z[10][20];` /\* matriz de 2 dimensões com 10x20 elem.\*/
  - `int b[10][10][10];` /\* matriz de 3 dimensões (cubo) com 10x10x10 el.\*/
  - `char tab[4][3]` /\* matriz de 2 dimensões com 4x3 elem.\*/
- Problema:
  - imprimir o conteúdo ma matriz `c`

Código a executar

```
for(i=0;i<4;i++)
  for(j=0;j<3;j++)
    printf("%c",tab[i,j]);
```

Conteúdo da matriz

a	b	c	d
e	f	g	h
i	j	k	l

Resultado

a e i b f j c g k d h l

V.Lobo@EN

16



## ESTRUTURAS DE DADOS - STRINGS

Linguagem C

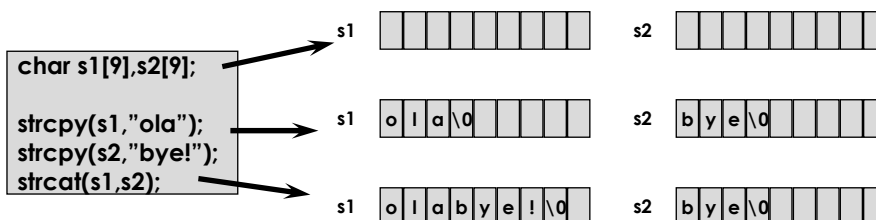
- Uma matriz unidimensional (vector) de caracteres chama-se uma cadeia ou *string*
- Uma *string* deve terminar com o caracter nº 0 (escreve-se \0)
  - O terminador pode estar em qualquer posição da string,. Se fôr logo o primeiro caracter, então a string está vazia
  - Por vezes chama-se a uma cadeia destas uma *null-terminated string*
- Para o compilador, um conjunto de caracteres delimitado com aspas (") é tratado como uma *null terminated string* (o \0 é acrescentado automaticamente)

17

## ESTRUTURAS DE DADOS - STRINGS

Linguagem C

- Existe um módulo de funções para manipular strings, definidas no módulo *string.h*
  - `char *strcpy(char *dest, const char *src);` - copia uma string
  - `char *strcat(char *dest, const char *src);` - junta 2 strings
  - `char *strchr(const char *s, int c);` - procura 1 char na string
  - `int strcmp(const char *s1, const char*s2);` - compara 2 strings



18

## ESTRUTURAS DE DADOS - "struct"

Linguagem C

- Semelhantes aos RECORDS do Pascal

- Definição

- **struct** *nome opcional* { *definição\_das\_variáveis* }  
*nome\_da\_variável*;

- Exemplo:

```
struct {
    int telef;
    char nome[20];
} a, b[20];

strcpy( a.nome, "V.Lobo" );
a.telef=2745658;
b[3].telef=4662315;
```

- Referência

- Como em pascal, as estruturas são referenciadas indicando o nome, um ponto, e o nome da variável interna da estrutura

19

V.Lobo@EN

## ESTRUTURAS DE DADOS - TIPOS

Linguagem C

- Definição de tipos

- **typedef** { *definição\_das\_variáveis* } *nome\_do\_tipo*;

- O novo tipo pode ser usado a partir daí como um tipo elementar

- Deve ser usado sempre que se usam structs

- Exemplo;

```
typedef struct x {
    int telef;
    char nome[20];
} ficha;
```

```
ficha a,b[20];
```

```
strcpy( a.nome, "V.Lobo" );
a.telef=2745658;
b[3].telef=4662315;
```

Nome opcional das structs

Serve para, dentro da definição da struct, fazer referências à própria struct

Será usada quando usarmos apontadores

20

V.Lobo@EN

## Pré-compilador

Linguagem C

- Em C, existe um pré-compilador que se encarrega de processar as *directivas do compilador*
- As *directivas do compilador* começam sempre na primeira coluna, com o símbolo #
- Definição de constantes
  - **#define** nome definição
  - Substitui a partir desse ponto TODAS as ocorrências da string nome pela string definição.
  - É conveniente usar “defines” SEMPRE que trabalhamos com uma constante, pois será muito mais fácil mudar o seu valor.
  - É costume os “defines” definirem nomes escritos em maiúsculas.

```
#define PI 3.1415
a=2+PI
```



```
a=2+3.1415
```

21

V.Lobo@EN

## Pré-compilador

Linguagem C

- Inclusão de ficheiros externos
  - **#include** <nome\_do\_ficheiro>
  - **#include** “nome\_do\_ficheiro”
  - Insere nesse ponto o ficheiro indicado
  - Se se usar os separadores <>, o pré-compilador vai buscar o ficheiro à directoria das bibliotecas standardizadas do sistema: à *include directory*
  - Se se usar os separadores “”, o pré-compilador vai buscar o ficheiro à directoria actual
- Existem vários *include files* standardizados
  - `stdio.h` Definições básicas (inclui `printf`, `scanf`, etc)
  - `string.h` Funções para strings (`strcpy`, `strcmp`, etc)
  - `math.h` Funções matemáticas (`sin`, `cos`, etc)
  - `stdlib.h` Funções utilitárias várias (`malloc`, `atoi`, etc)
  - `time.h`, `dos.h`, `limits.h`, etc.

22

V.Lobo@EN

## Instruções de I/O

Linguagem C

### • Escrita formatada na consola

- **printf** ( *formatação, dados* )
- *formatação* é uma string que contém caracteres “printáveis” e especificadores de conversão que permitem a escrita de variáveis.
- *dados* são as variáveis que serão impressas através dos especificadores de conversão, e são separados por vírgulas.

### • Especificadores de conversão

- %d      Inteiros, em decimal
- %o      Inteiros, em octal
- %x      Inteiros, em hexadecimal
- %f      Floats, em vírgula flutuante
- %e      Floats, em notação exponencial
- %s      Strings
- %c      Caracteres
- existem muitos mais

```
a=20;
printf("Teste %d %x",a,a);
```

Escreve: Teste 20 14

V.Lobo # EN

23

## Instruções de I/O

Linguagem C

### • Leitura formatada da consola

- **scanf** ( *formatação, dados* );
- *formatação* é uma string que contém caracteres especificadores de conversão
- *dados* são as os ENDEREÇOS das variáveis onde os valores deverão ser escritos
- O endereço de uma string é o próprio nome da string
- Exemplo

```
int a,b;
char s1[10];

scanf("%d,%s,%d", &a, s1, &b);
```

### • Leitura/escrita directa

- *variável* = **getchar** ();
- **putchar** (*expressão* );
- exemplo: **putchar**(upper(getchar()));
- Não se deve misturar os tipos de acesso porque o acesso formatado é “bufferizado”

V.Lobo # EN

24

## Ficheiros

Linguagem C

- As variáveis e rotinas para trabalhar com ficheiros estão contidos em `stdio.h`.
- Para se usar um ficheiro é necessário *abrir o ficheiro antes de o usar, e fechá-lo no fim.*
- Um ficheiro é aberto num determinado *modo*: *modo leitura, modo escrita, modo acrescentar, etc.*
- Dentro do programa um ficheiro aberto é identificado por uma *variável* : um apontador da estrutura `FILE` que é o seu *identificador interno*.

25

V.Lobo © EN

## Ficheiros

Linguagem C

- Declaração da variável
  - `FILE *identificador;`
  - exemplo: `FILE *ficheiro_1, *dados_entrada, *dados_saida;`
- Abertura do ficheiro
  - `identificador = fopen( nome, modo );`
  - `nome` é uma string contendo o nome do ficheiro no disco.
  - `modo` é uma string contendo o modo de acesso ao ficheiro. Os modos mais usuais são:
    - r leitura
    - w escrita
    - a acrescento (escrita a partir do fim do ficheiro existente)
  - exemplo:

```
FILE *in_file,*out_file;
```

```
in_file = fopen("carta.doc","r");
out_file = fopen("carta.txt","w");
```

26

V.Lobo © EN

# Ficheiros

Linguagem C

## • Acesso ao ficheiro

- `fscanf` ( *identificador, formatação, dados* );
- `fprintf` ( *identificador, formatação, dados* )
- *variável* = `fgetc` ( *identificador* );
- `fputc` ( *identificador, expressão* );
- Na realidade, estas funções são todas para trabalhar com ficheiros: a consola é que é tratada como um ficheiro chamado *stdin* e *stdout*.

## • Fecho do ficheiro

- `fclose`( *identificador* );
- A interacção entre o programa e o ficheiro termina. É necessário fechar o ficheiro ou invocar a função `fflush()` para garantir que as alterações ao ficheiro ficam mesmo no disco e não no buffer (RAM)

```
fprintf( out_file, "o resultado é %d", x);
fclose( out_file );
```

V.Lobo # 08

27

# Operadores

Linguagem C

+	Soma
-	Subtracção
*	Multiplicação
/	Divisão
%	Módulo
&&	And lógico
	Ou lógico
!	Negação lógica
&	And, bit a bit
	Or, bit a bit
^	Or exclusivo, bit a bit
~	Complemento
>>	Shift para a direita
<<	Shift para a esquerda

<code>x++</code>	incrementa x(usa primeiro o valor, e incrementa depois)
<code>++x</code>	Incrementa (usa valor incrementado)
<code>--x, x--</code>	Decrementa

## INSTRUÇÕES DE AFECTAÇÃO

$$\underline{x} \text{ += expressão} \iff x = x + \text{expressão}$$

<code>-=</code>	<code>*=</code>	<code>/=</code>	<code>%=</code>	<code>&amp;=</code>
<code>^=</code>	<code> =</code>	<code>&lt;&lt;=</code>	<code>&gt;&gt;=</code>	

<code>sizeof( x )</code>	Número de bytes ocupados por x
<code>(um_tipo)x</code>	Converte a variável x para um novo tipo (cast).

↑  
Cuidado !

V.Lobo # 08

28

## Exemplo (Versão 1)

Linguagem C

### ● Problema:

- Manter um ficheiro dos 30 alunos de uma turma. O ficheiro deverá ter o
  - nome do aluno ( com 20 caracteres)
  - n° do telefone
  - Cota de mérito
- Alterar os dados de um aluno
- Guardar/ir buscar os dados a disco
- Mostrar no ecrã uma listagem de todos os alunos

### ● 1º Passo:

- definir a estrutura de dados

```
typedef struct x{
    char    nome[20];
    int     telef;
    float   cota;
} ficha;

ficha  ficheiro[30];
```

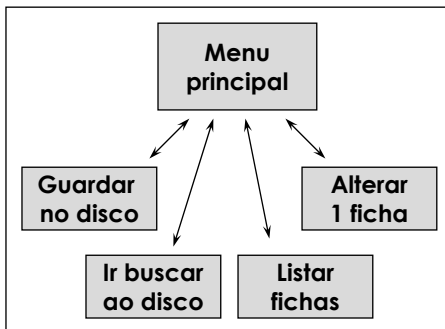
29

V.Lobo@EN

## Exemplo (Versão 1)

Linguagem C

### ● 2º Passo: diagrama de blocos



### ● 3º Passo: programa principal



```
main()
{
    char opcao;

    do
    {
        printf(" 1 - Guardar\n");
        printf(" 2 - Buscar\n");
        printf(" 3 - Listar\n 4- Alterar\n");
        printf(" 5 - Sair\n");
        scanf("%c", opcao );
        switch(opcao) {
            case 1: Guardar();break;
            case 2: Buscar();break;
            case 3: Listar();break;
            case 4: Alterar();
            }
        } while( opcao!= '5' );
    }
}
```

30

V.Lobo@EN

## Exemplo (Versão 1)

Linguagem C

### ● 4º Passo: escrever as rotinas

```

Listar()
{
    int i;

    for( i=0; i<30;i++)
        printf("%d - %s %d %5.2f",i,ficheiro[i].nome,ficheiro[i].telef,ficheiro[i].cota);
}

```

```

Guardar()
{
    int          i;
    FILE         *fich;
    fich = fopen( "dados.dat", "w");
    for( i=0; i<30;i++)
        fprintf(fich, "%s %d %5.2f",ficheiro[i].nome,
                ficheiro[i].telef,ficheiro[i].cota);
    fclose(fich);
}

```

V.Lobo@EN

31

## Exemplo (Versão 1)

Linguagem C

```

Buscar()
{
    int          i;
    char         nome_fich[12];
    FILE         *fich;

    printf("Qual o nome do ficheiro de dados ?");
    scanf("%s",nome_fich);
    fich = fopen( nome_fich, "r");

    for( i=0; i<30;i++)
        fscanf(fich, " %s %d %f",ficheiro[i].nome,
                &ficheiro[i].telef,&ficheiro[i].cota);
    fclose(fich);
}

```

V.Lobo@EN

32



## Exemplo (Versão 1)

Linguagem C

```

Alterar()
{
    int    i;

    printf("Qual o número da ficha a alterar ?");
    scanf("%d",&i);

    printf("Qual o novo nome ?");scanf("%s",ficheiro[i].nome);
    printf("Qual o novo telef ?");scanf("%d",&ficheiro[i].telef);
    printf("Qual a nova cota ?");scanf("%s",&ficheiro[i].cota);
}

```

- **Último passo: juntar tudo**

- Criar um ficheiro (ex: prob.h) com a definição dos tipos de dados, e declarações (como extern) de todas as rotinas e variáveis.
- Criar um ficheiro principal com a definição das variáveis globais (ficha ficheiro[30]) e do programa principal
- "Linkar" os módulos todos

33

V.Lobo@EN

## Apontadores

Linguagem C

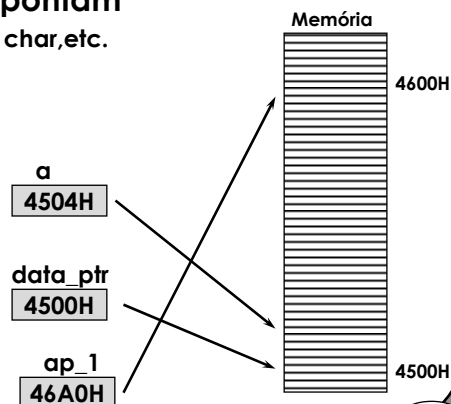
- Existem em C variáveis que guardam ENDEREÇOS e não o próprio dado: os *apontadores*.
- Os apontadores estão sempre associados ao tipo da variável para a qual apontam
  - Há apontadores para int, para char, etc.

- **Declaração**

- *tipo* \**nome*;
- Exemplos:  
int \*a, \*ap\_1, \*data\_ptr;

```
int x,*y;
```

```
x=2;
y=&x;
```



34

V.Lobo@EN

## Apontadores

Linguagem C

### ● Acesso às entidades apontadas

- A entidade apontada é referenciada com *\*nome*, isto é, se *x* for um apontador para um inteiro, *\*x* é o inteiro apontado por *x*.

```
int *a,b;
```

```
a=&b;
```

```
b=3;
```

```
*a=6;
```

Qual o valor de *b*?

### ● Aritmética de ponteiros

- Posso incrementar/decrementar ponteiros (++/--)
- O ponteiro será incrementado numa quantidade igual à dimensão da entidade apontada (1 para caracteres, 2 para inteiros de 2 bytes, etc.)

35

V.Lobo@EN

## Apontadores

Linguagem C

### ● Alocação dinâmica de espaço

- Todas as variáveis que vimos até agora são criadas (i.e. o espaço de memória é reservado para elas) antes do código começar a executar.
- É possível reservar espaço durante a execução do código, pondo um apontador a apontar para o endereço desse novo espaço
- Antes de terminar o programa é necessário libertar a memória reservada

### ● Motivos para usar estruturas dinâmicas:

- melhor aproveitamento do espaço de memória
- maior eficiência na manipulação de dados
- maior flexibilidade na construção da estrutura de dados

36

V.Lobo@EN

# Apontadores

Linguagem C

## • Alocação

- `apontador = malloc ( n°_de_bytes );`
- Reserva `n°_de_bytes` bytes de memória e põe o apontador a apontar para lá.

## • Liberação

- `free ( apontador );`
- Liberta o espaço apontado por `apontador`

37

V.Lobo # 08

# Manipulação de estruturas

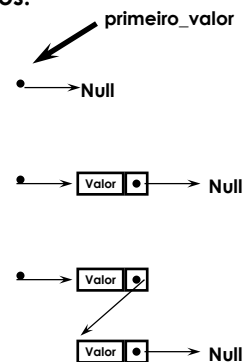
Linguagem C

## • Listas ligadas

- Temos que ter um “ponto fixo” que serve de ponto de entrada para a estrutura
- Necessitamos de variáveis auxiliares para percorrer a estrutura
- Exemplo: criar uma lista de 20 elementos.

```
typedef struct x{
    int    valor;
    struct x  *next;
} elemento;
elemento *primeiro_valor, *aux;

primeiro_valor = malloc(sizeof(elemento));
aux = primeiro_valor;
for(i=0;i<19;i++)
{
    aux->next = malloc(sizeof(elemento));
    aux=aux->next;
}
aux->next=NULL;
```



38

V.Lobo # 08

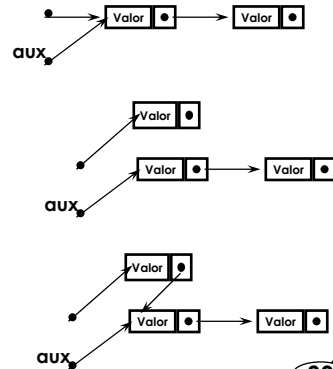
# Manipulação de estruturas

Linguagem C

- **Listas ligadas** (outra maneira de inserir elementos)
  - No exemplo anterior, os elementos mais recentes eram postos no fim. no exemplo que se segue são postos em primeiro lugar

```
typedef struct x{
    int    valor;
    struct x  *next;
} elemento;
elemento *primeiro_valor, *aux;

primeiro_valor = malloc(sizeof(elemento));
aux = primeiro_valor;
for(i=0;i<19;i++)
{
    primeiro_valor = malloc(sizeof(elemento));
    primeiro_valor ->next = aux;
    aux = primeiro_valor;
}
aux->next=NULL;
```



39

# Exemplo revisto

Linguagem C

- **Problema:**
  - Manter um ficheiro para qualquer número de alunos de uma turma.
  - Ver exemplo anterior

- **1º Passo:**
  - definir a estrutura de dados

```
typedef struct x{
    char    nome[20];
    int     telef;
    float   cota;
    struct x  *next;
} ficha;

ficha *prim_fich, aux1, aux2;
```

- **2º Passo: diagrama de blocos**
  - Idêntico.
- **3º Passo: programa principal**
  - Idêntico, mas podemos acrescentar *inserir ordenadamente*.

40

## Exemplo revisto

Linguagem C

### • 4º Passo: escrever as rotinas

```
Listar()
{
    for( aux1=prim_fich; aux1 != NULL; aux1=aux1->next )
        printf("%s %5.2f %d",aux1->nome, aux1->cota, aux1->telef);
}
```

```
Guardar()
{
    FILE      *fich;
    fich = fopen( "dados.dat", "w");

    for( aux1=prim_fich; aux1 != NULL; aux1=aux1->next )
        fprintf(fich, "%s %5.2f %d",aux1->nome, aux1->cota, aux1->telef);
    fclose(fich);
}
```

```
Buscar()
{ ...../* idêntica à anterior, mutatis mutandi */ ...}
```

V.Lobo@EN

41

## Exemplo revisto

Linguagem C

```
Alterar()
{
    char      nome[10];

    printf("Qual o nome que a alterar ?");
    scanf("%s", nome);

    aux1=prim_fich;          /*vou procurar a ficha */
    while( (aux1 != NULL) && strcmp(aux1->nome, nome) )
        aux1=aux1->next;

    if( aux1 != NULL )      /* se a encontrei...altero-a */
    {
        printf("Qual o novo nome ?");scanf("%s",aux1->nome);
        printf("Qual o novo telef ?");scanf("%d",&(aux1->telef));
        printf("Qual a nova cota ?");scanf("%s",&(aux1->cota));
    }
    else
        print("Desculpe mas essa ficha não existe !\n");
}
```

V.Lobo@EN

42

## Exemplo revisto

Linguagem C

```

Inserir_ordenadamente()
{
    ficha      *nova;
    nova=malloc(sizeof(ficha));
    printf("Qual o novo nome ?");scanf("%s",nova->nome);
    printf("Qual o novo telef ?");scanf("%d",&(nova->telef));
    printf("Qual a nova cota ?");scanf("%s",&(nova->cota));

    if( strcmp(nova->nome, prim_fich->nome) < 0 )
        { nova->next = prim_ficha; prim_ficha = nova; }
    else
        { aux1=prim_fich;
          aux2=aux1;
          while( (aux1 != NULL) && (strcmp(nova->nome, aux1->nome) < 0) )
              { aux2=aux1;
                aux1=aux1->next; }
          nova->next= aux1;
          aux2->next= nova; }
}

```

V.Lobo@EN

43

## Recursividade

Linguagem C

- **Ideia base:**
  - Ter uma função, que se chama a si própria
  - Definir um procedimento, à custa de usar próprio procedimento
- **Exemplo:**
  - Calcular o FACTORIAL de um numero (  $n!$  )
  - Definição:
    - O factorial de  $n$  é  $n$  vezes o factorial de  $n-1$ , e o factorial de 1 é 1.
    - $n! = n \times (n-1)! \wedge 1! = 1$

```

int factorial(int x)
{
    if( x== 1)
        return 1;
    else
        return factorial(x-1);
}

```

```

int factorial(int x)
{
    int valor;
    if( x== 1)
        valor=1;
    else
        valor=factorial(x-1);
    return valor;
}

```

V.Lobo@EN

44

# Recursividade

Linguagem C

## ● Problemas:

- Recursão infinita
- Tamanho do stack
- Interação entre parâmetros, variáveis locais, e variáveis globais

## ● Possibilidade de usar outros métodos

- Exemplo:

→  $\text{factorial}(n) = n \times (n-1) \times (n-2) \times (n-3) \dots \times 2 \times 1$

### CUIDADO !

Retorna-se ao ponto de onde a função foi chamada (ou seja, a si própria)

```
int factorial(int x)
{
    int i, valor;

    valor=1;
    for(i=x; i>1; i++)
        valor=valor*i;

    return valor;
}
```

45

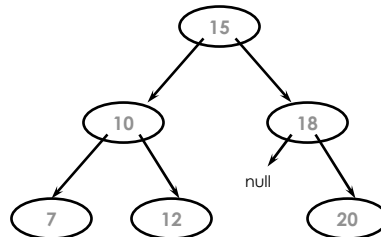
V.Lobo@EN

# Árvores Binárias

Linguagem C

## ● Árvores Binárias

- Motivação:
  - busca dicotômica
- Ideia base:
  - Cada elemento tem dois sucessores
  - Um dos sucessores está “à direita” o outro “à esquerda”
- Exemplo:



```
typedef struct x{
    int    valor;
    struct x  *smaller,*greater;
} elemento;

elemento *primeiro_valor, *aux;
```

46

V.Lobo@EN

## Árvores Binárias

Linguagem C

- Procurar um elemento na árvore:

```
elemento *procura(elemento *ponto_partida, int valor)
{
    if( ponto_partida==null )
        return null;
    if( ponto_partida->valor == valor )
        return ponto_partida;
    if( ponto_partida->valor > valor )
        return procura( ponto_partida->greater, valor );
    else
        return procura( ponto_partida->smaller, valor );
}
```

- Para inserir um elemento: (tpc)
- Para retirar um elemento: (tpc)

47

V.Lobo EN

## Manipulação de estruturas

Linguagem C

- Recursividade na manipulação de árvores
- Análise de eficiência de árvores/listas/matrizes
- Buffers circulares
- Stacks
- Sets

48

V.Lobo EN