Linguagem C: aritmética de ponteiros

Prof. Críston

Algoritmos e Programação

Ponteiros e vetores

- A criação de um vetor segue os seguintes passos:
  - Pedido para o sistema operacional reservar uma quantidade de bytes consecutivos capaz de armazenar o vetor
    - Ex.: um vetor com 10 doubles ocupa 80 bytes consecutivos
  - Criação de um ponteiro com o nome do vetor
  - Inicialização deste ponteiro com o endereço do início da memória reservada pelo sistema operacional
- Ou seja, na verdade o identificador de um vetor é um ponteiro que armazena o endereço do primeiro elemento do vetor
- Para evitar problemas, o compilador não permite que o ponteiro que representa o vetor seja modificado, mas podemos copiar o endereço do vetor para outro ponteiro e modificá-lo

Exemplo

```
main()
{
    int v[] = {10, 20};
    printf("%d\n", v[0]);
    printf("%p\n", &v[0]);
// v é um ponteiro que armazena o valor de &v[0]
    printf("%p\n", v);
// o segundo elemento está 4 bytes depois
    printf("%p\n", &v[1]);
}
```

Atribuição

- Se p1 e p2 são ponteiros, então a atribuição

  p1 = p2;

  faz p1 apontar para o mesmo endereço apontado por p2

- Para fazer a região apontada por p1 ter o mesmo valor da região apontada por p2 devemos fazer

  *p1 = *p2;

Incremento e decremento

- Quando incrementamos um ponteiro estamos fazendo ele apontar para o próximo elemento do vetor
  - Ex.: se p é um ponteiro para inteiro (ou seja, declarado como int *p), e p armazena o endereço 100, então p++ faz p armazenar o endereço 104 (pois um inteiro ocupa 4 bytes)
  - Por esta razão precisamos indicar o tipo na declaração do ponteiro
- A mesma regra vale para decremento

- Para incrementar o conteúdo da região apontada por p fazemos (*p)++;

Soma/subtração de ponteiros com inteiros

- Para fazer um ponteiro p apontar para N elementos a frente fazemos p += N; ou p = p+N;
- Para acessar o conteúdo de N elementos a frente usamos *(p+N)
- A subtração funciona de forma similar

Exemplo

```
main()
{
    int v[] = {10, 20, 30};
    printf("%p\n", v);
    printf("%p\n", v+2);
    printf("%d\n", *(v+2));
}
```

Comparação de ponteiros

- Podemos utilizar os operadores relacionais para testar:
  - == apontam para a mesma posição
  - != apontam para posições diferentes
  - > >= < <= qual aponta para a posição mais alta

Exemplo – percorrendo matriz utilizando ponteiros

```
main()
{
    int v[] = {1, 2, 3, 4, 5, 6}, *p;
    for (p = v; p <= &v[5]; p++)
        printf("%d\n", *p);
}
```

Ponteiro para ponteiro

- Podemos declarar um ponteiro que guarda o endereço de memória de outro ponteiro

```
tipo **nome;
```

- Neste caso,
  - *nome é o conteúdo do ponteiro intermediário (um endereço)
  - **nome é o conteúdo da região final apontada (um valor)

- Podemos ter ponteiro para ponteiro para ponteiro... basta aumentar o número de asteriscos.

Exemplo

```
main()
{
    float pi = 3.1415, *pf, **ppf;
    pf = &pi;
    ppf = &pf;
    printf("%f\n", **ppf);
    printf("%f\n", *pf);
}
```

- Na aula sobre parâmetros da função main vamos ver uma aplicação de ponteiros para ponteiros