Prática Profissional II – Linguagem de Programação Estruturada

Curso: Análise e Desenvolvimento de Sistemas
Modalidade: Presencial
Professor Esp. Wesley Tschiedel
Email: wesley.tschiedel@ucb.br

Unidade 6
Ponteiros.
Exercícios de fixação.

Um ponteiro proporciona um modo de acesso a variáveis sem referenciá-las diretamente.

O mecanismo usado para isto é o endereço da variável.

O endereço age como intermediário entre a variável e o programa que a acessa.

“Ponteiro é uma representação simbólica de um endereço”.

Por que usamos Ponteiros ?

Usamos em situações em que a passagem de valores é difícil ou indesejável.

Razões para uso do ponteiro:

- Fornecem maneiras com as quais as funções podem realmente modificar os argumentos que recebem;

- Para passar matrizes e “strings” mais convenientemente de uma função para outra, isto é, usá-los ao invés de matrizes;

- Para manipular matrizes mais facilmente através da movimentação de ponteiros para elas (ou parte delas), em vez de a própria matriz;

- Para criar estruturas de dados complexas, como listas encadeadas e árvores binárias, onde uma estrutura de dados deve conter referências sobre outra;

- Para comunicar informações sobre a memória;

- Uma outra razão importante para o uso de ponteiros é que notações de ponteiros compilam mais rapidamente tornando o código mais eficiente.

Ponteiros Constantes e Variáveis

Um ponteiro constante é um endereço;

Um ponteiro variável é um lugar para guardar endereços.

Declarando variável Ponteiro

A declaração de ponteiros tem um sentido diferente da de uma variável simples.

```
int *x, *y;
```

***x** e ***y** são do tipo int e **x**, **y** são ponteiros, ou seja, **x** e **y** contém endereços de variáveis do tipo inteiro.

Operador Indireto (*)

O C oferece dois operadores para trabalharem com ponteiros. Um é o **operador de endereço** (**&**) que retorna o endereço de memória da variável operando.

O outro é o operador indireto (*****) que é o complemento de (**&**) e retorna o conteúdo da variável localizada no endereço (**ponteiro**) operando, ou seja, devolve o conteúdo da variável apontada pelo operando.

Atribuindo valores a variáveis ponteiros

Utilizando o operador (*) podemos acessá-las:

```
*x = 3;
*y = 5;
```

Na declaração, o símbolo (*) indica o "tipo apontado", em outras instruções indica "a variável apontada por".

Ponteiros podem ser usados não somente para que a função passe valores para o programa que chama, mas também para passar valores do programa para a função.

EXEMPLO

```
1. int adcon(int *px, int *py);
2. main()
3. {
4.       int x=4, y=7;
5.       adcon(&x, &y);
6.       printf("O primeiro e %d, o segundo e %d. \n", x, y);
7.       system("PAUSE");
8.       }
9. int adcon(int *px, int *py)
10.{
11.    *px = *px + 10;
12.    *py = *py + 10;
13.}
```

Ponteiros sem Funções

O seguinte código executa a mesma tarefa do código anterior, mas em vez de chamar uma função para somar a constante às duas variáveis, executa estas operações ele próprio.

EXEMPLO

```
1. main()
2. {
3.       int x=4, y=7;
4.       int *px, *py;
5.       printf("x = %d, y = %d. \n", x, y);
6.       px = &x;
7.       py = &y;
8.       *px = *px + 10;
9.       *py = *py + 10;
10.      printf("Agora x = %d, y = %d. \n", x, y);
11.      system("PAUSE");
12.      }
```

Atribuição

Um endereço pode ser atribuído a um ponteiro. Normalmente fazemos isso usando o nome de uma matriz ou o operador de endereços (**&**) junto a uma variável simples.

O operador (*****) devolve o valor guardado no endereço apontado.

Endereço do Ponteiro

Como todas as variáveis, os ponteiros variáveis têm um endereço e um valor.

O operador (**&**) retorna a posição de memória onde o ponteiro está localizado.

Em resumo temos:

O nome do ponteiro retorna o endereço para o qual ele aponta.

O operador **&** junto ao nome do ponteiro retorna o endereço do ponteiro.

O operador ***** junto ao nome do ponteiro retorna o conteúdo da variável apontada.

Incrementando o ponteiro

Podemos incrementar um ponteiro através de adição regular ou pelo operador de incremento.

Incrementar um ponteiro acarreta a movimentação do mesmo para o próximo tipo apontado isto é, se **x** é um ponteiro para **int** com valor **3000** depois de executada a instrução: **x++;**

o valor de **x** será **3002** e não **3001**. Cada vez que incrementamos **x** ele apontará para o próximo tipo apontado, ou seja, o próximo inteiro.

O mesmo acontece para decremento.
Se **x** tem valor **3000** depois da instrução:

x--;

ele terá o valor **2998**.

Você pode adicionar ou subtrair de e para ponteiros. A instrução:

y = x +3;

fará com que **y** aponte para o terceiro elemento do tipo apontado após **x**.

Se **x** tem valor **3000**, depois de executada a instrução acima, **y** terá o valor **3006**.

Variáveis ponteiros podem ser testadas quanto à igualdade (**==**) ou desigualdade (**!=**) onde os dois operando sejam ponteiros ou um dos operando **NULL**:

(x != NULL ou x != '\0')

Ponteiros e Matrizes

O compilador transforma matrizes em ponteiros quando compila, pois a arquitetura do microcomputador entende ponteiros e não matrizes.

O nome de uma matriz é um endereço, ou seja, um ponteiro.

Ponteiros e matrizes são idênticos na maneira de acessar a memória.

Na verdade o nome de uma matriz é um ponteiro constante.

Um ponteiro variável é um endereço onde é armazenado um outro endereço.

Exemplo sem ponteiro

```
1. main()
2. {
3.       static int nums[]={92, 81, 70, 69, 58};
4.       int d;
5.       for (d=0; d<5; d++)
6.       printf("%d ", nums[d]);
7.       printf("\n\n");
8.       system("PAUSE");
9.       }
```

Exemplo com ponteiro

```
1. main()
2. {
3.       static int nums[]={92, 81, 70, 69, 58};
4.       int d;
5.       for (d=0; d<5; d++)
6.       printf("%d ", *(nums+d));
7.       printf("\n");
8.       system("PAUSE");
9.       }
```

A expressão **nums+d** é o endereço do elemento de índice da matriz.

Se a matriz tiver o endereço **3000** e **d** for **3**, então **nums+d** terá o endereço **3006**.

Se **d** tem valor **3** então ***(nums+d)** expressa o conteúdo do elemento **3** da matriz **nums[]**, que no nosso exemplo tem o valor **69**.

*(matriz+índice)

É O MESMO QUE

matriz[índice]

Existem duas maneiras de referenciar o endereço de um elemento da matriz: em notação de ponteiro, **nums+d**, ou em notação de matriz, **&nums[d]**.

Se o endereço da matriz é **3000**, então:

&nums[2] == (nums+2) == 3004

nums[2] == *(nums+2) == 70

Exemplo

```
1.  #define LIM 40
2.  main()
3.  {
4.      float notas[LIM], soma=0.0;
5.      int i=0;
6.      do{
7.          printf("Digite a nota do aluno %d: ", i);
8.          scanf("%f", notas+i);
9.          if(*(notas+i)>0)
10.         soma+=*(notas+i);
11.         } while(*(notas+i++)>0);
12.         printf("Media das notas: %.2f \n", soma/(i-1));
13. }
```

Você não pode alterar o valor de um ponteiro constante, somente o de um ponteiro variável.

Exemplo

```
1.  #define LIM 40
2.  main()
3.  {
4.      float notas[LIM], soma=0.0;
5.      int i=0;
6.      float *ptr;
7.      ptr=notas;
8.      do{
9.          printf("Digite a nota do aluno %d: ", i++);
10.         scanf("%f", ptr);
11.         if(*ptr > 0)
12.         soma+=*ptr;
13.         } while(*(ptr++) > 0);
14.         printf("Media das notas: %.2f \n", soma/(i-1));
15. }
```

PONTEIROS E “STRINGS”

Várias funções utilizam ponteiros para manipulação de ***“strings”***.

Exemplo

```
1. main()
2. {
3.     char *lista="1234567890";
4.     printf("O tamanho do string '%s' e %d caracteres.\n\n", lista,
                    tamanho(lista));
5.     printf("Acabou. \n\n");
6.     system("PAUSE");
7. }
8. int tamanho(char *s){
9.     int tam=0;
10.    while(*(s + tam++) != '\0');
11.    return tam-1;
12.}
```

No exemplo anterior aparece uma função que conta o número de caracteres de um string.

Observe o ponteiro para o string constante e na função o ponteiro ***(s+tam++)** apontando caracter por caracter.

Exemplo

```
1.  char copia(char *d, char *o);
2.  main()
3.  {
4.      char destino[20];
5.      char  *origem="string de origem";
6.      copia(destino, origem);  /* copia o string origem para o destino */
7.      printf("%s\n", origem);
8.      printf("%s\n", destino);
9.      printf("Acabou.\n");
10.     system("PAUSE");
11. }
12. char copia(char *d, char *o){
13.     while ((*d++ = *o++) != '\0');
14.     return 1;
15. }
```

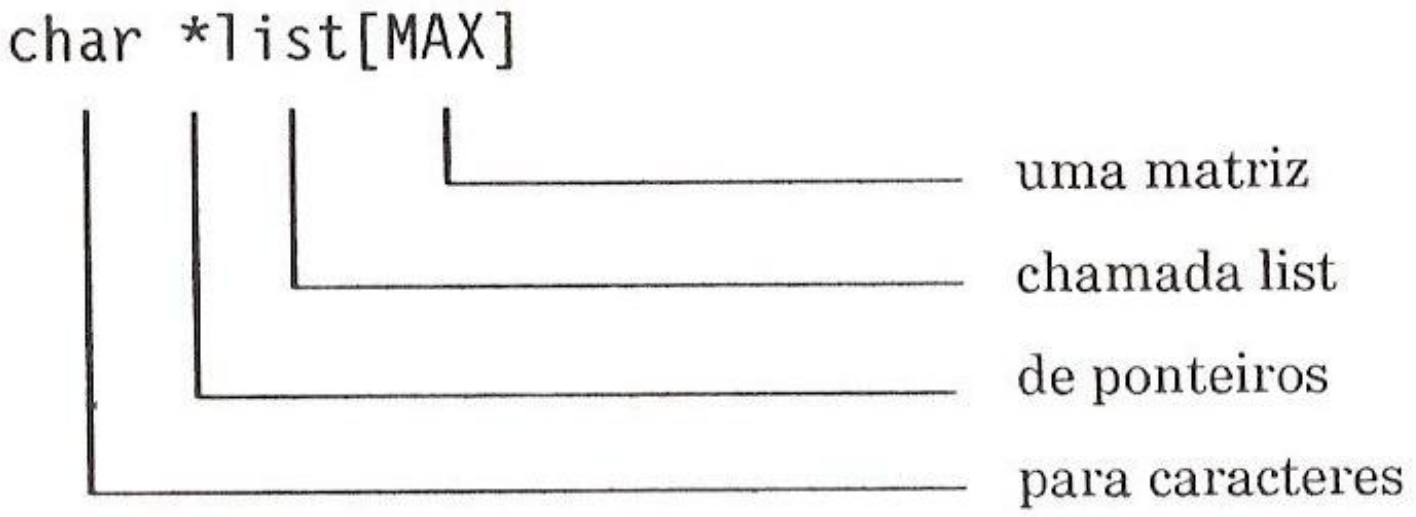
Inicialização de “string” através de ponteiros

```
1.  main()
2.  {
3.      char *salute="saudacoes, ";
4.      char nome[81];
5.      puts("Digite seu nome: ");
6.      gets(nome);
7.      puts(salute);
8.      puts(nome);
9.      system("PAUSE");
10.     }
```

Inicializando uma matriz de ponteiros para “strings”

```
1.  #define MAX 5
2.  main()
3.  {
4.      int d;
5.      int entra=0;
6.      char nome[40];
7.      static char *list[MAX] =
8.          { "Katarina",
9.            "Nigel",
10.           "Gustavo",
11.           "Francisco",
12.           "Airton"    };
```

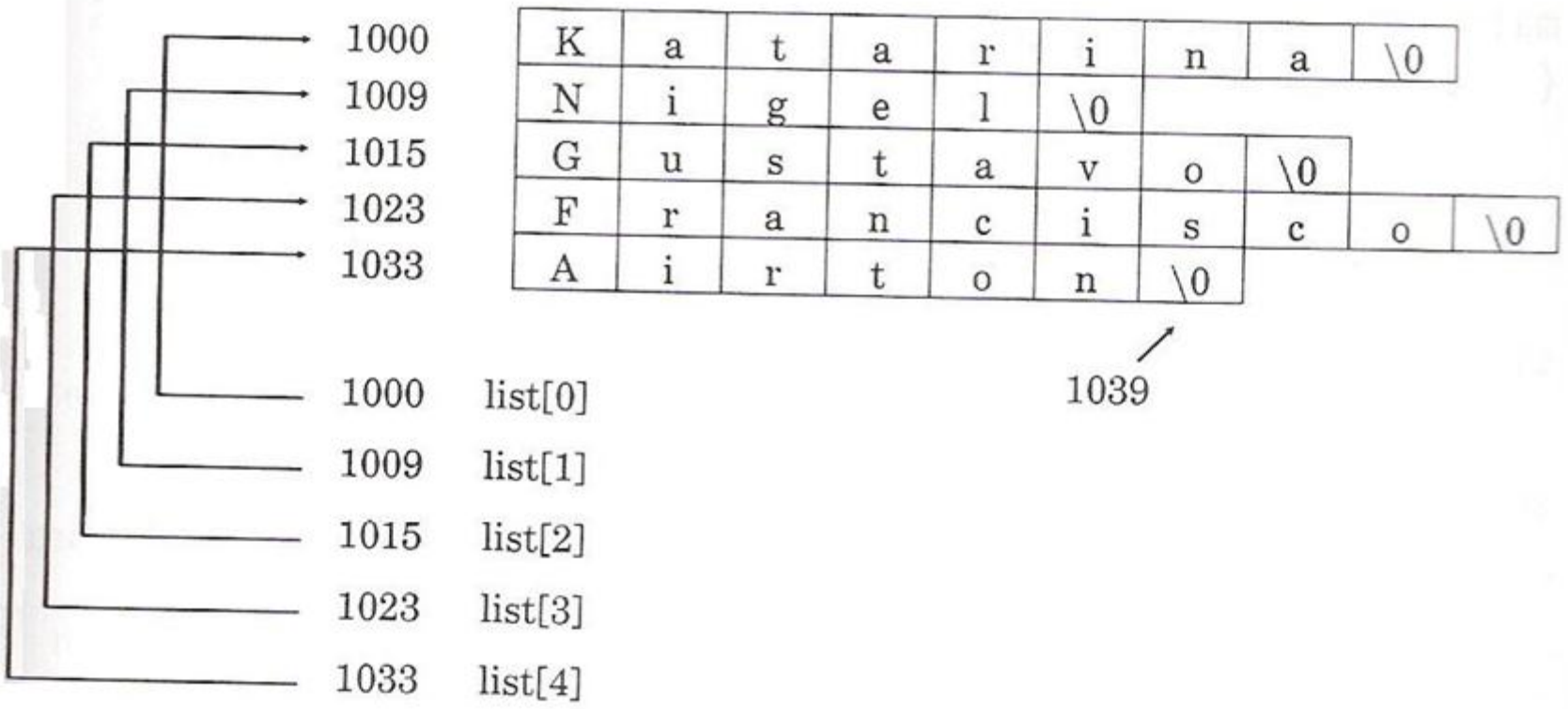
```
13.     printf("Digite seu nome: ");
14.     gets(nome);
15.     for(d=0;d<MAX;d++)
16.     if(strcmp(list[d], nome)==0)
17.     entra=1;
18.     if(entra==1)
19.     printf("Voce pode entrar, o honrado Sr.! \n\n");
20.     else
21.     printf("Guardas! removam este sujeito! \n\n");
22. }
```


char *list[MAX]
uma matriz
chamada list
de ponteiros
para caracteres

list[0] → 1000
list[1] → 1010
list[2] → 1020
list[3] → 1030
list[4] → 1040

0	1	2	3	4	5	6	7	8	9
K	a	t	a	r	i	n	a	\0	
N	i	g	e	l	\0				
G	u	s	t	a	v	o	\0		
F	r	a	n	c	i	s	c	o	\0
A	i	r	t	o	n	\0			

1049


1000
1009
1015
1023
1033
K a t a r i n a \0
N i g e l \0
G u s t a v o \0
F r a n c i s c o \0
A i r t o n \0
1039
1000 list[0]
1009 list[1]
1015 list[2]
1023 list[3]
1033 list[4]

INICIALIZAR UMA MATRIZ DE "*STRINGS*" USANDO PONTEIROS ALOCA MENOS MEMÓRIA QUE A INICIALIZAÇÃO ATRAVÉS DE MATRIZ.

Duplamente Indireto: Ponteiros para Ponteiros

Um elemento de uma matriz de duas dimensões pode ser referenciado através de um ponteiro para ponteiro.

Exemplo

```
1.  #define LIN 4
2.  #define COL 5
3.  main()
4.  {
5.      static int tabela[LIN][COL] =
6.          { {13,15,17,19,21},
7.            {20,22,24,26,28},
8.            {31,33,35,37,39},
9.            {40,42,44,46,48} };
10.         int c=10;
11.         int j, k;
12.     for(j=0;j<LIN;j++)
13.     for(k=0;k<COL;k++)
14.     *(*(tabela+j)+k) +=c;
15.     for(j=0;j<LIN;j++){
16.     for(k=0;k<COL;k++)
17.     printf("%d ", *(*(tabela+j)+k));
18.     printf("\n");
19.     }
20. }
```

Ponteiros para Funções

Um ponteiro para uma função é um caso especial de tipo apontado.

Se você definiu um ponteiro para função e inicializou-o para apontar para uma função particular ele terá o valor do endereço onde a função está localizada na memória.

Exemplo

```
1. void func1()
2.       {
3.            printf("\n Estou na funcao 1\n");
4. }
5. main()
6.      {
7.       void func1();
8.       void (*ptrf)();
9.       ptrf=func1;
10.      (*ptrf)();
11.      system("PAUSE");
12.      }
```

A declaração de **func1( )** como do tipo **void** em **main( )** e na sua definição é obrigatória para que **main( )** conheça seu endereço, caso contrário o compilador apresentará um erro.

A instrução: **void (*ptrf)( );**

declara um ponteiro para uma função do tipo **void**.

É claro que o tipo **void** é uma das possibilidades.

Se a função a ser apontada retornar um **float**, por exemplo, o ponteiro para ela deve ser declarado como tal.

Os primeiros parênteses são necessários, pois se omitidos como em **void *ptrf( )** você estará declarando que **ptrf( )** é uma função do tipo ponteiro para **void** e não que **ptrf** é um ponteiro para uma função do tipo **void**.

O código: **ptrf = func1;**

atribui o endereço de **func1( )** a **ptrf**.

Observe que não colocamos parênteses juntos ao nome da função.

Se eles estiverem presentes como em **ptrf = func1( );** você estará atribuindo a **ptrf** o valor retornado pela função e não o endereço dela.

O nome de uma função desacompanhado de parênteses é o endereço dela.

O código: **(*ptrf)( );**

é equivalente a **func1( );** e indica uma chamada à função **func1( )**.

Podemos:

- Declarar um ponteiro para função.
- Atribuir o endereço de uma função a um ponteiro.
- Chamar a função através do ponteiro para ela.

Atividade Prática

Referências Bibliográficas

Básica:

- EVARISTO, J.,**Aprendendo a programar programando em C**, Book Express, 2001, 205p.
- MIZRAHI, V. V., **Treinamento em Linguagem C**, Módulo 1 e 2, Makron Books do Brasil Editora Ltda, 1990, 273 p.
- SCHILDT, H., **C Completo e Total**, Editora Makron Books doBrasil Editora Ltda, 1997, 827p.

Referências Bibliográficas

Complementar:

- DEITEL, H. M. e Deitel, P. J.,**C++ Como Programar**, 3. ed. Porto Alegre: Artmed Editora S.A, 2001. 1098 p.

- MANZANO, J. A. N. G. **Estudo Dirigido: Linguagem C.** 6. ed. São Paulo: Érica, 2002.

- SOFFNER, Renato. **Algoritmos e programação em linguagem C**. São Paulo Saraiva 2013.

- TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. **Estruturas de Dados Usando C.** São Paulo: Makron Books, 1995.

- ZIVIANI, Nivio. **Projeto de algoritmos: com implementações em Pascal e C.** 3. ed., rev. e ampl. São Paulo, SP: Cencage Learning, 2015. xx, 639 p.