SI100

Algoritmos e Programação de Computadores I

1º Semestre - 2018

Tópico 5: Vetores

Unidade 9

Prof. Guilherme Palermo Coelho

guilherme@ft.unicamp.br

Roteiro

- Introdução;
- Definições;
- Armazenando dados em um vetor;
- Lendo dados de um vetor;
- Verificação de limites em vetores;
- Inicialização de vetores;
- Busca e ordenação em vetores;
- Exercícios;
- Referências.

Introdução

- **Ex.:** ler as notas de uma classe de 3 alunos e então calcular a média da classe.

```
#include <stdio.h>
int main(){
        float nota0,nota1,nota2,media;

        printf("Digite a nota do aluno 0: ");
        scanf("%f",&nota0);
        printf("Digite a nota do aluno 1: ");
        scanf("%f",&nota1);
        printf("Digite a nota do aluno 2: ");
        scanf("%f",&nota2);

        media = (nota0+nota1+nota2)/3;
        printf("média: %.2f\n", media);
        return 0;
}
```

Introdução

- **Ex.:** ler as notas de uma classe de 3 alunos e então calcular a média da classe.

```
#include <stdio.h>
int main(){
        float nota0,nota1,nota2,media;
        return 0;
}
```

E se a turma de alunos fosse formada por 2000 alunos?

Definiríamos 2000 variáveis simples e repetiríamos os comandos de leitura 2000 vezes?

Inviável!

DEFINIÇÕES

Definições

- **Variável Simples**: uma posição de memória;
  - O tamanho da variável depende do seu tipo;
  - A referência a uma variável significa o acesso ao conteúdo deste pedaço de memória.
- **Vetor**: é um conjunto de posições ***consecutivas*** de memória, identificadas por um mesmo nome, individualizadas por ***índices*** e cujo conteúdo é do ***mesmo tipo***.
  - Ou seja, é um ***conjunto de variáveis simples do mesmo tipo***;
  - Um ***vetor*** é um dos tipos mais simples de estrutura de dados;
  - Os elementos de um vetor são guardados em uma ***sequência contínua*** na memória (um após o outro).

Ex. (vetor de números inteiros):

10	5	7	2	3	6

Definições

- Sintaxe de declaração de um ***vetor*** em C:

```
tipo identificador[número de variáveis];
```

  - **tipo**: é o tipo das variáveis que farão parte do vetor (*int*, *char*, *float* etc.);
  - **identificador**: nome que será usado para referenciar o vetor;
  - **número de variáveis**: quantidade de variáveis que farão parte do vetor;

- Exemplo:
  - **float notas[30];** // declara um vetor chamado 'notas' que possui
    // trinta variáveis do tipo float.

Referenciando um elemento

- Declarado um ***vetor***, precisamos definir como *referenciar* seus elementos individualmente;
  - O **acesso** é feito por "**nome_vetor[posição_elemento]"**;
    - **Ex.**: float nota = notas[24];
- Note que o número entre '[' ']' tem ***significado*** diferente nas seguintes situações:
  - **Declaração do vetor**: significa o tamanho do vetor;
  - **Referência a elemento do vetor**: indica a posição do elemento;
- **ATENÇÃO**: o primeiro elemento do vetor tem índice 0 (zero)!
  - "**notas[3]**" acessa o **quarto** elemento do vetor!

Referenciando um elemento

- **Exemplo 1**:

```
...
int notas[10]; /* declara vetor de inteiros
                *  com 10 posições */
notas[2] = 7;  /* atribui o valor 7 à
                *  posição 3 do vetor */
...
```

- **Exemplo 2**:

```
...
int notas[10];
int i = 2;
notas[i] = 7;  /* atribui o valor 7 à
                *  posição 3 do vetor */
...
```

Armazenando Dados em Vetores

Armazenando dados no vetor

- O armazenamento de dados em uma posição específica de um vetor é feito como em qualquer outra variável;
  - **Ex.**: notas[24] = 8;
  - notas[24] é uma variável como qualquer outra!
- **Exemplo**: leitura de 10 notas do teclado e armazenamento em um vetor

```
...
float notas[10];
int i;
for (i=0; i<10; i++) {
        scanf("%f", &notas[i]);
}
...
```

Cuidado com o número total de elementos do vetor!

Lendo Dados de Vetores

Lendo dados de um vetor

- Estando os dados já armazenados no vetor, o acesso a estes dados também pode ser feito sem dificuldades;
  - **Ex.**: float notaIndividual = notas[24];
- **Ex.**: cálculo da média das 10 notas armazenadas em um vetor;

```
...
float soma = 0.0;
for (i=0; i<10; i++) {
        soma = soma + notas[i];
}
printf("Média das notas: %.2f.\n", soma/10);
...
```

Obs.: se a constante for utilizada em uma variável *float*, é importante sempre defini-la com a casa decimal.

Exemplo

- **Ex.:** ler as notas de uma classe de 100 alunos e então calcular a média da turma.

```
#include <stdio.h>
int main(){
        float notas[100], media= 0.0;
        int i;

        printf(''Entre com as 100 notas:\n'');
        for (i=0; i < 100; i++) {
                scanf(''%f'',&notas[i]);
                media += notas[i];
        }
        media /= 100.0;
        printf(''Média: %5.2f\n'', media);

        return 0;
}
```

Lendo número desconhecido de elementos

- No exemplo anterior, é possível existir uma situação em que não se sabe quantos alunos realmente fizeram provas no semestre;
  - Só se sabe que não existem mais de 100 alunos na turma;
- Como poderíamos escrever um programa que lesse as notas disponíveis para a turma e calculasse a média?
- **Possível solução**: ler as notas até que seja inserida uma nota ***negativa*** (critério de parada);

Lendo número desconhecido de elementos

```
#include <stdio.h>
#define LIM 100
int main(){
        float notas[LIM], media = 0.0;
        int numNotas = 0;
        printf(``Entre com as notas:\n'');
        do {
                if (numNotas >= LIM) {
                        printf("Máximo excedido.\n");
                        numNotas++;
                        break;
                }
                scanf(``%f'',&notas[numNotas]);
                if (notas[numNotas] > 0.0)
                        media += notas[numNotas];
        } while (notas[numNotas++] > 0.0);
        printf(``Média: %5.2f\n'', media/(numNotas-1));
        return 0;}
```

Lendo número desconhecido de elementos

```
#include <stdio.h>
#define LIM 100
int main(){
        float notas[LIM], media = 0.0;
        int numNotas = 0;
        printf(''Entre com as notas:\n'');
        do {
                if (numNotas >= LIM) {
                        printf("Máximo excedido.\n");
                        numNotas++;
                        break;
                }
                scanf(''%f'',&notas[numNotas]);
                if (notas[numNotas] > 0.0)
                        media += notas[numNotas];
        } while (notas[numNotas++] > 0.0);
        printf(''Média: %5.2f\n''
        return 0;}
```

Define uma constante chamada LIM de valor 100

Repete a leitura até encontrar nota negativa

Força saída do laço se LIM for excedido

VERIFICAÇÃO DE LIMITES EM VETORES

Verificação de limites em vetores

- Deve-se tomar ***muito cuidado*** para garantir que o ***tamanho máximo de um vetor*** seja respeitado em um programa em C;
- A linguagem C **não verifica** se um programa respeita ou não o limite definido para um vetor;
  - Se o programa transpuser os limites de um vetor durante uma operação de atribuição, as posições de memória subsequentes ao vetor terão seu conteúdo sobrescrito;
    - **O programa terá comportamento imprevisível!**
- **Cabe ao programador garantir a verificação de limites em vetores.**

Verificação de limites em vetores

- Exemplo:

```
#include <stdio.h>

int main() {
        int vetor[3], i;
        vetor[0] = 1;
        vetor[1] = 2;
        vetor[2] = 3;
        for (i=0; i<=3; i++)
                printf("Valor=%d\n", vetor[i]);

        return 0;
}
```

```
Valor=1
Valor=2
Valor=3
Valor=1670971486
```

Inicialização de Vetores

Inicialização de vetores

- Assim como variáveis simples, é possível atribuir um ***valor inicial*** a um vetor no momento de sua criação;
- Isto pode ser feito da seguinte maneira:
  - **Ex.**: int vetor[3] = {50, 25, 10};
    - No exemplo acima, é criado um vetor de 3 posições;
    - As posições são inicializadas com os valores 50 (posição 0), 25 (posição 1) e 10 (posição 2).
  - **Ex.**:int vetor[] = {50, 25, 10};
    - O exemplo acima funciona da mesma forma que o anterior;
    - O tamanho de **vetor** é identificado pelo compilador pelo número de valores passados na inicialização.

BUSCA EM VETORES

Busca em vetores

- Um problema comum quando se trabalha com vetores é a necessidade de verificar se um elemento com um determinado valor está armazenado no vetor;
  - Este problema é conhecido como ***busca em vetores***;
- A forma mais simples de realizar busca em um vetor é ***percorrer todos os elementos do vetor até encontrar* (*ou não*) o valor procurado***;
  - Esta forma de busca é chamada de **busca linear**;
  - No pior caso, ***todos*** os elementos do vetor devem ser verificados.
  - Existem outras formas de busca em vetores mais eficientes, que serão tratadas em cursos futuros.

Busca em vetores

- Exemplo:

```
#include <stdio.h>

int main() {
        int vetor[8]= {7, 10, 5, 4, 3, 6, 8, 9};
        int x = 8;
        int i = 0;

        while ((vetor[i]!=x) && (i<8))
                i++;
        if (i<8)
                printf("Valor na pos. %d\n", i);
        else
                printf("Valor não encontrado.\n");

        return 0;
}
```

Ordenação de Vetores

Ordenação de vetores

- Além da busca em vetores, outro problema muito comum é a necessidade de ***ordenar*** um dado vetor de dados;
  - **Ex.**:
    - Entrada: |5|4|2|6|3|9|
    - Saída: |2|3|4|5|6|9|
- Existem diversos ***algoritmos de ordenação*** propostos, que realizam a operação de ordenação com maior ou menor eficiência;
  - **Eficiência**: número de comparações entre elementos, quantidade de memória adicional necessária etc.
- Serão vistos aqui dois algoritmos: ***ordenação por seleção*** e ***ordenação por permutação*** (*bubble sort*);

Ordenação por Seleção

- Conhecido pelo nome em inglês ***selection sort***, é a forma mais intuitiva de ordenação;
- **Ideia básica**: percorrer o vetor várias vezes, trocando o maior valor encontrado pelo da última posição não utilizada;
  - **Ex.**: |5|4|9|6|3|2|
    - Passagem 1:
      - Última posição: 5 (percorre de 0 a 5);
      - Maior valor encontrado: 9 (troca entre posições 2 e 5);
      - Vetor resultante: |5|4|2|6|3|9|
    - Passagem 2:
      - Última posição: 4 (percorre de 0 a 4);
      - Maior valor encontrado: 6 (troca entre posições 3 e 4);
      - Vetor resultante: |5|4|2|3|6|9|

Ordenação por Seleção

- **Passagem 2**: |5|4|2|3|6|9|
- Passagem 3:
  - Última posição: 3 (percorre de 0 a 3);
  - Maior valor encontrado: 5 (troca entre posições 0 e 3);
  - Vetor resultante: |3|4|2|5|6|9|
- Passagem 4:
  - Última posição: 2 (percorre de 0 a 2);
  - Maior valor encontrado: 4 (troca entre posições 1 e 2);
  - Vetor resultante: |3|2|4|5|6|9|
- Passagem 5:
  - Última posição: 1 (percorre de 0 a 1);
  - Maior valor encontrado: 3 (troca entre posições 0 e 1);
  - Vetor resultante: |2|3|4|5|6|9|

Ordenação por Seleção

```
#include <stdio.h>

int main() {
        int vetor[6]= {5, 4, 9, 6, 3, 2};
        int i,j,jm,temp;

        for (j=5; j >= 1; j--){
                jm =j;
                for (i=0; i < j; i++){ // Encontra maior
                        if (vetor[i] > vetor [jm])
                                jm =i;
                }
                if (j != jm){ // Troca valores
                        temp = vetor[jm];
                        vetor [jm] = vetor [j];
                        vetor [j] = temp;
                }
        }
        return 0;}
```

Ordenação por Permutação

- O algoritmo de ordenação por permutação é mais conhecido pelo seu nome em inglês ***bubble sort***;
- **Ideia básica**: replicar o efeito de bolhas de ar dentro da água;
  - Os maiores indivíduos vão para as posições finais do vetor;
  - São feitas comparações entre pares de indivíduos e, caso estejam em ordem invertida, suas posições são trocadas.
- **Ex.**: |5|4|9|6|3|2|
  - **Passagem 1:** último elemento na posição 5
    - |5|4|9|6|3|2| → Realiza troca!
    - |4|5|9|6|3|2| → Não realiza troca
    - |4|5|9|6|3|2| → Realiza troca!
    - |4|5|6|9|3|2| → Realiza troca!
    - |4|5|6|3|9|2| → Realiza troca e encerra passagem.

Ordenação por Permutação

- **Ex.**: |5|4|9|6|3|2|
  - **Passagem 2:** último elemento na posição 4
    - |4|5|6|3|2|9| → Não realiza troca
    - |4|5|6|3|2|9| → Não realiza troca
    - |4|5|6|3|2|9| → Realiza troca!
    - |4|5|3|6|2|9| → Realiza troca e encerra passagem.
  - **Passagem 3:** último elemento na posição 3
    - |4|5|3|2|6|9| → Não realiza troca
    - |4|5|3|2|6|9| → Realiza troca!
    - |4|3|5|2|6|9| → Realiza troca e encerra passagem.
  - **Passagem 4:** último elemento na posição 2
    - |4|3|2|5|6|9| → Realiza troca!
    - |3|4|2|5|6|9| → Realiza troca e encerra passagem.

Ordenação por Permutação

- **Ex.**: |5|4|9|6|3|2|
  - **Passagem 5:** último elemento na posição 1
    - |3|2|4|5|6|9| → Realiza troca e encerra algoritmo

  - **Vetor final**: |2|3|4|5|6|9|.

Ordenação por Permutação

```
#include <stdio.h>

int main() {
        int vetor[6]= {5, 4, 9, 6, 3, 2};
        int i,j,temp;

        for (j=5; j > 0; j--) { //j: limite da última posição

                for (i=0; i < j; i++){
                        // Compara par a par:
                        if (vetor[i] > vetor[i+1]) {
                                temp = vetor[i];
                                vetor[i] = vetor[i+1];
                                vetor[i+1] = temp;
                        }
                }
        }
        return 0;}
```

EXERCÍCIOS

Exercícios

1. Escreva um programa que armazene o quadrado de todos os números inteiros de 1 a 20 em um vetor e depois imprima os valores armazenados em ordem **decrescente**, um por linha.

2. Escreva um programa que leia 30 notas, calcule a média e exiba na tela a média calculada e os alunos (com as respectivas notas) que tiveram notas acima da média da turma.

3. Escreva um programa que leia 20 notas, armazene-as em um vetor e então diga se uma determinada nota **x** digitada pelo usuário existe no vetor. O programa deve repetir o procedimento de leitura de **x** e busca de **x** no vetor até que o usuário digite um valor negativo.

Exercícios

4. Uma determinada loja comercializa 10 produtos diferentes. Escreva um programa que receba a quantidade vendida e o valor unitário de cada produto e os armazene em dois vetores. Após a leitura, o programa deve calcular o total de produtos vendidos e o faturamento total a loja, e exibi-los na tela.

5. Faça um programa que receba a temperatura média de cada mês do ano (em ordem) e as armazene em um vetor. Em seguida, este programa deve calcular e exibir a maior e a menor temperatura média do ano e em que mês elas ocorreram. O mês de ocorrência deve ser escrito por extenso.

Exercícios

6. Escreva um programa que carregue dois vetores com 10 valores inteiros cada um e mostre o vetor resultante da intercalação destes dois vetores:

   Exemplo:

   Vetor 1: |3|5|4|2|2|5|3|2|5|9|

   Vetor 2: |3|5|4|2|2|5|3|2|5|9|

   Saída: |3|3|5|5|4|4|2|2|2|2|5|5|3|3|2|2|5|5|9|9|

7. Escreva um programa que leia um vetor com 8 números inteiros, calcule e mostre o resultado da separação deste vetor em dois outros vetores: o primeiro deve conter apenas os números **positivos** enquanto que o segundo vetor deve conter apenas os números **negativos** (**dica**: cada vetor resultante terá no máximo 8 valores, sendo que nem todas as posições poderão ser utilizadas).

Exercícios

8. Modifique o algoritmo *BubbleSort* visto em aula para que ele ordene os valores de um vetor em ordem ***decrescente***. Implemente o algoritmo modificado e o teste em um programa que leia os valores de um vetor de 10 posições e os exiba em ordem decrescente na tela.

Exercícios

- **Observação**:
  - Os seguintes exercícios devem ser entregues via SuSy.
    - Exercício 3;
    - Exercício 4;
    - Exercício 6.
  - Veja os enunciados atualizados no site do sistema:
    - https://susy.ic.unicamp.br:9999/si100a (Turma A);
    - https://susy.ic.unicamp.br:9999/si100b (Turma B).

REFERÊNCIAS

Referências

- MIZRAHI, V. V., *Treinamento em Linguagem C – Curso Completo*. 2a Edição, Pearson Makron Books, 2005.
- ASCENCIO, A. F. G. & DE CAMPOS, E. A. V., *Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++*. Pearson Prentice Hall, 2003.
- FALCÃO, A. X.,. *MC102: Algoritmos e Programação de Computadores – Notas de Aula*. Instituto de Computação, Unicamp, 2005.