Professor
Edson Pedro FERLIN

Programação em C

Ponteiros

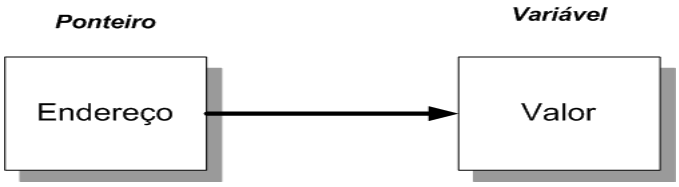
Prof. Edson Pedro Ferlin

Professor
Edson Pedro FERLIN

Programação em C

Definições

- Definição: é uma variável que contém um endereço de memória
- Esse endereço é a localização de uma outra variável de memória.



Variáveis Ponteiros

- Forma geral para a declaração:

  Tipo *nome_var;

- Exemplos:

```
char *p;
int *temp;
```

Obs: o tipo base do ponteiro define para que tipos de variáveis o ponteiro pode apontar.

Operadores de Ponteiros

- Dois operadores especiais:

  * - valor da variável;

  & - endereço da variável.

- São unários, requerem apenas um operando.

- Exemplos:

```
end_cont = &cont;  /* recebe o endereço de cont */
valor = *end_cont;  /* recebe o valor no endereço */
```

Obs: Assegurar que as variáveis com ponteiros sempre apontem para o tipo correto de dado.

Exemplo

```
main()
{  float x=10.1, y;
   int *p;
   p=&x;
   y=*p;
   printf ("%f", y);
}
```

Obs: Como declaramos que **p** é um ponteiro de inteiros, o compilador transferirá apenas dois bytes de informação para **y**, e não os quatro bytes que normalmente formam um número com ponto flutuante.

Expressões com Ponteiros
Atribuição

```
main()
{  int x;
   int *p1, *p2;
   p1=&x;
   p2=p1;
   printf ("%p", p2); /* imprime o valor HEX do endereço de x */
}
```

Pode-se usar um ponteiro no lado direito dos comandos de atribuição para atribuir seu valor a um outro ponteiro.

Expressões com Ponteiros
Aritmética (1)

- Apenas duas operações aritméticas são possíveis:
  - + (soma)
  - - (subtração)

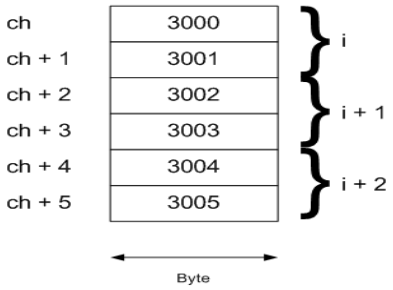
- Exemplo:
  - Supondo p1 um ponteiro para um inteiro com valor atual de 2000.
    p1++; /* p1 = 2002 */
  - A cada incremento de p1, ele apontará para o próximo inteiro. O mesmo vale para os decrementos.
    P1--; /* p1 = 1998 */

- Cada vez que o computador incrementa ou decrementa um ponteiro, ele aponta para a localização de memória do próximo elemento de seu tipo base.

Expressões com Ponteiros
Aritmética (2)

- Pode-se também adicionar ou subtrair de e para ponteiros:
  - p1=p1+9;
  - Nesse caso, p1 apontará para o nono elemento do tipo base de p1, além daquele para o qual estiver apontando no momento;

- Obs: pode-se somente efetuar a soma e subtração com ponteiros.
  - char *ch=3000;
  - Int *i=3000;



Expressões com Ponteiros
Comparações de Ponteiros

- É possível comparar dois ponteiros em uma expressão de relação.
  - Exemplo: (Sendo p e q dois ponteiros)

```
if (p<q) printf ("p aponta para posicao inferior de memória do que q\n");
```

- Em geral utiliza-se comparações de ponteiros quando dois ou mais ponteiros estão apontando para um objeto comum.

Ponteiros e Matrizes
Parte 1 - Geral

- Existe um relacionamento muito próximo entre ponteiros e as matrizes.
- Considerando o exemplo:

```
char str[80], *p;
p = str;
```

- Ajusta p para o endereço do primeiro elemento da matriz str.

- Obs:
  - No C o nome de uma matriz sem um índice é o endereço do início da matriz;
  - O nome da matriz é um ponteiro para aquela matriz.

Ponteiros e Matrizes
Parte 2 – Métodos de Acesso

- Exemplo: para acessar o nono elemento em str:
  str[8] ou *(p1+8)
- Obs: Lembrando que no C, todas as matrizes usam zero como índice do primeiro elemento.
- A linguagem C, permite, essencialmente, dois métodos de acesso aos elementos de uma matriz:
  - a indexação da matriz e
  - a aritmética de ponteiros
- Isto é importante porque a aritmética dos ponteiros pode ser mais rápida que a indexação de matrizes.

Ponteiros e Matrizes
Parte 3 – Exemplo com Matrizes

```
main ()                 /* Versao com Matrizes */
{
   char str[80];
   int i;
   printf ("Digite uma string em letras maiusculas: ");
   gets(str);
   printf ("Eis aqui a string em letras minusculas: ");
   for (i=0; str[i]; i++) printf ("%c", tolower (str[i]));
}
```

Ponteiros e Matrizes
Parte 4 – Exemplo com Ponteiros

```
main ()                 /* Versao com Ponteiros*/
{
   char str[80], *p;
   printf ("Digite uma string em letras maiusculas: ");
   gets(str);
   printf ("Eis aqui a string em letras minusculas: ");
   p = str;             /* obtem o endereco de str */
   while (*p) printf ("%c", tolower (*p++));
}
```


Ponteiros e Matrizes
Parte 5 – Observações

- A razão para a versão com matrizes ser mais lenta que a versão com ponteiros é que o "C" leva mais tempo para indexar uma matriz que para usar o operador (*);
- Para acessar uma matriz na ordem ascendente ou descendente, os ponteiros são mais rápidos e fáceis de usar;
- Para acessar de maneira aleatória, a indexação será melhor, porque é geralmente tão rápida quanto a avaliação de uma expressão complexa com ponteiros e porque será mais fácil de programar e entender;
- Na indexação de matrizes, estará sendo deixado o compilador fazer o trabalho por nós.

Indexando um Ponteiro

- Pode-se indexar um ponteiro como se ele fosse uma matriz.

```
main ()                 /* indexando um ponteiro*/
{
   int i[5]={1,2,3,4,5};
   int *p, t;
   p=i;
   for (t=0;t<5;t++) printf ("%d", p[t]);
}
```

- O comando p[t] é idêntico a *(p+t).

Matrizes sem Índice

Parte 1 - Exemplo

```
strcompara (char *s1, char *s2)          /* compara duas strings */
{
   while (*s1)                  /* sera verdadeiro ate que chegue ao fim da string */
   if (*s1 - *s2)
            return (*s1-*s2);     /* se não igual entao return a diferenca */
   else {
            s1++;
            s2++;
   }
   return ('\0');                /* se for igual */
}
```

O nome de uma matriz sem índice é um ponteiro ao primeiro elemento daquela matriz;

Quando usamos funções de strings, o computador apenas passará para as funções um ponteiro para a strings e não o valor real da string.

Matrizes sem Índice
Parte 2 - argumento

- O mesmo vale se passarmos uma constante de string como argumento.

```
If (!strcompara (“Alo”, str)) printf(“A str contem Alo”);
```

- De forma análoga, quando usamos uma constante de string o computador passará apenas um ponteiro para aquela constante.

Matrizes sem Índice
Parte 3 - Generalizando

```
main()
{
   char *s;
   s=“funcionando !!!”;
   printf (s);
}
```

Quando usamos constante de string em qualquer tipo de expressão, o computador trata a constante como se ela fosse um ponteiro para o primeiro caracter da string.

Matrizes de Ponteiros
Parte 1 -

- Podemos fazer matrizes de ponteiros da mesma forma como faz-se matrizes de qualquer outro tipo de dado;
- A declaração de uma matriz de ponteiros inteiros de tamanho 10 é:

```
int *x[10];
```

- Atribuindo o endereço de uma variável inteira chamada var ao terceiro elemento da matriz de ponteiros:

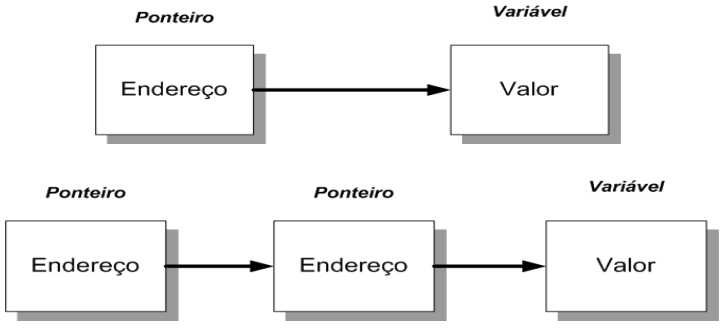
```
x[2]=&var;
```

- Para obtermos o valor de var:

```
valor=*x[2];
```


Ponteiros para Ponteiros
Parte 1 – Visão geral

- Uma matriz de ponteiros é a mesma coisa que ponteiros para ponteiros;
- Um ponteiro para um ponteiro é uma forma de indireção múltipla, ou uma cadeia de ponteiros;



Ponteiros para Ponteiros
Parte 2 - Exemplo

- Declaramos uma variável que é um ponteiro para ponteiro, bastando colocar um (*) adicional na frente do nome da variável:

  float **teste;

- Obs: Teste não é um ponteiro para um número de ponto flutuante (float), mas sim um ponteiro para um ponteiro float.

```
main()
{
   int x, *p, **q;
   x=10;
   p=&x;          /* endereco x */
   q=&p;          /* endereco p */
   printf("%d", **q); /* Imprime x */
}
```


Inicialização de Ponteiros

- Depois de declararmos um ponteiro, ele conterá um valor desconhecido;
- Se tentarmos usar um ponteiro antes de atribuir-lhe um valor, provavelmente danificaremos não apenas o programa, mas também o sistema operacional;
- Uma forma de inicializarmos é:

  char *p="Alo mundo";

Problemas com Ponteiros

- Um erro em ponteiro é difícil de ser encontrado porque o ponteiro em si não é o problema; o problema é que cada vez que executamos uma operação que usa o ponteiro, estaremos lendo ou gravando para um pedaço desconhecido da memória;
- Problemas:
  - Obter lixo;
  - Gravar sobre outras partes de seu código ou dados;
- Cuidados:
  - Saiba sempre para onde o ponteiro está apontando;
  - Nunca use um ponteiro que não foi inicializado.

Alocação Dinâmica

- Depende dos ponteiros para sua operação;
- Dois métodos por meio dos quais um programa pode armazenar informações na memória principal:
  - Variáveis Locais e Globais;
  - Utilizando-se das funções de Alocação Dinâmica {malloc() e free()}
- Neste caso o programa aloca armazenamento para informações da área de memória livre chamada HEAP (ou área de alocação dinâmica).

Alocação Dinâmica
Funções malloc() e free()

- Trabalham juntas usando a região de memória livre para estabelecer e manter uma lista da memória disponível;
- malloc () → solicitar memória;
- free () → liberar memória;

Alocação Dinâmica
Exemplo

```
#include "alloc.h"
#include "stdio.h"
main ()
{
   int *p, t;
   p=(int *) malloc (40*sizeof(int));
   if (!p) printf (“Memoria Insuficiente \n”);
   else {
            for (t=0;t<40;t++) *(p+t)=t;
            for (t=0;t<40;t++) printf (“%d”, *(p+t));
            free (p);
   }
}
```

Contato

eferlin@live.com

(BLOG) professorferlin.blogspot.com

(SITE) professorferlin.webnode.com.br

(YOUTUBE) ProfEdsonPedroFerlin