6 - APONTADOR (PONTEIRO)

Ponteiro ou apontador é uma variável que contém um endereço de memória, ou seja ela aponta para um endereço de memória.

Como toda variável, ela também corresponde a uma posição de memória, e precisa de um identificador.

6.1 - Variáveis ponteiro

- Apontam para um determinado tipo base.
- Declaração: <tipo_base> *nome;

<nome> aponta (armezena o endereço) para uma variável do tipo <tipo_base>.

- Exemplo:

Seja a seguinte declaração:

```
char *pc;
```

- pc é um ponteiro para um caracter, isto é, serve para guardar o endereço de um caracter.
- É bom enfatizar que o elemento a ser apontado por pc, ainda não existe, então pc não possui valor

6.2 - Operadores de ponteiro

Existem dois operadores especiais para ponteiros: **&** e *****

6.2.1 - Operador de endereço

&: devolve o endereço de memória de seu operando.

- Exemplo:

```
char  *pi,   a;
```

Após a declaração (supor que caracter **c** ocupe a posição 28 da memória):

Sejam as seguintes atribuições:

```
a = 'x';
```


```
pi = &a;
```

6.2.2 - Operador indireto (dereferenciação)

***** acessa o conteúdo das variáveis apontadas.

- Exemplo 1:

Seja o exemplo anterior:

Os seguintes comando são equivalentes:

printf ("% c ", a);

printf ("% c ", *pi);

Ambos produzem a mesma saída: x

- Observações:
  - Os operadores & e * têm precedência sobre os operadores aritméticos.
  - As variáveis ponteiros devem apontar para o tipo de dado correto, ou seja, se um ponteiro é declarado como int, deve sempre receber o endereço de um inteiro; caso contrário, pode produzir efeitos indesejáveis.

- Exemplo 2: Seja o programa abaixo:

```
main()
{ char  *pc, c, x;

  c = 'A';

  pc = &c;

  x = *pc;

  printf ("% c", c );          produz a saída:  A
  printf ("% c",  x);          produz a saída:  A
  printf ("% c",  *pc );       produz a saída:  A
  printf ("% p", pc );         produz a saída:  1000
  printf ("% p", &c);          produz a saída:  1000
}
```

	pc	c	x
		1000	
		'A'	
	1000	'A'	
	1000	'A'	'A'

6.3 - Expressões com ponteiros

Expressões com ponteiros incluem atribuição, aritmética e comparação.

6.3.1 - Atribuição

- Atribuições são feitas como qualquer outra variável.
- Uma variável ponteiro pode receber o conteúdo de outra variável ponteiro (certifique-se que eles apontam para o mesmo tipo!)

- Exemplo:

Seja o programa abaixo:

```
main()
{ int  *px, *pi, x;

    x = 300;

    px = &x;

    pi = px;
```

px	pi	x
	2000	
		300
2000		300
2000	2000	300

```
  printf ("% c", x );          produz a saída:  300

  printf ("% c",  *px);        produz a saída:  300

  printf ("% c",  *pi );       produz a saída:  300

  printf ("% p", &x );         produz a saída:  2000

  printf ("% p", px );         produz a saída:  2000

  printf ("% p", pi);          produz a saída:  2000
}
```

6.3.2 - Aritmética

- Existem duas operações aritméticas que podem ser feitas com ponteiros:

  soma e subtração.

- Pode-se usar os operadores de incremento e decremento (++ e --), bem como os operadores += e -= .

- O número inteiro que é somado a um ponteiro, corresponde a quantidade em bytes do tipo base do elemento.

- Exemplo:
  - Supor que o tipo inteiro ocupa 2 bytes.

```
main()
{ int  *px, x;

  px = &x;

*px =3;   /*ou x = 3*/
```

px	x
	2000
2000	
2000	3

  - O comando

  px= px + 1;

2000	
2002	300

  é equivalente a px ++; e a px+=1;

  Após este comando, px deixou de apontar para x.

  - O comando

  px= px + 2;

2000	
2004	300

  é equivalente a px += 2; supondo que px estava com o valor 2000.

  Após este comando, px deixou de apontar para x.

- Se for utilizado o operador * antes do ponteiro, então trabalha-se com o conteúdo do elemento apontado.

- Exemplo:
  - Seja a seguinte situação do exemplo anterior:

px	x
	2000
2000	3

  - O comando 2000

*px= *px + 1; → | 2000 | 4 |

incrementa o contéudo de x e é equivalente à

x = x + 1; ou a

*px + = 1; ou a

+ + *px; ou a

(*px)+ +;

- Deve-se tomar cuidado com a precedência dos operadores.
  - O comando ***px + +;** não é equivalente a nenhuma das expressões do exemplo anterior:
    (*px) + +; ≠ *px + +;
  - Em ***px + +;** é incrementado o endereço e não o conteúdo.
  - *px + 1 ≠ *(px + 1)

    conteúdo do elemento apontado + 1 — conteúdo do elemento da próxima posição de memória

- Exemplo:

Seja o programa abaixo:

```
main()
  { float x = 10,  y,  *px;

    px =  &x

    y = *px + 1

    px ++;
```

x	y	px
	1050	
10		
		1050
10	11	1050
10	11	1054

- Supor z uma variável real. O comando z = *px pode provoca um erro, pois não é conhecido o que ocupa a posição 1054 da memória.

- No exemplo acima, um real ocupa 4 bytes.