IFSP
Piracicaba

Ponteiros em C

Parte 1

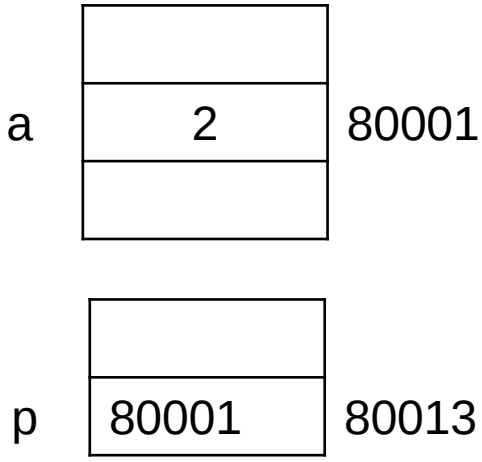
Marcio Kassouf Crocomo

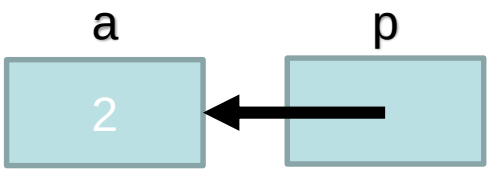
marciokc@ifsp.edu.br

IFSP
Piracicaba

Ponteiro

- É um tipo “especial” de variável. Armazena um endereço de memória, podendo armazenar o endereço de uma outra variável





IFSP
Piracicaba

Ponteiro

- Declaração
  - Tipo *nome_variável;
  - Exemplo: int *p; //declarando um ponteiro para um tipo inteiro

IFSP
Piracicaba

Operadores

- Dois operadores:
- * Interpretado como “no endereço”
- & Interpretado como “endereço de”

IFSP
Piracicaba

Atribuição

- Atribuição
  - Exemplo:

```
int a=5;
int *p;
p = &a; //ponteiro p recebe o endereço de a
```

- Declaração com inicialização
  - Exemplo:

```
int a=5;
int *p=&a;
```

IFSP
Piracicaba

Ponteiro

- Utilizando o conteúdo da variável apontada

Exemplo Leitura

```
int a = 2;
int b = 0;
int *p;
p = &a;
b = *p; //b passa a ter o conteúdo de “a”
```

Exemplo Escrita

```
int a = 2;
int b = 0;
int *p;
p = &a;
*p = b; //a passa a ter o conteúdo de b
```

IFSP
Piracicaba

Inicialização de ponteiros

- Ao declarar um ponteiro, ele contem um valor desconhecido. Acessar o endereço referenciado por um ponteiro sem inicializa-lo antes com algum valor pode quebrar seu programa! (ou pior!!!)

- Para representar que um ponteiro não aponta para nada, é atribuido ao mesmo o valor NULL (que é 0)

IFSP
Piracicaba

Ponteiro

- Uso:
  - Passagem por referência em Funções
  - Estruturas de dados (alocação dinâmica de memória
- Cuidado: Ponteiro perdido!

IFSP
Piracicaba

Passagem por referência em Funções

- A função “duplica” a seguir recebe o endereço de uma variável do tipo int, e então dobra o valor contido nessa variável.

IFSP
Piracicaba

Passagem por referência em Funções

```
void duplica(int *x)
{
    *x= 2*(*x);
}

int main()
{
    int num = 4;
    duplica(&num);
    printf("%d",num);
    return 0;
}
```

IFSP
Piracicaba

Exercício

- Complete a função a seguir para que a mesma faça com que a variável apontada por “menor” receba o menor valor entre “a” e “b” e a variável apontada por “maior” o maior valor.

```
void compara(float a, float b, float *menor, float*maior)
{
    /*completar função aqui*/
}
```

IFSP
Piracicaba

Ponteiros para ponteiros

- É possível criar ponteiros para ponteiros
- Exemplo:

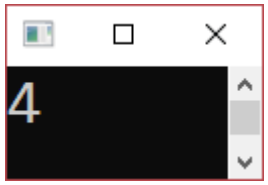
```
int a=5;
int *pi; /*ponteiro para inteiro*/
int **ppi; /*ponteiro para ponteiro para inteiro*/
pi=&a;
ppi=&pi;
**ppi=4;
printf(“%d”,a);
```

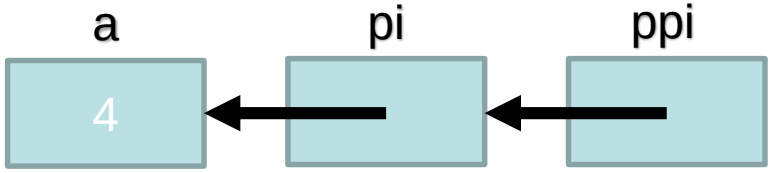
IFSP
Piracicaba

Ponteiros para ponteiros

- É possível criar ponteiros para ponteiros
- Exemplo:

```
int a=5;
int *pi; /*ponteiro para inteiro*/
int **ppi; /*ponteiro para ponteiro para inteiro*/
pi=&a;
ppi=&pi;
**ppi=4;
printf("%d",a);
```





IFSP
Piracicaba

Aritmética de ponteiros

- Duas operações possíveis:
- Adição
- Subtração

IFSP
Piracicaba

Aritmética de ponteiros

- Embora qualquer ponteiro de qualquer tipo possa apontar para qualquer lugar na memória, é importante definir o tipo correto do ponteiro por causa da aritmética de ponteiros.

IFSP
Piracicaba

Aritmética de ponteiros

- Considere o ponteiro para inteiro abaixo

int *p;

Caso p Aponte para o endereço de memória 5000, p+1 aponta para o próximo endereço de memória onde estaria um próximo inteiro. Assumindo que um inteiro ocupe 4 bytes na memória, p+1 aponta para o endereço 5004.

Alternativamente, após a instrução

p++;

p passa a ter o valor 5004

IFSP
Piracicaba

Aritmética de ponteiros

- Caso o ponteiro do exemplo anterior fosse para o tipo char como mostrado abaixo:

char *p;

Assumindo que p possua o valor 5000, p+1 possuiria o valor 5001, já que um char ocupa um byte na memória.

IFSP
Piracicaba

Ponteiros e vetores

- Existe uma forte relação entre ponteiros e vetores em C.

Quando um identificador de um vetor é utilizado sem os colchetes, o mesmo retorna o endereço inicial do vetor.

IFSP
Piracicaba

Ponteiros e vetores

- Podemos fazer:

```
int v[3] = {1,5,10};
int *p;
p = v;
```

- Considerando o que foi visto na aritmética de ponteiros, as expressões nas mesmas linhas abaixo se referem as mesmas posições na memória:

v[0]	*p
v[1]	*(p+1)
v[2]	*(p+2)

IFSP
Piracicaba

Alocação dinâmica de memória

- Estudar as funções malloc, para alocar memória dinamicamente, e free, para liberar memória.

- http://www.cplusplus.com/reference/cstdlib/malloc/
- http://www.cplusplus.com/reference/cstdlib/free/

IFSP
Piracicaba

referências

- SCHILDT, Herbert. **C Completo e Total**. 3. ed. São Paulo: Pearson Makron Books, 2004.