Variáveis e Endereços

- Memória abstrata (Como vemos a memória):

  {x→5, y→9, z→'a'} **Id→Valor**

- Memória concreta:
  - Associações:

    {x→13, y→72, z→00} **Id→Endereço**

  - Memória de fato:

    {00→'a',...,13→5, **Endereço→Valor**

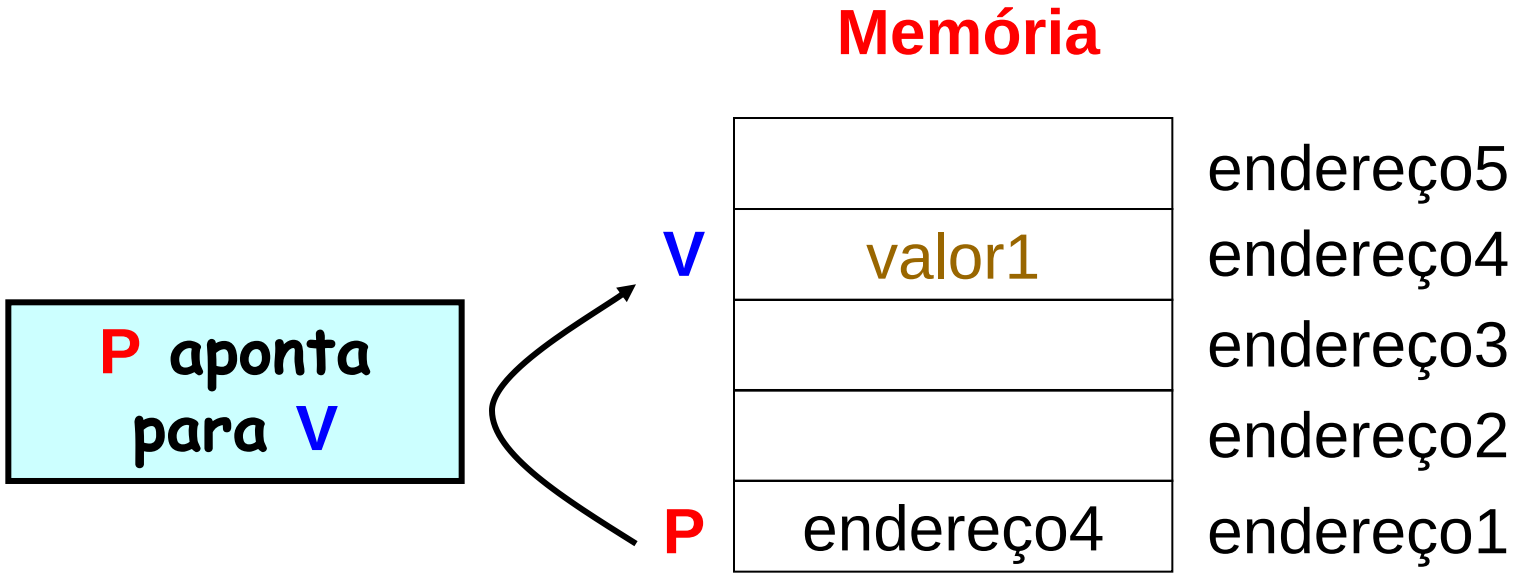
    72→9,...,99→*undefined*}

Ponteiros

- Toda variável tem um endereço ou uma posição associados na memória
- Este endereço é visto como um ponteiro (ou apontador), uma referência para a posição de memória de uma variável
- Ponteiros fornecem um modo de acesso à variável sem referenciá-la diretamente
- Um endereço pode ser armazenado em uma variável do tipo ponteiro (ponteiro variável)

Ponteiro Variável

- Um ponteiro variável é uma variável que contém o endereço de outra variável



P = endereço da variável V

Declarando Variáveis do Tipo Ponteiro em C

```
int b;
```

Declara uma variável de nome `b` que pode armazenar valores inteiros

- Para declarar uma variável do tipo ponteiro:

Forma Geral:

```
tipo* variavel
```

```
int* p;
```

Declara uma variável de nome `p` que pode armazenar um endereço de memória para um inteiro

Operador &

- Operador unário que fornece o endereço de uma variável

Forma Geral:

&variavel

```
int *p;
int v;
p = &v;
```

Variável p de tipo ponteiro para inteiro recebe endereço da variável v de tipo inteiro

- Não pode ser aplicado a expressões ou constantes
  - Ex: x = &3;

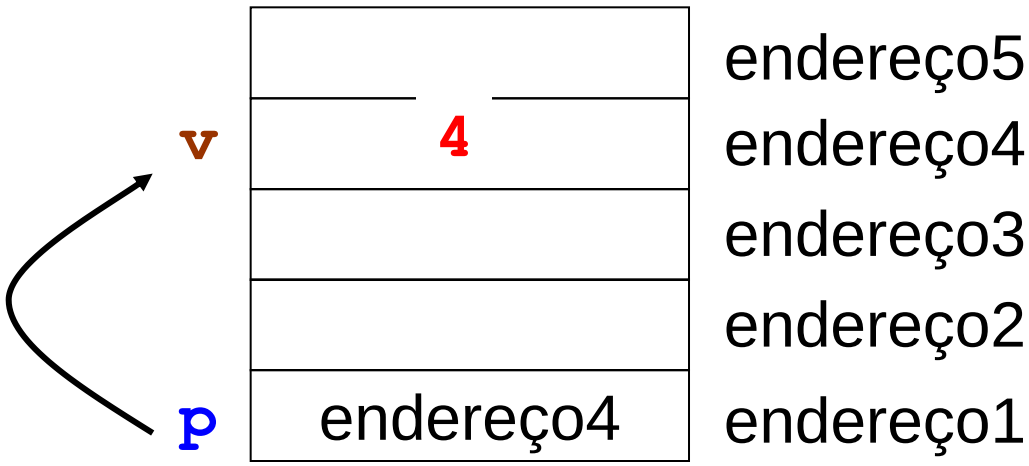
ERRADO!

Operador de Indireção *

- Quando aplicado a uma variável do tipo ponteiro, acessa o conteúdo do endereço apontado por ela

Forma Geral:
***variavel**

```
int *p;
int v = 3;
p = &v;
*p = 4;
```



Usando Ponteiros

```
int i, j;

int *ip;

i = 12;

ip = &i;

j = *ip;

*ip = 21;
```

A variável **ip** armazena um ponteiro para um inteiro

O endereço de **i** é armazenado em **ip**

O conteúdo da posição apontada por **ip** é armazenado em **j**

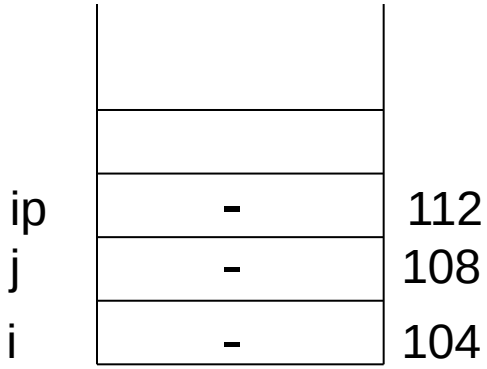
O conteúdo da posição apontada por **ip** passa a ser **21**

Usando Ponteiros

1)

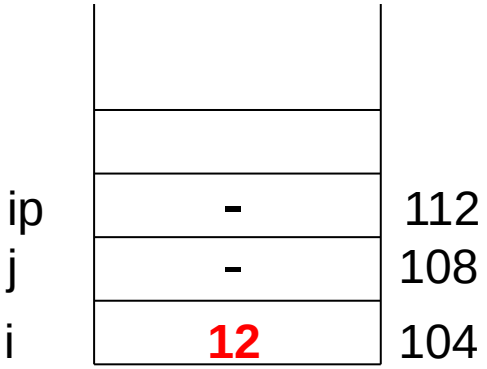
```
int  i,j ;

int  *ip ;
```



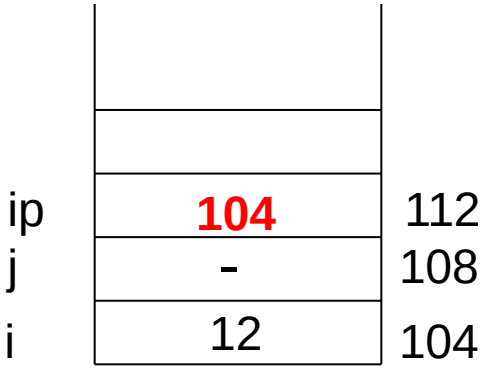
2)

```
i = 12 ;
```



3)

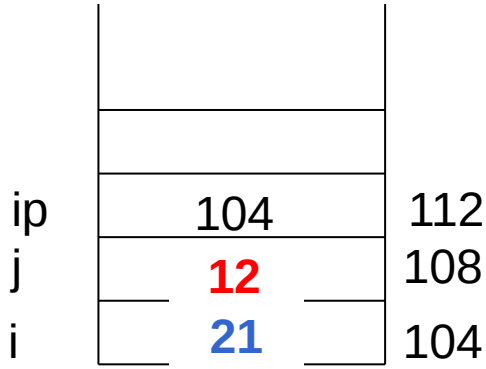
```
ip = &i ;
```



4)

```
j  = *ip;

*ip  = 21;
```



Manipulando Ponteiros

```
int main  () {  /* função  principal */
   int   a , *p ;
   p = &a ;
   *p = 2 ;
   printf ("%d",a);
   return  0 ;
}
```

Imprime o valor 2

```
int  main  () {      /* função principal*/
   int a,b,*p ;
   a = 2 ;
   *p = 3 ;
   b = a +( *p );
   printf("%d",b);
   return 0;
}
```

Erro típico de manipulação de ponteiros - ponteiro não inicializado!

Funções e Ponteiros

- Retorno explícito de valores não permite transferir mais de um valor para a função que chama

```
# include <stdio.h>
void somaprod(int a, int b, int c, int d) {
    c  =  a  +  b ;
    d  =  a  *  b ;
}

int main () {
    int s,p ;
    somaprod(3,5,s,p) ;
    printf("Soma = %d e Produto = %d \n",s,p);
    return  0 ;
}
```

Esse código não funciona como esperado !

A Passagem de Argumentos em C é por valor...

```
int a,b;
a = 8;
b = 12;
swap(a,b);
```

A chamada da função não afeta os valores de **a** e **b**

Copia os valores que estão em **a** e **b** para parâmetros **x** e **y**

```
void swap(int x, int y){
   int temp;
   temp = x;
   x = y;
   y = temp;
}
```

Mas C Permite a Passagem por Referência

```
int a,b;
a = 8;
b = 12;
swap(&a,&b);
```

A chamada da função **afeta** os valores de **a** e **b**

Copia os endereços de **a** e **b** para parâmetros **px** e **py**

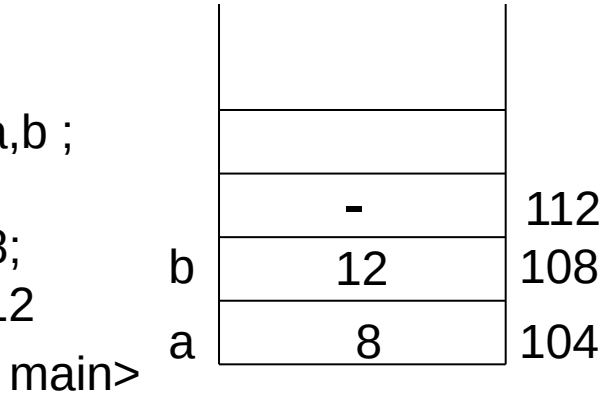
```
void swap(int* px, int* py){
   int temp;
   temp = *px;
   *px = *py;
   *py = temp;
}
```

Passagem por Referência em C

1)

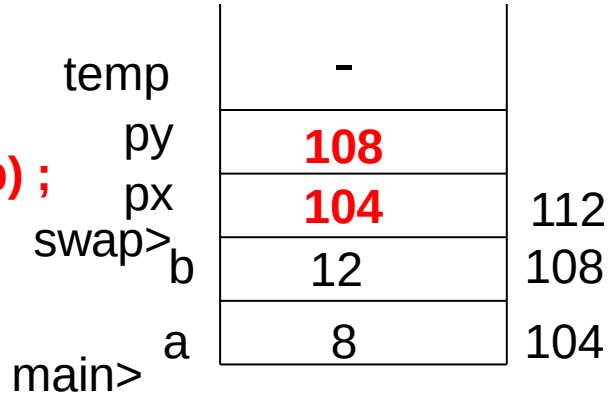
```
int  a,b ;

a = 8;
b = 12
```



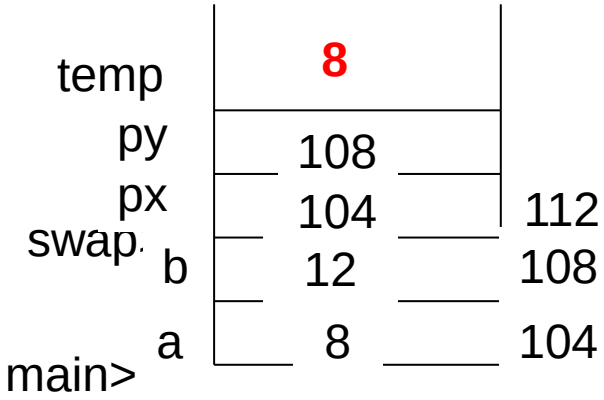
2)

swap(&a,&b) ;



3)

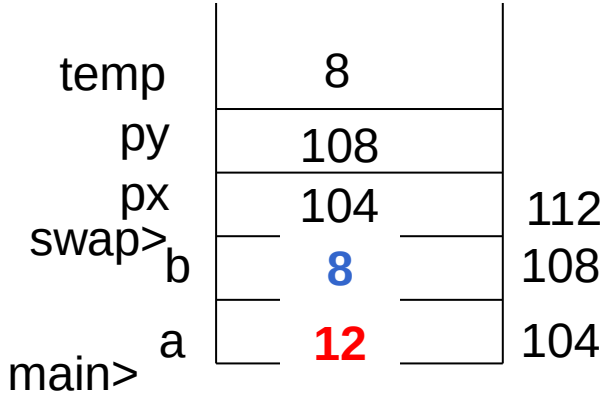
temp = *px;



4)

***px = *py;**

***py = temp;**



Passando endereços para uma função

- Como uma função pode alterar variáveis de quem a chamou?
  1) função chamadora passa os endereços dos valores que devem ser modificados
  2) função chamada deve declarar os endereços recebidos como ponteiros

Usando Passagem por Referência para Função *somaprod*

```
# include "stdio.h"
void somaprod(int a,int b,int* p, int* q){
     *p  =  a  +  b ;
     *q  =  a  *  b ;
}

int  main (){
    int   s ,  p ;
    somaprod (3 , 5 ,  &s ,  &p) ;
    printf("Soma= %d e Produto = %d \n",s,p);
    return  0 ;
}
```

Passagem por Referência

Por que ponteiros são usados ?

- Possibilitar que funções modifiquem os argumentos que recebem
- Manipular vetores e strings - útil para passar vetores como parâmetro
- Criar estruturas de dados mais complexas, como listas encadeadas, árvores binárias etc.

Operações com Ponteiros

```
int main( ) {
        int x=5, y=6;
        int *px, *py;
        px = &x;
        py = &y;
        if (px < py)
                printf("py-px = %u\n",py-px);
        else
                printf("px-py = %u\n",px-py);

        return 0;
}
```

px e py armazenam endereços para inteiros

Resultado: diferença entre endereços dividido pelo tamanho em bytes de um inteiro

Operações com Ponteiros

Se px = 65488 e
py = 65484

Saída será:
px - py = 1

Testes relacionais >=, <=, <, >, ==, são aceitos em ponteiros

A diferença entre dois ponteiros será dada na unidade do tipo de dado apontado

Operações com Ponteiros

```
int main( ) {
        int x=5, y=6;
        int *px, *py;
        px = &x;
        py = &y;
        printf("px = %u\n",px);
        printf("py = %u\n",py);
        py++;
        printf("py = %u\n",py);
        py = px+3;
        printf("py = %u\n",py);
}
```

Podemos utilizar operador de incremento com ponteiros

Podemos fazer aritmética de ponteiros

Operações com ponteiros

- O incremento de um ponteiro acarreta na movimentação do mesmo para o próximo valor do ***tipo apontado***
  - Ex: Se ***px*** é um ponteiro para int com valor 3000, depois de executada a instrução ***px++***, o valor de ***px*** será 3004 e não 3001 !!!
    - Obviamente, o deslocamento varia de compilador para compilador dependendo do número de bytes adotado para o referido tipo