

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;
```



- Exemplo 2: Seja o programa abaixo:

```

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

  c = 'A';

  pc = &c;

  x = *pc;

  printf ("%c", c );
  printf ("%c", x);
  printf ("%c", *pc );
  printf ("%p", pc );
  printf ("%p", &c);
}

```

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

produz a saída: A
 produz a saída: A
 produz a saída: A
 produz a saída: 1000
 produz a saída: 1000

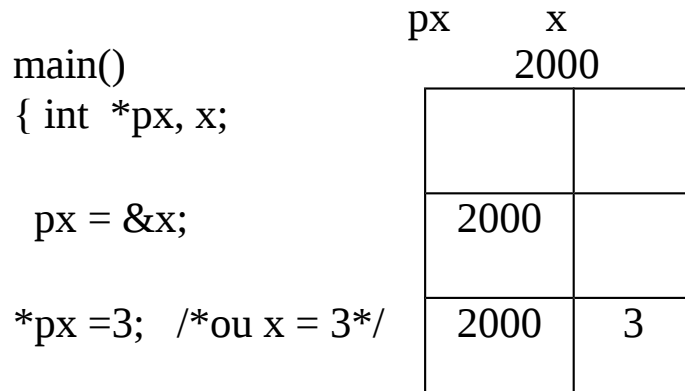
6.3 - Expressões com ponteiros

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!)

- 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.



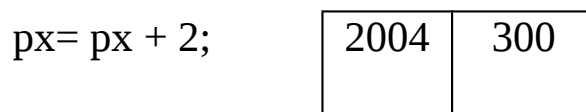
- O comando



é equivalente a `px ++;` e a `px+=1;`

Após este comando, `px` deixou de apontar para `x`.

- O comando



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 o programa abaixo:

	x	y	px
main()		1050	
{ float x = 10, y, *px;	10		
px = &x			1050
y = *px + 1	10	11	1050
px ++;	10	11	1054

- Supor z uma variável real. O comando `z = *px` pode provocar um erro, pois não é conhecido o que ocupa a posição 1054 da memória.
- No exemplo acima, um real ocupa 4 bytes.