Linguagens de Programação I

Tema # 7

Vetores ou Matrizes Unidimensionais

Susana M Iglesias

MOTIVAÇÃO

- Escreva um programa que processe as notas de uma turma de 3 alunos. Seu programa deve calcular a média e informar se cada aluno esta abaixo ou acima da média.

Entrada: `3 notas dos alunos (n1, n2, n3)`

Saída: `media = (n1 + n2 + n3) /3`

`Mensagem (n1>=media) ou (n1<media)`

- As notas devem ser armazenadas.

```c
int main(){
  int n1, n2, n3;
  float media;

  printf("Digite as 3 notas: ");
  scanf("%d %d %d", &n1, &n2, &n3);

  media = (n1 + n2 + n3)/3.0;
  if (n1>=media)
    printf("Aluno 1 acima da media.\n");
  else
    printf("Aluno 1 abaixo da media.\n");
  if (n2>=media)
    printf("Aluno 2 acima da media.\n");
  else
    printf("Aluno 2 abaixo da media.\n");
  if (n3>=media)
    printf("Aluno 3 acima da media.\n");
  else
    printf("Aluno 3 abaixo da media.\n");

  return 0;
}
```

Como melhorar o código?

```c
#define N 3

int main(){
  int n[N], i;
  float media = 0;

  for(i=0;i<N;i++){
    printf("Digite a nota[%d]: ", i+1);
    scanf("%d", &n[i]);
    media += n[i];
  }
  media /= N;
  printf("\nMedia: %.2f\n", media);
  for(i=0;i<N;i++)
    if (n[i]>=media)
      printf("Aluno %d acima da media.\n", i+1);
    else
      printf("Aluno %d abaixo da media.\n", i+1);

  return 0;
}
```

INTRODUÇÃO

- Um vetor geralmente é associado a uma lista ou conjunto de elementos similares,
- Exemplos:
  - Os números inteiros de 1..10
  - As notas de uma turma em uma disciplina
  - Os resultados de uma enquête sobre a cantina da UESC
  - Preços de venda dos produtos de uma loja

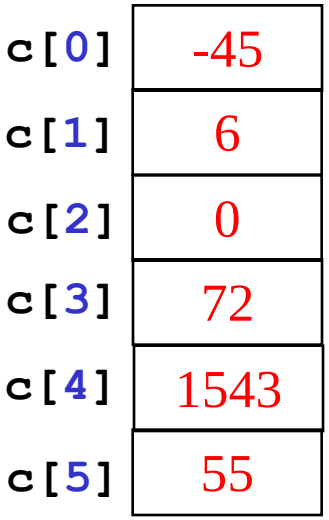
INTRODUÇÃO

- Vetores definição:
  1. itens de dados **do mesmo tipo**, relacionados entre si.
  2. coleção de variáveis **do mesmo tipo** que é referenciado por um **nome comum**.
- Os vetores são entidades estáticas, i.e. permanecem do mesmo tamanho ao longo da execução do programa,
- Um vetor é um grupo de locais de memória relacionados pelo fato que tem o mesmo nome e o mesmo tipo,

INTRODUÇÃO

- os elementos do vetor ocupam posições contínuas na memória,
- o endereço mais baixo corresponde ao primeiro elemento e o mais alto ao último,

Representação de um vetor



c – nome do array
- valores do array
- subscritos do array

- Variáveis independentes são alocadas na memória em forma contínua?

SUBSCRITOS

- para fazer referencia a um elemento no vetor precisamos o nome do vetor e a posição de aquele elemento no vetor,

```
c[0], a[21]
```

- a posição de um elemento no vetor é formalmente conhecida como subscrito,
- um subscrito deve ser um inteiro ou uma expressão inteira,
- para um vetor de *N* elementos os subscritos variam de 0 a *N-1*,

DECLARANDO VETORES

- Ao declarar um vetor reservamos espaço na memória para ele,
- portanto, devemos especificar o tipo de cada elemento e a quantidade apropriada de elementos,

```
tipo nome_do_array[tamanho]
```

DECLARANDO VETORES

```
tipo nome_do_array[tamanho]
```

- O tamanho do vetor deve ser uma constante inteira, variáveis não podem ser utilizadas.
- Por que?
- Exemplos:
  - `int c[6];`
  - `float b[100];`
  - `char s[25];`
- A linguagem C não incorpora verificação de subscritos, utilizar um valor do subscrito fora do vetor não gera nenhuma mensagem de erro.

EXEMPLO

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

int main()
{
  int n[10], i;

  for(i=0; i<=9; i++)
    n[i] = i;

  printf("%s%13s\n", "Elemento", "Valor");
  for(i=0; i<=9; i++)
    printf("%7d%13d\n",i,n[i]);

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

```
Elemento        Valor
      0            0
      1            1
      2            2
      3            3
      4            4
      5            5
      6            6
      7            7
      8            8
      9            9
Press any key to continue . . .
```

INICIALIZAÇÃO DE VETORES

- Ao igual que as variáveis os elementos de um vetor podem ser inicializados na declaração do vetor,
- para inicializar um vetor utilizamos um sinal de igual e uma lista de inicializadores separados por vírgulas, (entre chaves),

```
int n[5] = {1, 2, 3, 4, 5};
char pal[6] = {'b','r','a','s','i','l'};
```

INICIALIZAÇÃO DE VETORES

- se houver menos inicializadores que o número de elementos do vetor, os elementos restantes são inicializados com zero,

```
float num[10] = {1.0, 2.0, 3.0};
int n[5] = {};
```

- se houver mais inicializadores que termos no vetor teremos um erro de compilação,

```
int n[2] = {1, 2, 3};
```

- se o tamanho do vetor for omitido na declaração, o numero de elementos será igual ao número de inicializadores,

```
float num[] = {1.0, 2.0, 3.0};
```

EXEMPLO

```
Elemento          Valor
      0             32
      1             27
      2             64
      3             18
      4             95
Soma: 236
Press any key to continue . . .
```

```
#include <stdio.h>
#include <stdlib.h>
#define N 5

int main()
{
  int i, n[N] = {32, 27, 64, 18, 95}, soma = 0;

  for(i=0; i<N; i++)
    soma += n[i];

  printf("%s%13s\n", "Elemento", "Valor");
  for(i=0; i<N; i++)
    printf("%7d%13d\n", i, n[i]);
  printf("Soma: %d\n", soma);

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

INICIALIZAÇÃO DE VETORES

- Que acontece se utilizarmos os elementos de um vetor sem ter inicializado eles?

PONTEIROS vs VETORES

- Na linguagem C vetores e ponteiros estão intimamente relacionados,

```
int a[10]
```

- `a`, é um ponteiro inteiro que aponta ao endereço base do vetor,
- `a[0]`, referencia o conteúdo do endereço base do vetor,

PONTEIROS vs VETORES

- no lugar de vetores podemos utilizar ponteiros e alocação dinâmica de memória, este enfoque tem maior complexidade do ponto de vista computacional e uma sobrecarga ao executar o programa,
- a quantidade de memória para um vetor é reservada pelo compilador no ato de compilação do programa,

PONTEIROS vs VETORES

- se a quantidade de elementos do vetor não for conhecida, o maior número de elementos possíveis deverá ser alocado, geralmente esta estratégia produz um uso ineficiente da memória,

- em tais casos existe a alternativa de utilizar ponteiros e alocação dinâmica de memória,

ARMAZENAMENTO - VETORES

- a quantidade memória para armazenar uma matriz é alocada durante a compilação,
- a quantidade de memória alocada dependera do número de elementos e o tipo de dado dos elementos,
- podemos mostrar o endereço de uma matriz imprimindo seu nome,
- podemos mostrar a quantidade de memória ocupada pela matriz utilizando a função **`sizeof(nome_da_matriz)`**

ARMAZENAMENTO - VETORES

- Exemplo:

```
int c[10];
printf("Endereço do array %p\n", c);
printf("Memória em bytes: %d\n", sizeof(c));
```

```
Endereco do array 0022FF10
Memoria alocada em bytes: 40
```

ARMAZENAMENTO - VETORES

```c
#define N 10

int main(){
  int c[N], n;

  printf("Digite o numero de elementos: ");
  scanf("%d", &n);

  if (n>N){
    printf("ERRO, estouro de memoria.");
    return 0;
  }

  /*
   PROCESSAMENTO
  */

  return 0;
}
```

ARMAZENAMENTO - VETORES

```c
#define N 10

int main(){
  int notas[N], i=0, media=0;

  printf("Digite a nota do aluno (%d): ", i+1);
  scanf("%d", &notas[i]);
  while(notas[i]!=-1){
    media +=notas[i];
    i++;
    if (i>=N){
      printf("ERRO, estouro de memoria.");
      return 0;
    }
    printf("Digite a nota do aluno (%d): ", i+1);
    scanf("%d", &notas[i]);
  }
  if(i) printf("Media: %.2f\n", (float)media/i);

  return 0;
}
```

EXEMPLO

Foi perguntado a 20 alunos o nível de qualidade da comida na cantina da UESC, em uma escala de 0 a 5 (0 significa horrorosa e 5 significa excelente). Escreva um programa que resuma os resultados da pesquisa (mostre a quantidade de vezes que aparece cada nota), imprima um histograma com os resultados.

```
#define TAM_R 20
#define TAM_F 6

int main()
{
  int i, j;
  int respostas[TAM_R] = {3, 4, 5, 2, 1, 1, 2, 0, 3, 4,
                          5, 2, 2, 4, 4, 3, 1, 2, 2, 3};
  int frequencias[TAM_F] = {0};

  for(i=0; i<TAM_R; i++)
    ++frequencias[respostas[i]];
  printf("%s%13s%12s\n", "Nivel", "Frequencia", "Histograma");
  for(i=0; i<TAM_F; i++){
    printf("%5d%13d    ", i, frequencias[i]);
    for(j=0; j<frequencias[i]; j++)
      printf("*");
    printf("\n");
  }

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

```
Nivel    Frequencia  Histograma
    0             1    *
    1             3    ***
    2             6    ******
    3             4    ****
    4             4    ****
    5             2    **
```

APLICAÇÕES - OPERAÇÕES ARITMETICAS

- Adição e Subtração de vetores:

$$\vec{c} = \vec{a} \pm \vec{b}$$

$$c[i] = a[i] + b[i]$$

- os vetores a e b devem ter a mesma dimensão,
- o vetor resultante c tem a mesma dimensão que a e b

- Norma do vetor: $\left\|\vec{c}\right\| = \sqrt{\sum_{i=0}^{N-1} \left(c[i]\right)^2}$

APLICAÇÕES - ESTATISTICA

- Media

$$\overline{c} = \frac{\sum_{i=0}^{N-1} c[i]}{N}$$

- Moda: é o elemento que mais se repete entre os elementos do vetor.
- Maximo
- Mínimo
- Mediana: é o elemento "central" de um vetor ordenado (N – par ou N – ímpar)

BUSCA E ORDENAÇÃO

- A operação de busca consiste em informar se um determinado valor existe ou não em um conjunto de elementos (vetor).
- Busca linear (quadro).
- A operação de ordenar (classificar) um vetor consiste em organizar seus elementos de forma que

$$c[0] \leq c[1] \leq \ldots \leq c[i] \leq c[i+1] \leq \ldots \leq c[N-1]$$

- Existem diversos algoritmos de ordenação um dos mais simples é o método de ordenação por bolhas.

CLASSIFICAÇÃO POR BOLHAS

- realizar passagens sucessivas no arquivo, em cada passagem se x[i] > x[i+1], permutar os elementos

Vetor Inicial	25	57	48	37	12	92	86	33
iteração 1	25	48	37	12	57	86	33	92
iteração 2	25	37	12	48	57	33	86	92
iteração 3	25	12	37	48	33	57	86	92
iteração 4	12	25	37	33	48	57	86	92
iteração 5	12	25	33	37	48	57	86	92
iteração 6	12	25	33	37	48	57	86	92
iteração 7	12	25	33	37	48	57	86	92

CLASSIFICAÇÃO POR BOLHAS

```
#define N 8
int main()
{
  int x[N] = {25, 57, 48, 37, 12, 92, 86, 33};
  int i, j, k, aux;

  for(k=0;k<N;k++)
    printf("%4d", x[k]);
  printf("\n");

  for(i=0; i<N; i++){
    for(j=0; j<(N-i-1); j++)
      if (x[j]>x[j+1]){
        aux = x[j];
        x[j] = x[j+1];
        x[j+1] = aux;
      }
  for(k=0;k<N;k++)
    printf("%4d", x[k]);
  printf("\n");
  }
return 0;
}
```

CONSIDERAÇÕES FINAIS

- Matrizes unidimensionais ou vetores são, essencialmente, listas de informações do mesmo tipo, que são armazenadas em posições contíguas na memória em ordem crescente.

- Os elementos do vetor estão relacionados por um índice ou subscrito, através dele acessamos os elementos do vetor.