Apontadores: Definição e declaração de apontadores; operações elementares sobre apontadores

Definição e declaração de apontadores

Um apontador é uma variável cujo objetivo é armazenar o endereço de outra variável, o qual é, por sua vez um número.
Um apontador, por armazenar um número (o endereço de outra variável), ocupa algum espaço em memória, pelo que tem de ser declarado.

Sintaxe:

```
tipo *ptr;
```

Sendo:

ptr o nome da variável do tipo apontador;
tipo o tipo de variável para a qual apontará;
***** o operador conteúdo indica que ptr é uma variável do tipo apontador (só pode ser usado neste tipo de dados!).

Notas:

- a declaração de apontadores pode ser realizada no meio de outras variáveis do mesmo tipo;
- O símbolo ***** usado na declaração de apontadores é o mesmo que é utilizado para a operação de multiplicação, o seu significado só depende do contexto em que é usado;
- Uma vez declarado, podem ser realizadas, sobre o apontador, praticamente todas as operações que podemos realizar sobre inteiros. No entanto, este serve, sobretudo, para permitir aceder a outros objetos, através dos seus endereços;
- O * pode ser colocado onde quisermos, desde que antes da variável do tipo ponteiro e após a declaração do tipo.

Exemplo: char a, b, *p, c; int * idade;

A inicialização de apontadores faz-se através do operador endereço de - &

O operador endereço & só pode ser usado numa única variável. Não pode ser usado em expressões como, por exemplo, &(x+y)
Este operador pode também ser utilizado para inicializar uma variável do tipo apontador, no momento da sua declaração.

Exemplo: float x=5, *prt_x=&x ; char a,*p=&a;

Um bom hábito para evitar problemas de programação é proceder à inicialização dos apontadores.

Se quisermos que um apontador não aponte para nenhuma variável, colocamo-lo a apontar para **NULL** (representando o endereço de memória número 0).

Exemplo: int *ptr=NULL;

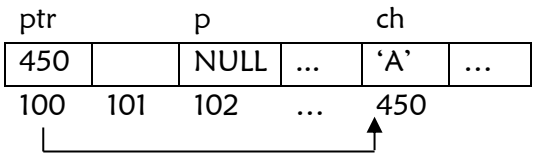
Exemplificação (resumo):

a)

```
#include<stdio.h>
main()
{
char ch='A', *ptr=&ch;  /*ptr aponta para ch*/

int *p=NULL;  /*p aponta aponta para NULL*/
...
```

Fig.1



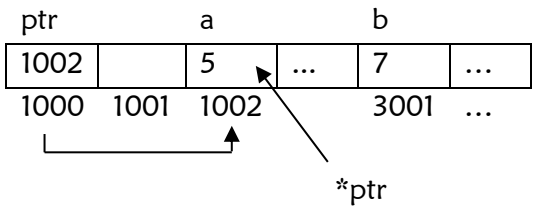
Expressão	Valor	Descrição
ch	'A'	Valor de ch
&ch	450	Endereço de ch
ptr	450	Valor de ptr
&ptr	100	Endereço de ptr
*ptr	'A'	Valor apontado por ptr

b)

```
#include <stdio.h>
main()
{
   int a=5, b=7, *ptr;
   ptr = &a;
   printf("%d",a);

/* é equivalente a printf("%d", *ptr);*/
…
```

fig.2



Expressão	Valor	Descrição
a	5	Valor de a
ptr	1002	Valor de ptr
ptr=&a	1002	ptr aponta para a
*ptr	5	Valor apontado por ptr, isto é, aquilo que está na casa para o qual aponta ptr, ou seja, o valor da variável cujo endereço está armazenado em ptr. Deve ler-se "o apontado por ptr"

Nota: Se declarássemos outro apontador denominado ptr2 e este apontasse também para a, era idêntico fazer: ptr2=&a; ou ptr2 = ptr;

EXERCÍCIOS:

1. Tendo em conta os dados anteriores responde às seguintes questões:

   a) Qual o output da seguinte instrução?
   printf("%d %d %d", a,b,*ptr);

   b) Qual o output depois de fazermos ptr = &b?

   c) E se executássemos a instrução

   *ptr = 20; Qual seria o output?

   d) Completa o quadro, tendo em conta os dados atuais:

Expressão	a	&a	b	&b	ptr	&ptr	*ptr
Valor							

2. Escreve um programa que ilustre a **fig.2** e de modo a imprimir os valores das expressões da tabela da alínea 1.d) com a respetiva descrição.

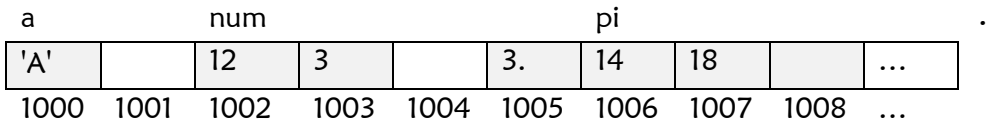
Operações elementares sobre apontadores

Apontadores e tipo de dados

As variáveis do tipo apontador, declaradas nos 2 primeiros exemplos, ocupavam apenas 1 byte de memória. Sabemos que os tipos de dados em C (char, int, float e double) ocupam diferentes n.ºs de bytes em memória, podendo, inclusive, o mesmo tipo ocupar um n.º diferente de bytes em diferentes arquiteturas, como é o caso do inteiro: 2 ou 4 bytes).
Quando declaramos as seguintes variáveis:

```
char a = 'A';        char ocupa 1 byte
int num = 123;       int ocupa 2 bytes (ou 4 bytes)
float pi = 3.1418;   float ocupa 4 bytes
```

Estamos a indicar ao compilador qual o espaço que este deverá reservar para cada uma das variáveis.



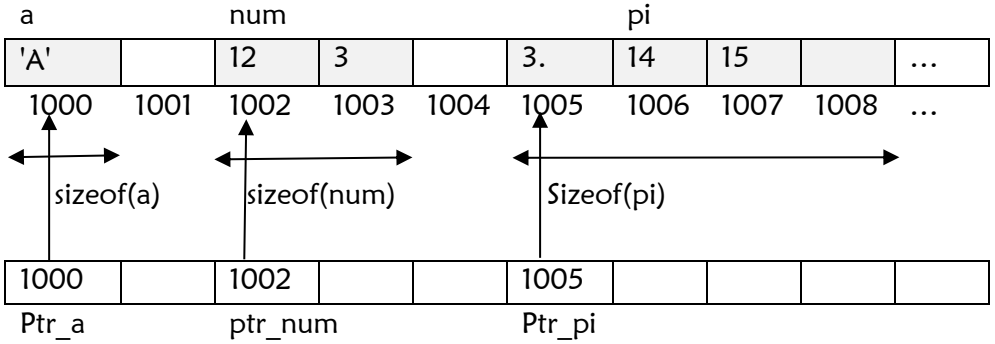
Um apontador para o tipo xyz, endereça sempre o nº de bytes que esse ocupa em memória, isto é, endereça sempre sizeof(xyz) bytes[1](aponta sempre para o menor endereço)

1

```
char *ptr_a = &a;  /*considera sizeof(char) bytes a partir do endereço contido em ptr_a */

int *ptr_num = &num;    /*considera sizeof(int) bytes a partir do endereço contido em
                          ptr_num */

float *ptr_pi = &pi;        /*considera sizeof(float) bytes a partir do endereço contido
                            em ptr_pi */
```



Incremento

Se **ptr** é um apontador para um determinado tipo, quando **ptr** é **incrementado**, por exemplo de 1 unidade, o endereço que passa a conter é igual **ao endereço anterior de ptr MAIS sizeof(tipo) para que o apontador aponta**, isto é, **o apontador avança**, não 1 byte mas sim, **a dimensão do tipo do objeto para o qual aponta**.

> **Um apontador para o tipo xyz avança sempre sizeof(xyz) bytes por cada unidade de incremento**

Exemplo

```
#include <stdio.h>
main()
{
   int x = 5, *p_x = &x;
   float y = 5.0, *p_y = &y;
   printf("%d    %d\n", x, p_x);
   printf("%d   %d\n", x+1, p_x+1);
   printf("%.2f   %d\n", y,  p_y);       /* o endereço de uma variável é sempre um nº inteiro! */
   printf("%.2f   %d\n", y+1, (p_y+1));
   getchar();
}
```

Decremento

O decremento funciona da mesma forma que o incremento.

> **Um apontador para o tipo xyz recua sempre sizeof(xyz) bytes por cada unidade de decremento**

EXERCÍCIO:

3. Escreve um programa semelhante ao anterior que ilustre o decremento de apontadores.

Diferença

A operação de diferença entre 2 apontadores para elementos (só) do mesmo tipo permite saber quantos elementos existem entre um endereço e outro.

Por exemplo, o comprimento de uma string pode ser obtido através da diferença entre o endereço do caráter '\0' (que indica o fim da string) e o endereço do 1º caráter.

Comparação

É também possível a comparação de 2 apontadores para o mesmo tipo, utilizando os operadores relacionais (>, >=, <, <=, == e !=).

Exemplos:

```
 …
int *px, *py;
…
If (px==py)          */ se px aponta para o mesmo endereço que py ….*/

If (px > py)          */ se px aponta para um bloco posterior a py ….*/

if (px!=py)          */ se px aponta para um bloco diferente de py ….*/

if (px ==NULL)       */ se px é nulo ….*/
```

RESUMO

Operação	Exemplo	Observações
Atribuição	ptr = &x	Podemos atribuir um valor (endereço) a um apontador. Para apontar para nada atribuímos-lhe o valor da constante NULL
Incremento	ptr+=2	Incremento de 2 x sizeof (tipo) da variável apontada por ptr
Decremento	Ptr-=ptr	decremento de 1 x sizeof (tipo) da variável apontada por ptr
Apontado por	*ptr	Permite obter o valor existente na posição cujo endereço está armazenado em ptr
Endereço de	&ptr	Endereço que o apontador ptr ocupa em memória
Diferença	ptr1-ptr2+1	N.º de elementos entre ptr1 e ptr2 (diferença entre os endereços)
Comparação	ptr1>ptr2	Permite, por exemplo, verificar a ordem de 2 elementos num vetor, através do valor dos seus endereços.

EXERCÍCIOS

4. Indica se são verdadeiras ou falsas as seguintes afirmações. Justifica.

a) O operador & permite obter o endereço de uma variável e até de um apontador;

b) O endereço de memória que ocupa mais que 1 byte de memória é o seu menor endereço;

c) O operador * permite saber qual o valor de um apontador;

d) NULL é um outro nome para o caráter de fim de string '/0'.

e) Se x é um inteiro e p um apontador para inteiros e ambos contêm no seu interior o número 100, então x+1 e p+1 seriam iguais a 101.

5. Responde, sucintamente, às seguintes questões.

a) Qual o operador que nos permite obter o endereço de uma variável?

b) Qual o caráter que se coloca na declaração de uma variável para indicar que é um apontador?

c) Onde se coloca esse caráter?

d) Qual o símbolo, que podemos colocar num apontador, para indicar que este não aponta para nada?

e) O que contém uma variável do tipo apontador?

f) Se ptr for um apontador qual o valor de *(&ptr)?

g) Se p1 de p2 forem dois apontadores para um vetor então p2-p1-1 indica o número de elementos entre p1 e p2?

h) Como declaramos uma variável k que tenha a capacidade de conter o endereço de outra variável do tipo float?

6. Considera a seguinte declaração:

```
int x=2, *px, *py, y=3;
```

A que corresponde o seguinte esquema de memória.

x	px	py	y			
2			3			
100	101	102	103	104	105	106

Supõe que a escrita de inteiros e apontadores se pode fazer através da função *printf* , usando o formato %d.

a) Escreve o código necessário para colocar **px** a apontar para **x** e **py** a apontar para **y**.

b) Depois de executada a alínea anterior, qual o output das seguintes instruções?

```
printf("%d   %d\n", x, y);
printf("%d   %d\n",*px, *py);
printf("%d   %d\n", &px, &py);
printf("%d   %d\n", px, py);
```

c) Se se fizer **px=py** qual o output de:

printf("%d %d %d %d %d %d %d %d ", x, &x, px, *px, y, &y, py, *py);