SSC0101 - ICC1 – Teórica

Introdução à Ciência da Computação I

Estruturas Dinâmicas - Ponteiros
Parte III

Prof. Vanderlei Bonato: vbonato@icmc.usp.br
Prof. Claudio Fabiano Motta Toledo: claudio@icmc.usp.br

Sumário

- Matrizes e vetores de ponteiros
- Funções como argumentos

Matrizes e vetores de ponteiros

- Ponteiros podem ser dispostos em vetores ou matrizes da mesma forma que outros tipos de dados.
- int *x[10]
  - Cria um vetor de tamanho 10 formado por ponteiros do tipo int
- x[2] = &var
  - Atribui o endereço de uma variável ao terceiro elemento do vetor de ponteiros do tipo int.
- *x[2]
  - Exibe o conteúdo armazenado em var que é apontado por x[2].

Matriz e vetores de ponteiros

- Considere a função abaixo:

```
void display_array(int *q[])
{
  int t;
  for (t=0; t<10; t++)
      printf("%d", *q[ t ]);
}
```

- O ponteiro q é um ponteiro para um vetor de ponteiros do tipo int.

- Por isso, deve ser declarado como um vetor de ponteiros.

- Se int *q fosse utilizado, teríamos q apenas como um ponteiro do tipo int.

Matrizes e vetores de ponteiros

- Há semelhanças e diferenças no uso de uma matriz bidimensional e um vetor de ponteiros do tipo char.

```
#include <stdio.h>
int main (void){
 char a[2][15] = {“abc:”, “a is for apple”};
 char  *p[2] = {“abc:”, “a is for apple”};
printf(“%c%c%c  %s  %s\n”, a[0][0], a[0][1], a[0][2], a[0],
   a[1]);
printf(“%c%c%c  %s  %s\n”, p[0][0], p[0][1], p[0][2], p[0],
   p[1]);
}
```

```
abc   abc :  a is apple
 abc  abc :  a is apple
```

Matrizes e vetores de ponteiros

- O identificador **a** é um vetor bidimensional com espaço alocado para 30 caracteres inicializado da seguinte forma:

{ {‘a’, ‘b’, ‘c’,’:’,’\0’} , {‘a’, ‘ ‘, ’i’,’s’,’ ‘, ‘f’, ‘o’, ‘r’, ...} }

a[0] = {‘a’, ‘b’, ‘c’,’:’,’\0’}

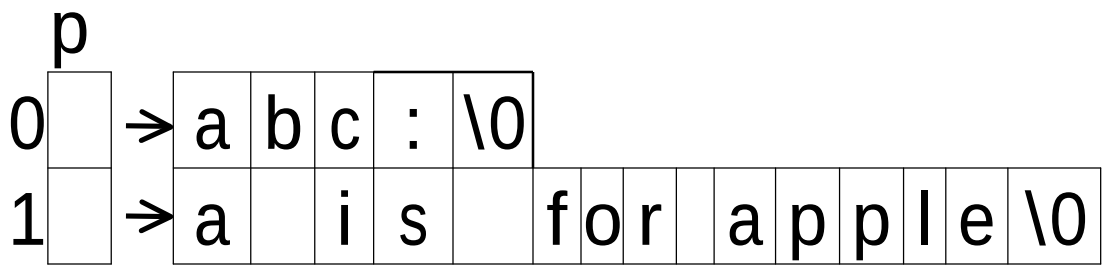
- a[0] e a[1] são cadeias de caracteres (*strings*)

- Apenas 5 elementos estão especificados em a[0], mas há espaço alocado para 15 caracteres. Os demais elementos são iniciados com valor zero (caractere do tipo null).

- Devido à alocação de espaço, qualquer das 30 posições pode ser acessada fazendo a[i][j].

Matrizes e vetores de ponteiros

- O identificador **p** representa um vetor de ponteiros para char.
- A declaração de **p** aloca espaço para dois ponteiros.
- O ponteiro p[0] é iniciado apontando para a cadeia de caracteres “abc:” que requer 5 espaços do tipo char.

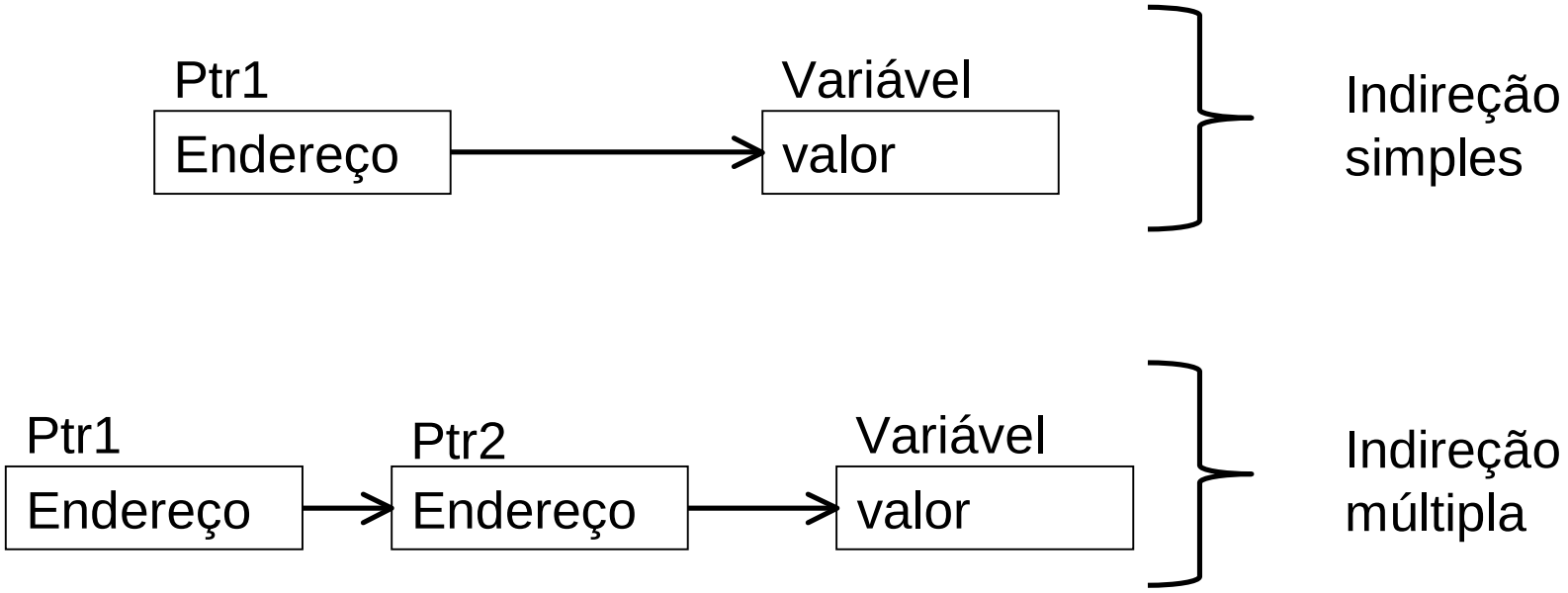


Matrizes e vetores de ponteiros

- **p** trabalha com menos espaço do que **a**.
- Observe que a[1][14] é uma posição válida, mas p[0][14] não.
- As cadeias de caracteres apontadas por p[0] e p[1] devem ser modificadas com cuidado respeitando a área alocada na declaração.

Matrizes e vetores de ponteiros

- **Indireção múltipla** ou **ponteiros para ponteiros** ocorre quando temos ponteiro apontando para outro ponteiro que aponta para determinado valor.



Matrizes e vetores de ponteiros

```
#include <stdio.h>

int main(void){
 int x, *p, **q;
 x=10;
 p=&x;
 q=&p;
 printf("%d",**q);
return 0;
}
```

- Um ponteiro para um ponteiro deve ser declarado com a adição de mais um *.
- int **q;
  - Indica que q é um ponteiro para um ponteiro do tipo int.
- **q
  - acessa o valor apontado pelos ponteiros.

Matrizes e vetores de ponteiros

```
#include <stdio.h>
#include <stdlib.h>
double *alocandoVetor(int *);
double *liberandoVetor(int , float *);

void main (void)
{
  double *vetor;
  int tam;
  vetor = alocandoVetor (&tam);
  printf("tam=%d",tam);
  vetor = liberandoVetor(tam, vetor);
}
```

```
double *alocandoVetor(int *tam)
{
  double *vet;
  do{
     printf ("\nTamanho do vetor:");
     scanf("%d",tam);
     }while(tam<1);
  vet = (double *) calloc (*tam+1,
     sizeof(double));
  if (!vet) {
     printf ("\nEspaço em Memoria
     Insuficiente\n");
     return (NULL);
     }
  return (vet);
}
```

```
double *liberandoVetor(int tam,
     float *vet)
{
  if (!vet) return (NULL);
  free(vet);
  return (NULL);
}
```

Matrizes e vetores de ponteiros

```
#include <stdio.h>
#include <stdlib.h>

double **alocandoMatriz (int *, int *);
float **liberandoMatriz(int, int, float **);

void main (void)
{
  float **matriz;  /* matriz a ser alocada */
  int   ln, cl;   /* numero de linhas e colunas da matriz */
  matriz = alocandoMatriz(&ln, &cl);
  matriz = liberandoMatriz(ln, cl, matriz);
  }
```

```
double **alocandoMatriz (int* linhas, int* colunas)
{
  int   i;
  double **mat;
 //Recebendo num. de linhas e colunas
  do{
      printf("Num. linhas=");
      scanf("%d",linhas);
  }while(linhas<1);

  do{
      printf("Num. colunas=");
      scanf("%d",colunas);
  }while(colunas<1);
```

```
  //Alocação das linhas
   mat = (double **) calloc (*linhas, sizeof(double *));
   if (!mat) {
      printf ("\nEspaço em Memoria Insuficiente\n");
      return (NULL);
   }

//Alocação das colunas
   for ( i = 0; i < *linhas; i++ ) {
       mat[i] = (double*) calloc (*colunas, sizeof(double));
       if (!mat[i]) {
          printf ("Espaço em Memoria Insuficiente");
          return (NULL);
        }
   }
   return (mat);
}
```

```
float **liberandoMatriz(int linhas, int colunas, float **mat)
{
  int  i;

  if (!mat)
     return (NULL);

  //Liberando as linhas da matriz
  for (i=0; i<linhas; i++) free (mat[i]);

  //Liberando a matriz
  free (mat);

  return (NULL);
}
```

Funções como argumentos

- Uma função não é uma variável em linguagem C.
- Porém, um apontador para uma função pode ser definido.
- Assim, esse apontador pode ser atribuído, colocado em vetores, passado para funções e retornado de funções, etc.

Funções como argumentos

```
include "sum_sqr.h"

int main(void)
{

  printf("%s%.7f\n%s%.7f\n",
        "Primeiro cálculo: ", sum_square(f,1,1000),
         "Segundo cálculo: ", sum_square(sin,2,13));

    return 0;
}
```

sum_sqr.h

```
#include <math.h>
#include <stdio.h>
double f(double x);
double sum_square(double f(double x), int m, int n);
```

sum_sqr.c

```
#include "sum_sqr.h“
double sum_square(double f(double x), int m, int
    n)
{
   int k;
   double sum = 0.0;
   for(k=m; k<=n; ++k)
     sum += f(k)*f(k);
   return sum;
}
```

fct.c

```
#include "sum_sqr.h“
double f(double x)
{
   return 1.0/x;
}
```

Funções como argumentos

- Na definição da função sum_square() em sum_sqr.c, o identificador x não é necessário.

```
double sum_square(double f(double), int m, int n)
                                        {...}
double sum_square (double (*f)(double), int m, int n)
                                        {...}
```

- Uma função na lista de parâmetros de outra função é interpretada como um ponteiro pelo compilador.

- No exemplo, leia-se “f é um ponteiro para uma função que recebe um único argumento double e retorna um double.”

Funções como argumentos

- As seguintes instruções são equivalentes:

sum += (*f)(k)*(*f)(k) $\Leftrightarrow$ sum += f(k)*f(k);

- Podemos interpretar da seguinte forma (*f)(k):

f: ponteiro para uma função.
*f: a função.
(*f)(k) chamada para a função.

Funções como argumentos

- Os seguintes protótipos são equivalentes:

```
double  sum_square(double f(double x), int m, int n);

double  sum_square(double f(double ), int m, int n);

double  sum_square(double f(double ), int , int );

double  sum_square(double (* f)(double ), int , int);

double  sum_square(double  (*)(double ), int , int);

double  sum_square(double  g(double y), int a, int b);
```

Referências

Ascencio AFG, Campos EAV. Fundamentos de programação de computadores. São Paulo : Pearson Prentice Hall, 2006. 385 p.

Kelley, A.; Pohl, I., *A Book on C*: programming in C. 4ª Edição. Massachusetts: Pearson, 2010, 726p.

Kernighan, B.W.; Ritchie, D.M. C, *A Linguagem de Programação*: padrão ANSI. 2ª Edição. Rio de Janeiro: Campus, 1989, 290p.

Schildt, Herbet, *C Completo e Total*, Pearson, 2006,

FIM Aula 16