Disciplina:
Programação Computacional

Prof. Fernando Rodrigues
e-mail: fernandorodrigues@sobral.ufc.br

Aula 10: Programação em C

- Estruturas de dados homogêneas;
- Conceito de estruturas de dados estáticas;
- Vetores e matrizes;
- Manipulação de cadeias de caracteres (strings).

Vetores unidimensionais (Arrays)

- São estruturas de dados homogêneas, ou seja, formam um conjunto ordenado de dados onde todos os elementos são do mesmo tipo.
- São referenciadas por um nome comum, e acessadas através de um índice.
- A estrutura de dados é dita estática quando tem um tamanho definido, onde os dados ficam em posições contíguas na memória . Ex: int vet[3];

Vetores unidimensionais (Arrays)

- Sintaxe:
  - tipo nome_var[tamanho];

```
9          int vet[3];
```

- Onde tamanho precisa ser um valor inteiro ou uma expressão avaliada como um inteiro, podendo conter variáveis e literais.

Ex: #define N 20

int vet[N+5];

- Precisa ser declarado e ter seu tamanho definido, para o compilador alocar espaço para cada célula do vetor na memória.
- Matrizes e ponteiros estão intimamente ligados em C.
- Vet é um ponteiro para o elemento vet[0]:
  - Por conta disso vet e &vet[0] são a mesma coisa, a referência para o endereço de memória do primeiro elemento

Vetores unidimensionais (Arrays)

- Percorrendo e inicializando um vetor

```
9       int vet[3];
10
11      for(int i = 0; i < 3 ; i++)
12              vet[i] = i+1;
```

- Lendo valores em um vetor

```
9       int vet[3];
10
11      for(int i = 0; i < 3 ; i++){
12              printf("Digite o valor: ");
13              scanf("%d",&vet[i]);
14      }
```

- Índice é diferente de posição
  - 0 1 2 3 4 - índices
  - 5 10 12 8 9 - valores
  - 1 2 3 4 5 - posição

Matrizes bidimensionais

- C suporta matrizes multidimensionais.
- A forma mais simples de matriz multidimensional é a bidimensional.
- É uma matriz de matrizes unidimensionais, ou seja, é um vetor de vetores.

Matrizes bidimensionais (Matrizes)

- Sintaxe

<tipo_de_dado> <nome_matriz>[tamanho1] [tamanho2];

```
9       int mat[3][3];
```

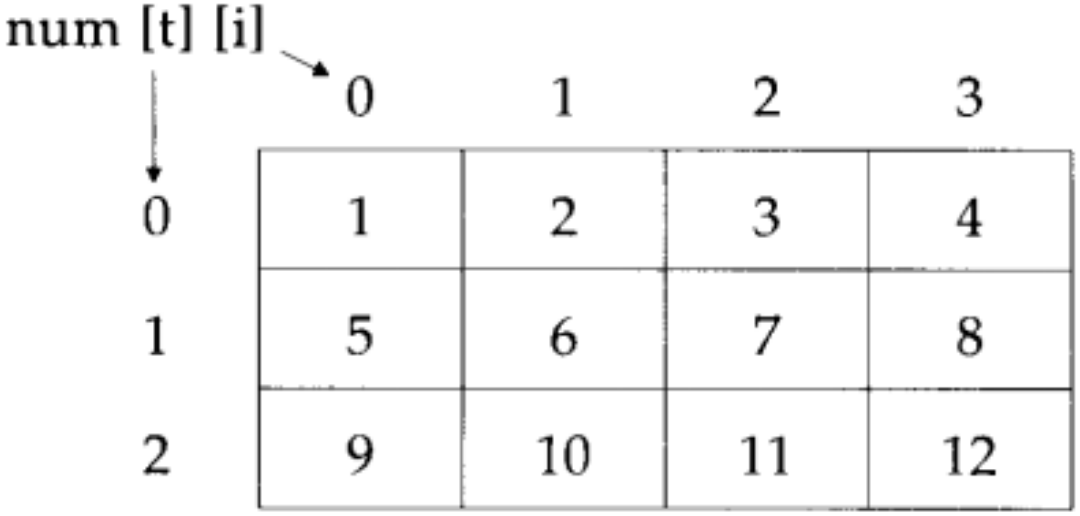
```
#include <stdio.h>

void main(void)
{
  int t, i, num[3][4];

  for(t=0; t<3; ++t)
    for(i=0; i<4; ++i)
      num[t][i] = (t*4)+i+1;

/* agora escreva-os */
  for(t=0; t<3; ++t) {
    for(i=0; i<4; ++i)
      printf("%3d ", num[t][i]);
    printf("\n");
  }
}
```

Matrizes bidimensionais (Matrizes)



Matrizes multidimensionais

- Sintaxe
  - <tipo> <nome>[tamanho1][tamanho2]...[tamanhoN];
  - Ex:
    - int matNum[3][5][2];
    - float matPesos[5][2][4][5];
    - double valores[10][20][30];

Matrizes

- Podemos aplicar o conceito de matrizes a todos os tipos de dados, sendo ele um tipo básico, uma estrutura homogênea ou heterogênea(composta).

- Ex: char vet[] = “teste”;

Inicialização de vetores e matrizes

- int vet[ ] = {1,2,3,4,5,6};
  - vet[6]
- int mat[ ][3] = {1,2,3,4,5,6};
  - Aqui deve ser informado pelo menos o número de colunas
  - mat[2][3]
- char string[ ] = “Mensagem”;
  - string[9];

O tipo **char**

Variáveis ou constantes do tipo **char** são usadas para amazenar caracteres.

Na atribuição de valores a variáveis do tipo **char** os símbolos devem ser escritos entre aspas simples ' '.

Usa-se o código de formatação %c para ler ou exibir valores do tipo **char**.

```
int main()
{
  char c1, c2, c3;

  c1 = 'O';
  c2 = 'b';
  c3 = 'A';
  printf("%c %c %c\n", c1, c2, c3);   // exibe O b A
  printf("%c%c%c\n", c2, c1, c3);     //exibe bOA

  system("PAUSE");
  return(0);
}
```

Funções nativas de entrada e saída

- `getchar()`: lê um caractere até que a tecla <ENTER> seja pressionada. Se mais de um caractere for digitado, apenas o primeiro caractere será considerado e o restante será descartado.

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

int main()
{
  char a;

  printf("Digite um caractere: ");
  a = getchar();          // armazena a entrada até pressionar <ENTER>
  printf("Caractere digitado: %c\n", a);
  system("PAUSE");
  return(0);
}
```

Funções nativas de entrada e saída

- `putchar( )`: Exibe na tela o caractere passado como argumento, a partir da posição atual do cursor.

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

int main()
{
  char a;

  printf("Digite uma letra minuscula: ");
  a = getchar();
  putchar( toupper(a) );
  putchar('\n')
  system("PAUSE");
  return(0);
}
```

A função puts ( )

- É utilizada apenas para exibir mensagens na tela.
- A mensagem a ser exibida deverá ser escrita entre aspas.
- Após a exibição da mensagem, a função puts ( ) muda de linha automaticamente.

```
int main()
{
  puts("Digite sua opcao:");
  puts("[1] Consultar");
  puts("[2] Incluir");
  puts("[3] Atualizar");
  puts("[4] Excluir");
  puts("[5] Encerrar");
  printf("-> ");
  ...
}
```

Trabalhando com vetores (atribuição)

- Temos que tomar cuidado ao trabalhar com vetores na operação de atribuição, pois não pode ser feita atribuição direta, como ilustrado abaixo:

Não se pode fazer atribuição de arrays inteiros, apenas de suas posições individualmente.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    int v[5] = {1,2,3,4,5};
05    int v1[5];
06    v1 = v; //ERRADO!
07    int i;
08    for(i=0; i<5; i++)
09       v1[i] = v[i]; //CORRETO
10    system("pause");
11    return 0;
12  }
```

Trabalhando com strings (atribuição)

- Como strings são vetores de caracteres em C, então acontece a mesma coisa em relação a operação de atribuição:

Strings são arrays. Portanto, não se pode fazer atribuição de strings.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    char str1[20] = "Hello World";
05    char str2[20];
06
07    str1 = str2; //ERRADO!
08
09    system("pause");
10    return 0;
11  }
```

Atribuição de strings (correto)

- O correto seria fazer como a seguir:

Exemplo: copiando uma string

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    int i;
05    char str1[20] = "Hello World";
06    char str2[20];
07    for (i = 0; str1[i]!='\0'; i++)
08       str2[i] = str1[i];
09    str2[i] = '\0';
10    system("pause");
11    return 0;
12  }
```