Linguagem C

Ponteiros

Prof. Daniel Ferreira, MSc
(Adaptado de Prof. Ajalmar Rocha, Dr.)

Instituto Federal do Ceará
Campus Maracanaú

2014.1

Ponteiros I

Razão de Uso

Permitem mudar os argumentos das funções;
Manipular as rotinas de alocação dinâmica;
Aumentar a eficiência do programa.

Definição

Ponteiro é uma variável que contém um endereço de memória.

O conteúdo dessa variável é a posição de outra variável na memória.

Assim, um ponteiro aponta para outra variável quando contém o endereço desta.

Declaração I

Declarar:

```
<tipo> *nome_identificador;
```

Inicializar:

```
<nome_identificador> = NULL;
```

Operador: &

O & é um operador que devolve o endereço da memória do seu operando.

Exemplo:

```
m = &count;
```

Esse endereço é a posição interna da variável na memória do computador.

Não tem relação nenhuma com o valor de count.

Declaração II

Operador: *

O * é um operador que devolve o valor da variável localizado no endereço que o segue.

```
q = *m;
```

Operador: *

```
int main()
{
  int numero = 5; /* suponha na posicao 1000*/
  int *p = &numero; /* p aponta pra numero*/
  printf("numero = %d", *p);
}
```

Exemplo: I

Declarar:

```
int main()
{
   int x = 100; /* suponha na posicao de memoria 2000*/
   int *p = &x; /* p aponta pra x*/
   printf("endereco de x na memoria = %p", &x);
   printf("endereco de x na memoria = %p", p);
   printf("valor de x = %d", x);
   printf("valor de x = %d", *p);
}
```

Expressões com Ponteiros: I

Atribuição de Ponteiros

Da mesma forma que outra variável.

```
int main()
{
  int x;
  int *p1, *p2;

  p1 = &x;
  p2 = p1;
  printf("%p",p2); /*escreve o endereço de x.*/
}
```

Expressões com Ponteiros: II

Aritmética de Ponteiros

Existem duas operações aritméticas com ponteiros: adição e subtração.

```
int main()
{
  int x;  /* supor inteiro com 2 bytes*/
  int *p;
  p = &x; /* supor endereço 2000 contido em p*/
  p++;    /* p = p+1 = 2000 + 1*sizeof(int) = 2002*/
}
```

Expressões com Ponteiros: III

Aritmética de Ponteiros

```
int main()
{
  int x;  /* supor inteiro com 2 bytes*/
  int *p;

  p = &x; /* supor p1 contendo o valor 2000*/
  p++;
  printf("%p",p); /*escreve 2002 e não 2001*/
}
```

Expressões com Ponteiros: IV

Comparação de Ponteiros

Em geral o objetivo é ver se os ponteiros apontam para o mesma região de memória.

```
int main()
{
  int x;
  int *p1, *p2;

  p1 = &x;
  p2 = p1;
  printf("%p %p %d",p1, p2, p1==p2);
}
```

Comparação de Ponteiros

Há uma estreita relação entre ponteiros e matrizes. Veja o código abaixo.

```
int main()
{
  char str[80], *p;
  p = str; /* p aponta para o 1o elemento da matriz*/

  printf("%d %d",str[4], *(p+4)); /* imprime o 5o
elemento da matriz */
}
```

Ponteiros e Matrizes II

Acessar Elementos de Matrizes

Há duas formas:
- aritmética de ponteiros e
- indexação de matrizes.

Aritmética de ponteiros é mais rápida.

Indexação de Matriz

```
/*acessa como um matriz*/
void puts(char *p)
{
  int t;
  for(t=0;p[t];t++)
    putchar(p[t]);
}
```

Ponteiros e Matrizes III

Aritmética de Ponteiros

```
/*acessa como um ponteiro*/
void puts(char *p)
{
  int t;
  while(*p)
   putchar(*s++);
}
```

Matrizes de Ponteiros I

Definição

Cada posição da matriz contem um endereço que deve apontar para uma variável.

Declaração

```
<tipo> *<nome>[tamanho];

int *notas[10];
```

Atribuição

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

Obter o conteúdo:

```
printf("%d",*notas[2]);
```

Indireção Múltipla I

Ponteiro para ponteiro

Você pode ter um ponteiro apontado para outro ponteiro que aponta para o valor final.

Essa situação é chamada ponteiro para ponteiro. O mesmo que, indireção múltipla.

Declaração

```
int **valor;
```

Indireção Múltipla II

Ponteiro para ponteiro

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

Alocação Dinâmica I

Definição

É o meio pelo qual se obtém memória em tempo de execução.

Informação Adicional

Variáveis globais têm o seu armazenamento alocado em tempo de compilação.

Variáveis locais usam a pilha.

A memória alocada pelas funções de alocação dinâmica é obtida do heap.

Alocação Dinâmica II

Funções de Alocação Dinâmica

As funções de alocação dinâmica mais comuns são: malloc() e free().

Função: malloc()

```
void *malloc(size_t numero_bytes);
```

Exemplo 1: malloc()

```
char *p;
p = malloc(1000); /*obtém mil bytes*/
```

Exemplo 2: malloc()

```
int *p;
p = malloc(50 * sizeof(int)); /*espaço p/ 50 inteiros*
```

Alocação Dinâmica III

O heap é infinito?

```
int *p;
p = malloc(50 * sizeof(int));
if(!p){
  printf("Sem memória!!!");
  exit(1);
}
```

Função: free()

```
void free(void *p); /*libera memória,
                      oposto de malloc()*/
```

Matriz Dinamicamente Alocada I

Exemplo

```
int main()
{ char *s;
  int t;
  s = malloc(80);
  if(!s){
    printf("Sem memória!");
    exit(1);
  }
  gets(s);
  for(t=strlen(s) - 1; t >= 0; t-- )
    putchar(s[t]);
  free(s);
}
```