

Linguagem C

- A origem da linguagem C
- Primeiro exemplo
- Variáveis e Constantes
- Operadores
- Operadores relacionais e lógicos
- Expressões
- Cast
- Estruturas de controlo e fluxo
- Ciclos
- Vetores
- Strings
- Matrizes bidirecionais
- Apontadores

A origem da linguagem C

■ Origem da Linguagem C

- ▶ O C nasceu na década de 70. Seu inventor, Dennis Ritchie, implementou-o pela primeira vez usando um DEC PDP-11 rodando o sistema operacional UNIX. O C é derivado de uma outra linguagem: o B, criado por Ken Thompson. O B, por sua vez, veio da linguagem BCPL, inventada por Martin Richards.

■ Como correr um programa na linguagem C?

- ▶ A linguagem C é uma linguagem que precisa de ser compilada e depois ligada:
 - Utilizar um compilador significa passar o código em linguagem C em código Objeto (com extensão “.o”)
 - Utilizar o link, significa passar o código compilado em código executável (com a extensão “.exe”)

Primeiro exemplo

■ Na linha de comandos

- 1) Abrir o editor *gedit* (se não estiver instalado executar *apt-get install gedit*) e atribuir o nome *primeiro_teste.c*
- 2) Introduzir o **programa** em baixo.
- 3) Executar a instrução *gcc -c primeiro_teste.c -o primeiro_teste.o*
- 4) Executar a instrução *ls* (vai ver que surge um ficheiro com extensão “.o”)
- 5) Executar a instrução *gcc primeiro_teste.o -o primeiro_teste.exe*
- 6) Executar o programa com a instrução: *./primeiro_teste.exe*

■ O programa

```
#include <stdio.h>

void main ()    /* Um Primeiro Programa */
{
    printf ("Ola! Eu estou vivo!\n");
}
```

Variáveis e Constantes

■ Variáveis e constantes

- O nome deve começar com uma letra ou sublinhado: `_a`, `a`, `b`, `c`

Habitualmente utilizam-se letras minúsculas para nomes de variáveis e letras maiúsculas para constantes.

- Tipos em C
 - ▶ Cinco tipos: **char**, **int**, **float**, **void** e **double**.
 - ▶ Modificadores dos tipos: `signed`, `unsigned`, `long` e `short`.

Tipo	Num de bits	Intervalo	
		Início	Fim
char	8	-128	127
unsigned char	8	0	255
signed char	8	-128	127
int	16	-32.768	32.767
unsigned int	16	0	65.535
signed int	16	-32.768	32.767
short int	16	-32.768	32.767
unsigned short int	16	0	65.535
signed short int	16	-32.768	32.767
long int	32	-2.147.483.648	2.147.483.647
signed long int	32	-2.147.483.648	2.147.483.647
unsigned long int	32	0	4.294.967.295
float	32	3,4E-38	3,4E+38
double	64	1,7E-308	1,7E+308
long double	80	3,4E-4932	3,4E+4932

Variáveis e Constantes

- As variáveis têm de ser declaradas (diferente do Python) ou declaradas e inicializadas (*):

```
char ch, letra;  
long count;  
float pi;
```

```
char ch='D';  
int count=0;  
float pi=3.141;
```

- Exemplo de declaração de variáveis:

```
#include <stdio.h>  
int contador;
```

Variável global: Existe em todo o programa.

```
int func1(int j){
```

Variáveis local: Existe apenas no contexto da função *func1*.

```
}
```

```
int main()
```

```
{
```

```
char condicao;
```

```
int i;
```

```
for (i=0;i<10;i++)
```

```
{
```

```
/* Bloco do for */
```

```
float f2;
```

```
...
```

```
func1(i);
```

```
}
```

Variáveis local: Existe apenas no contexto do programa principal.

(*) Como por defeito o tipo é *int*, o *long* não precisa do *int*)

Exercício 1

- Escreva um programa que declare uma variável inteira global e lhe atribua o valor 10. Declare outras 5 variáveis inteiras locais ao programa principal e atribua-lhes os valores 20, 30, ..., 60. Declare 6 variáveis caracteres e atribua-lhes as letras *c*, *o*, *e*, *l*, *h*, *o* . Finalmente, o programa deverá imprimir, usando todas as variáveis declaradas:

As variáveis inteiras contem os numeros:

10, 20, 30, 40, 50, 60

O animal contido nas variáveis caracteres é o coelho.

Operadores

- Os operadores utilizados em C são os seguintes

Operador	Ação
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto de divisão (de inteiros)
++	Incremento (inteiro e ponto flutuante)
--	Decremento (inteiro e ponto flutuante)

- Execute as seguintes operações:

```
int a = 17, b = 3;
int x, y;
float z = 17. , z1, z2;
x = a / b;
y = a % b;
z1 = z / b;
z2 = a/b;
A ++
B = b-1
```

- ▶ A divisão inteira ocorre entre dois inteiros.
- ▶ O resultado y é 2? Porquê?
- ▶ Z1 é um resultado de divisão inteira? Porquê?

Operadores Relacionais e lógicos

- Existem operadores relacionais e lógicos...

Operador	Ação
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

Operador	Ação
&&	AND (E)
	OR (OU)
!	NOT (NÃO)

- Será que j é incrementado?

```
int i = 5, j = 7;  
if ( ( i > 3 ) && ( j <= 7 ) && ( i != j ) ) j++;
```


Expressões

- Expressões são combinações de variáveis, constantes e operadores. Quando montamos expressões temos que levar em consideração a ordem com que os operadores são executados, conforme a tabela de precedências da linguagem C.
- Como são realizadas estas operações? Justifique com base nas precedências.

```
Anos=Dias/365.25;  
i = i+3;  
c= a*b + d/e;  
c= a*(b+d)/e;
```

Maior precedência () [] ->
! ~ ++ -- . -(unário) (cast) *(unário) &(unário) sizeof
* / %
+ -
<< >>
<<= >>=
== !=
&
^
|
&&
||
?
= += -= *= /=
Menor precedência ,

Cast

- Executar um **cast** significa converter um tipo de variável noutra tipo
 - ▶ A conversão em float da variável inteira, permite-nos ter o resultado correto (uma divisão com resultado decimal)

```
#include <stdio.h>
int main ()
{
    int num;
    float f;
    num=10;
    f=(float)num/7;
    printf ("%f", f);
    return(0);
}
```

- ▶ Faça a compilação e experimente o programa com e sem *cast*. Que diferenças observa?

Auto-avaliação

- Realize a auto-avaliação
- Qual foi a sua nota (0-100)?

Estruturas de controlo de fluxo

■ Os comandos “*if ... else ...*”

```
include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num==10)
    {
        printf ("\n\nAcertou!\n");
        printf ("O numero e igual a 10.\n");
    }
    else
    {
        printf ("\n\nErrou!\n");
        printf ("O numero e diferente de 10.\n");
    }
    return(0);
}
```

Estruturas de controle de fluxo

■ Os comandos “*if ... else*” e “*else if ...*”

```
#include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num>10)
        printf ("\n\n0 numero e maior que 10");
    else if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("0 numero e igual a 10.");
    }
    else if (num<10)
        printf ("\n\n0 numero e menor que 10");
    return(0);
}
```

Estruturas de controle de fluxo

■ Comandos aninhados

```
#include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.\n");
    }
    else
    {
        if (num>10)
        {
            printf ("O numero e maior que 10.");
        }
        else
        {
            printf ("O numero e menor que 10.");
        }
    }
    return(0);
}
```

Ciclos

■ O comando “for”

```
#include <stdio.h>
int main ()
{
    int count;
    for (count=1; count<=100; count++) printf ("%d ",count);
    return(0);
}
```

■ O ciclo infinito

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int Count;
    char ch;
    for (Count=1;;Count++)
    {
        ch = getch();
        if (ch == 'X') break;
        printf("\nLetra: %c",ch);
    }
    return(0);
}
```

Ciclos

■ O ciclo sem conteúdo: *criar tempo de espera...*

```
#include <stdio.h>
int main ()
{
    long int i;
    printf("\a");
    for (i=0; i<100000000; i++);
    printf("\a");
    return(0);
}
```

/* Imprime o caracter de alerta (um beep) */
/* Espera 10.000.000 de iteracoes */
/* Imprime outro caracter de alerta */

■ O comando “while”

```
#include <stdio.h>
int main ()
{
    char Ch;
    Ch='\0';
    while (Ch!='q')
    {
        Ch = getch();
    }
    return(0);
}
```


Vetores

■ Declaração (estática) de vetor

- ▶ Um vetor é declarado com determinada dimensão para que, na sua execução, se crie espaço para armazenar os valores a ser introduzidos.

```
#include <stdio.h>
int main ()
{
    int num[100]; /* Declara um vetor de inteiros de 100 posicoes */
    int count=0;
    int totalnums;
    do
    {
        printf ("\nEntre com um numero (-999 p/ terminar): ");
        scanf ("%d",&num[count]);
        count++;
    } while (num[count-1]!=-999);
    totalnums=count-1;
    printf ("\n\n\n\t Os números que você digitou foram:\n\n");
    for (count=0;count<totalnums;count++)
        printf (" %d",num[count]);
    return(0);
}
```

Vetores

■ Chamada de atenção sobre os vetores

● Para o comando *float exemplo [20];*

- ▶ O C irá reservar $4 \times 20 = 80$ bytes.

Estes bytes são reservados de maneira contígua.

- ▶ Na linguagem C a numeração começa sempre em zero:

```
exemplo[0]  
exemplo[1]  
.  
.  
.  
exemplo[19]
```

- ▶ Mas ninguém o impede de escrever:

```
exemplo[30]  
exemplo[103]
```

- O C não verifica se o índice que você usou está dentro dos limites válidos.
- Se o programador não tiver atenção com os limites de validade para os índices ele corre o risco de ter variáveis sobre-escritas ou de ver o computador parar.

Exercício 2

- Reescreva o exemplo da página 17, realizando a cada leitura um teste para ver se a dimensão do vetor não foi ultrapassada.
 - ▶ Caso o utilizador entre com 100 números, o programa deverá abortar o *loop* de leitura automaticamente.
 - ▶ O uso da *Flag* (-999) não deve ser retirado.

Strings

- *strings* são vetores de caracteres. Elas têm como último elemento o caracterer `'\0'` (*scape character*) que dá instrução para mudar de linha.

char nome_da_string [tamanho];

- As strings não são igualadas com a instrução
string1=string2
 - ▶ São apontadores (ver próxima secção)
 - ▶ Têm de ser igualadas elemento a elemento.

```
#include <stdio.h>
int main ()
{
    int count;
    char str1[100],str2[100];
    ....
    /* Aqui o programa le str1 que sera copiada para str2 */
    for (count=0;str1[count];count++)
        str2[count]=str1[count];
    str2[count]='\0';
    /* Aqui o programa continua */
    ....
}
```

Strings

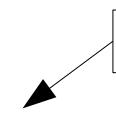
■ Funções básicas de manipulação de *strings*:

● **fgets()**

- ▶ Formato: *fgets (nome_da_string);*

```
#include <stdio.h>
int main ()
{
    char s[100];
    printf ("Digite o seu nome: ");
    fgets (s,100,stdin);
    printf ("\n\n Ola %s",s);
    return(0);
}
```

res = input("Digite o seu nome")

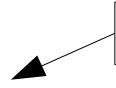


● **strcpy()**

- ▶ Formato: *strcpy (string_destino,string_origem);*

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char str1[100],str2[100],str3[100];
    printf ("Entre com uma string: ");
    fgets (str1,100,stdin);
    strcpy (str2,str1); /* Copia str1 em str2 */
    strcpy (str3,"Voce digitou a string "); /* Copia "Voce digitou a string" em str3 */
    printf ("\n\n%s%s",str3,str2);
    return(0);
}
```

str2 = str1[:]

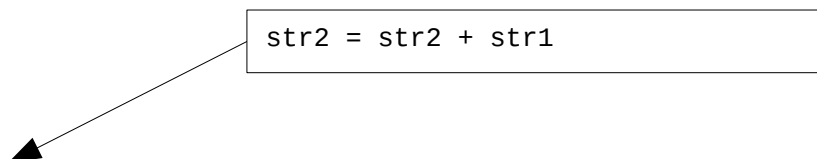


Strings

- **strcat(), strncat()**

- ▶ *Formato: strcat (string_destino,string_origem);*

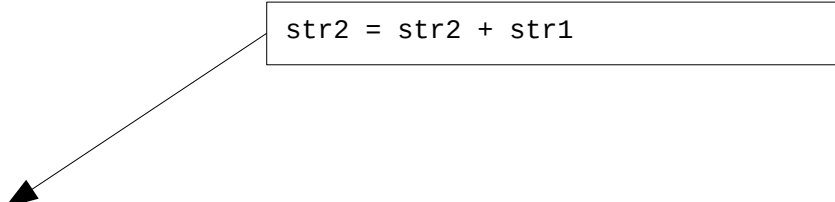
```
#include <stdio.h>
#include <string.h>
int main ()
{
    char str1[100],str2[100];
    printf ("Entre com uma string: ");
    fgets (str1,100,stdin);
    strcpy (str2,"Voce digitou a string ");
    strcat (str2,str1);      /* str2 armazenara' Voce digitou a string + o conteudo de str1 */
    printf ("\n\n%s",str2);
    return(0);
}
```



- **strlen()**

- ▶ *Formato: strlen (string);*

```
#include <stdio.h>
#include <string.h>
int main ()
{
    int size;
    char str[100];
    printf ("Entre com uma string: ");
    fgets (str,100,stdin);
    size=strlen (str);
    printf ("\n\nA string que voce digitou tem tamanho %d",size);
    return(0);
}
```



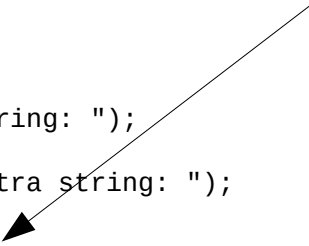
Strings

- **strcmp()**

- ▶ Formato: *strcmp (string1,string2);*

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char str1[100],str2[100];
    printf ("Entre com uma string: ");
    gets (str1);
    printf ("\n\nEntre com outra string: ");
    gets (str2);
    if (strcmp(str1,str2))
        printf ("\n\nAs duas strings são diferentes.");
    else printf ("\n\nAs duas strings são iguais.");
    return(0);
}
```

if (str2 == str1):
...



Exercício 3

- Faça um programa que leia quatro palavras do teclado e armazene cada palavra numa *string*. Depois, concatene todas as *strings* lidas numa única *string*. Por fim apresente esta como resultado ao final do programa.

Matrizes bidimensionais

■ Declaração de matrizes:

tipo_da_variável nome_da_variável [altura][largura];

```
#include <stdio.h>
int main ()
{
    int mtrx [20][10];
    int i,j,count;
    count=1;
    for (i=0;i<20;i++)
        for (j=0;j<10;j++)
        {
            mtrx[i][j]=count;
            count++;
        }
    return(0);
}
```

Exercício 4

- O que imprime o programa?

```
# include <stdio.h>
int main()
{
    int t, i, M[3][4];
    for (t=0; t<3; ++t)
        for (i=0; i<4; ++i)
            M[t][i] = (t*4)+i+1;

    for (t=0; t<3; ++t)
    {
        for (i=0; i<4; ++i)
            printf ("%3d ", M[t][i]);
        printf ("\n");
    }
    return(0);
}
```

Apontadores

- A linguagem C é altamente dependente dos ponteiros. Para ser um bom programador em C é fundamental que se tenha um bom domínio deles.
 - Por isto, recomenda-se “um carinho especial” com esta parte da linguagem.
 - Os Ponteiros são tão importantes na linguagem C que já os manipulámos mas nem nos apercebemos, pois mesmo para se fazer um introdução básica à linguagem C eles são necessários.

;-)

Apontadores

- Para declarar um apontador (ou ponteiro em PT-BR)

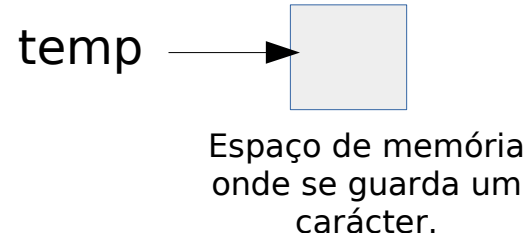
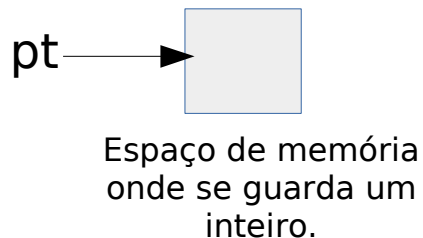
*tipo_do_ponteiro *nome_da_variável;*

- ▶ *Exemplo*

*int *pt;*

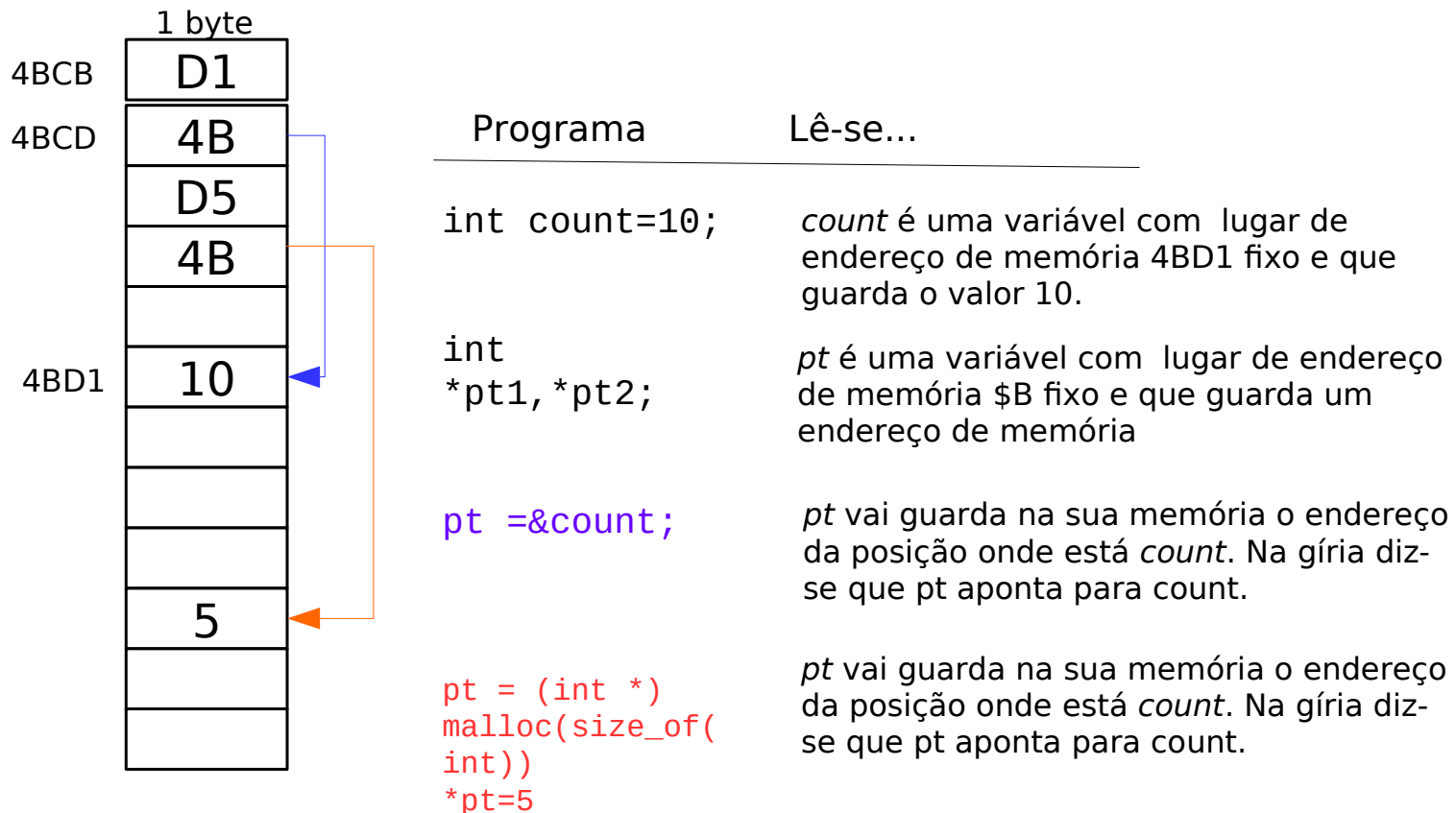
*char *temp, *pt2;*

- O primeiro declara um apontador para um inteiro
- O segundo declara dois apontadores, ambos apontam para um carácter.
- O nome apontador vem do facto de poderemos dizer que uma variável apontador, aponta para um local de memória:
 - *pt* aponta para um local de memória onde vai guardar um inteiro.
 - *temp* aponta para um local de memória onde se vai guardar um carácter.



Apontadores

- Um apontador guarda uma referência para um endereço de memória.
 - ▶ Mas ele próprio está num endereço de memória!



Apontadores

- Que valores são impressos? Experimenta.

```
#include <stdio.h>
int main ()
{
    int num;
    int *p;
    num=55;
    p=&num;      /* Pega o endereco de num */
    printf ("Endereco para onde o ponteiro aponta: %p\n",p);
    printf ("Valor da variavel apontada: %d\n",*p);
    Return(0);  * Retorna 0: tudo bem *
}
```

valor = ***p** lê-se

“valor recebe o conteúdo da memória para onde *p* aponta”

p = **&num** lê-se

“*p* aponta para o endereço da variável num.”

Apontadores

■ Outro exemplo

```
#include <stdio.h>
int main ()
{
    int num, *p;
    num=55;
    p=&num;      /* Pega o endereco de num */
    printf ("\nValor inicial: %d\n", num);
    *p=100; /* Muda o valor de num de uma maneira indireta */
    printf ("\nValor final: %d\n", num);
    return(0);
}
```

p = &num lê-se

“O apontador p aponta para (o endereço da) a variável num.”

****p = 100*** lê-se

“o espaço de memória para onde p aponta vai receber o valor 100”. Logo a variável num fica com que valor?

Apontadores

■ No seguimento do exemplo anterior

```
#include <stdio.h>
int main ()
{
    int num, *p1, *p2;
    num=55;
    p1=&num;      /* Pega o endereco de num */
    printf ("\nValor inicial: %d\n",num);
    *p1=100; /* Muda o valor de num de uma maneira indireta */
    p2 = p1
    printf ("\nValor final: %d\n",num);
    return(0);
}
```

p1 = &num lê-se

“O apontador p1 vai apontar para a variável *num*”

p1* = 100 lê-se

“o apontador p1 vai guardar no local para onde aponta o valor 100”

p2 = p1 lê-se

“o apontador p2 aponta para o local para onde aponta p1”

Apontadores

■ No seguimento do seguimento do exemplo anterior

```
#include <stdio.h>
int main ()
{
    int num,*p1, *p2, *p3;
    num=55;
    p1=&num;      /* Pega o endereco de num */
    printf ("\nValor inicial: %d\n",num);
    *p1=100; /* Muda o valor de num de uma maneira indireta */
    p2 = (int *)malloc(sizeof(int));
    *p2 = *p1;
    printf ("\nValor final: %d\n",num);
    return(0);
}
```

p1 = &num lê-se

“O apontador p1 vai apontar para a variável *num*”

p1* = 100 lê-se

“o apontador p1 vai guardar no local para onde aponta o valor 100”

p2 = (int *) malloc(sizeof(int)) lê-se

“Cria espaço de memória do tamanho de um inteiro e coloca o p2 a apontar para ele”

****p2 = *p1*** lê-se

“Atribuo ao espaço de memória para onde p2 aponta, o valor da memória para onde p1 aponta”

Exercício 5

■ Responda às questões

- Explique a diferença entre

`p++;` `(*p)++;` `*(p++);`

- ▶ O que quer dizer `*(p+10);`?
- ▶ Explique o que você entendeu da comparação entre apontadores

- Qual o valor de `y` no final do programa?

Tente primeiro descobrir e depois verifique no computador o resultado. A seguir, escreva um `/* comentário */` em cada comando de atribuição explicando o que ele faz e o valor da variável à esquerda do `'='` após sua execução.

```
int main()
{
    int y, *p, x;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    x--;
    (*p) += x;
    printf ("y = %d\n", y);
    return(0);
}
```

Nota sobre apontadores em estruturas

- A linguagem C permite definir estruturas de dados mais complexas.

- Por exemplo

```
struct dados_aluno{  
    int numero;  
    char nome[50];  
}
```

- Definida esta estrutura, posso definir uma variável

```
struct dados_aluno Aluno;
```

- Posso introduzir os dados nesta estrutura

```
Aluno.numero = 1023;  
strcpy(Aluno.nome,"António Martins");
```

- Posso imprimir resultados

```
printf("O aluno com o número %d tem o nome  
%s",Aluno.numero,Aluno.nome);
```

Nota sobre apontadores em estruturas

- Posso criar um vetor de alunos

```
struct dados_aluno Alunos[50];
```

- Vou introduzir os dados nesta estrutura

```
Aluno[0].numero = 1023;
```

```
strcpy(Aluno[0].nome,"António Martins");
```

```
Aluno[1].numero = 1024;
```

```
strcpy(Aluno[1].nome,"Miguel Soares");
```

- Vou imprimir os dados numa função

```
void imprime_alunos ( struct dados_aluno *pAlunos, int nr_alunos){
```

```
    for (int i=0;i<nr_alunos;i++){
```

```
        printf("Aluno com número %d tem o nome %S",
```

```
            pAlunos→numero,pAlunos→nome);
```

```
        pAlunos ++;
```

```
    }
```

Exemplo

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

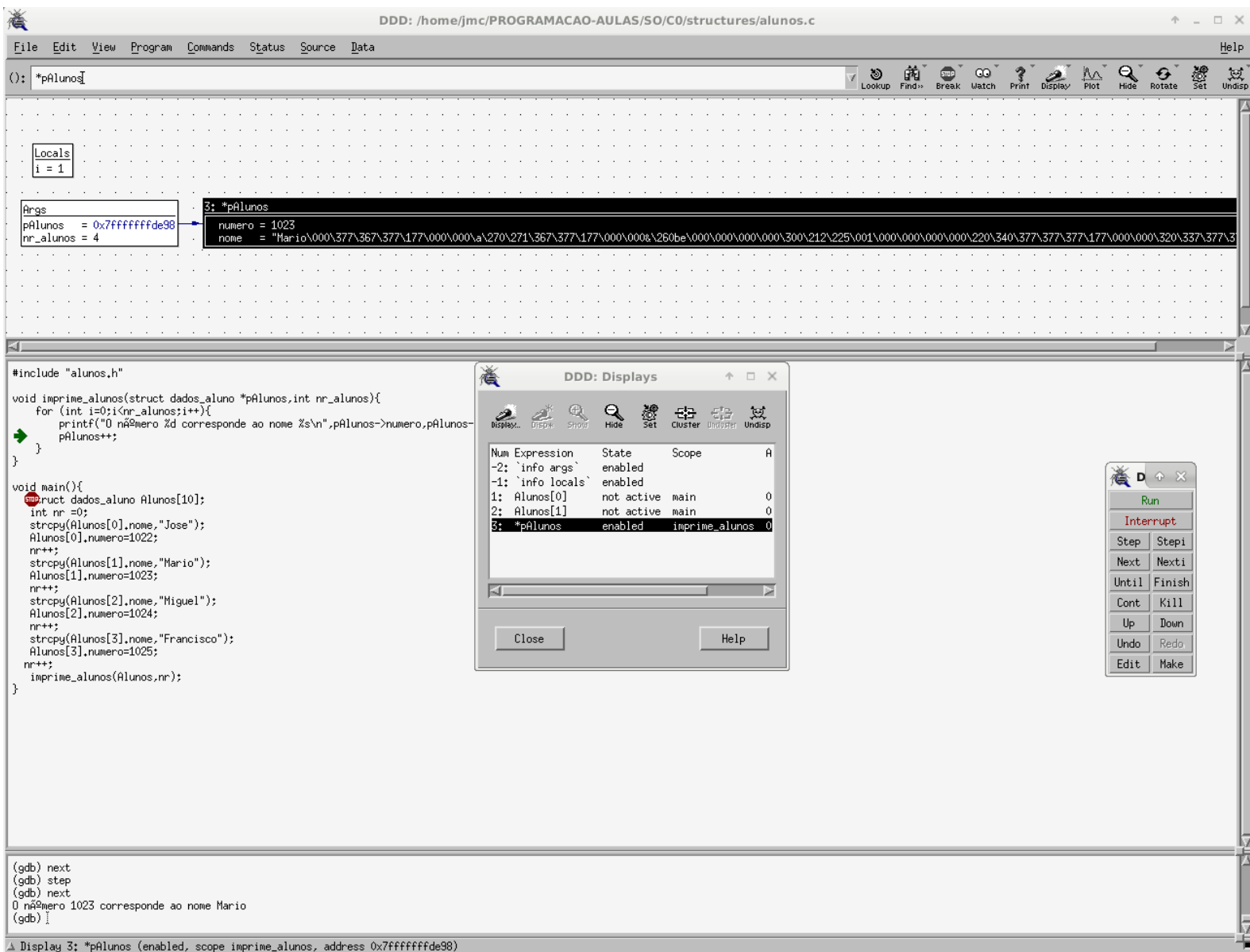
```
struct dados_aluno{
    int numero;
    char nome[50];
};
```

```
void imprime_alunos(struct dados_aluno *pAlunos, int nr_alunos);
```

Exemplo

```
include "alunos.h"
void imprime_alunos(struct dados_aluno *pAlunos,int nr_alunos){
    for (int i=0;i<nr_alunos;i++){
        printf("O número %d corresponde ao nome %s\n",pAlunos->numero,pAlunos->nome);
        pAlunos++;
    }
}
void main(){
    struct dados_aluno Alunos[10];
    int nr =0;
    strcpy(Alunos[0].nome,"Jose");
    Alunos[0].numero=1022;
    nr++;
    strcpy(Alunos[1].nome,"Mario");
    Alunos[1].numero=1023;
    nr++;
    strcpy(Alunos[2].nome,"Miguel");
    Alunos[2].numero=1024;
    nr++;
    strcpy(Alunos[3].nome,"Francisco");
    Alunos[3].numero=1025;
    nr++;
    imprime_alunos(Alunos,nr);
}
```

Exemplo



Exercício 6

- Crie um programa que receba o nome de 10 alunos e a sua idade. O número dos alunos é numerado a partir de 100 de forma sequencial. Esses dados são guardados num vetor de uma estrutura que contém 3 campos: o número, o nome e a idade do aluno.
O programa tem que ter uma função que imprime os dados.