Programação Estruturada Aula 8 - Matrizes e Strings

Prof. Luis Carlos Retondaro

Técnico em Telecomunicações
2º Ano

CEFET/RJ - Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

Campus Petrópolis

2017

Sumário

1. Vetores
2. Strings
3. Matrizes
4. Referências

Motivação

- Em diversas situações os tipos básicos de dados (int, char,...) não são suficientes para representar a informação que se deseja armazenar;
- **Exemplo:**
  - Armazenar uma palavra: “AULA”;
  - Armazenar as notas de todos os alunos;
  - Armazenar os dados dos alunos de uma turma;
  - Armazenar os gastos mensais por um período de um ano;
  - etc...

Motivação

- **Problema :** definir a média das notas de 50 alunos de uma turma.

```
#include<stdio.h>
#define N 50

void main(){
   int i;
   float nota, media=0;

   for(i=0; i<N; i++){
      printf("Digite a nota\n");
      scanf("%f", &nota);
      media = media+nota;
      //media += nota;
   }
   media/=N;
   printf("Média: %0.2f\n", media);
}
```

Motivação

Problema!

- Mas se além da média quiséssemos fazer um programa imprimir as notas lidas juntamente com a média da turma?
- Quando uma determinada Estrutura de Dados for composta de variáveis com o mesmo tipo primitivo, temos um conjunto homogêneo de dados.

Conceitos de vetores em C

- Um vetor é uma coleção de variáveis, do mesmo tipo e que são referenciadas por um mesmo identificador;
- Um elemento específico de um vetor é referenciado (acessado) por meio do índice, que indica a posição desse elemento;
- Em C todos os vetores consistem em posições contíguas na memória, o endereço mais baixo representa o primeiro elemento do vetor e o mais alto o último;
- Vetores e matrizes de qualquer dimensão possuem todos os elementos do mesmo tipo de dado!
- Um vetor comum é a string.

Declaração de vetores

- Um vetor nada mais é que uma matriz que contém apenas uma dimensão, como por exemplo uma string;
- Quando se faz uma declaração de string em C, na verdade declara-se um vetor de caracteres;
- Forma geral:

```
tipo nome_do_vetor[tamanho_do_vetor];
```

- Assim como as demais variáveis, o vetor deve ser declarado para que o compilador aloque espaço para o mesmo na memória.

Declaração de vetores

- Exemplos:

```
int vet[80];
```

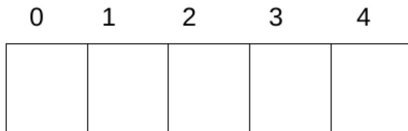
- 80 é o tamanho do vetor, a declaração reserva 80 "gavetas" consecutivas de memória que correspondem ao tamanho do vetor e cada "gavetas" guarda um int;

```
float vet[20];
```

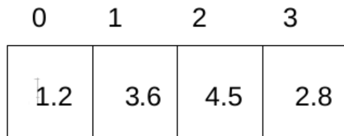
- A declaração acima reserva 20 "gavetas" consecutiva de memória, cada uma guarda um float.

Declaração de vetores

```
int dados[5];
```




```
float valores[4];
```



Declaração de vetores

- Importante!
- Na declaração de vetor, o valor que se encontra entre os colchetes pode ser um número constante ou uma variável;
- Portanto, se usar variável, cuidado ao modificar o valor da variável que armazena o tamanho do vetor.

```
//possíveis declarações de vetores

char nome[30];

//ou
int n = 30;
char nome[n];

//ou
int n;

printf("Entre com o tamanho máximo do nome\n");
scanf("%d", &n);
char nome[n];
```

Vetores na memória

- Vetores é uma coleção de variáveis mesmo tipo, acessíveis com um único nome, que são armazenadas em posições contíguas de memória;
- A individualização de elemento de um vetor é feita através do uso de índices.
- Um índice é sempre um número natural;
- Ilustração de um vetor se ele começasse na posição 1000 de memória e fosse declarado como char a[5];

Elemento	a[0]	a[1]	a[2]	a[3]	a[4]
Endereço	1000	1001	1002	1003	1004

Percorrendo e lendo vetores

- Percorrer um vetor significa varrer o vetor de casa em casa a partir do índice 0 (zero);
- Para percorrer um vetor é necessário conhecer o tamanho do vetor;
- Para a leitura do vetor deve-se ler elemento a elemento usando o padrão de percorrimento do vetor;
- A leitura é feita com a mesma sintaxe de uma variável única, porém deve-se identificar o índice onde será o armazenamento.

```
scanf("%f", &v[i]);
```

Vetores

Acesso aos elementos

- Para acessar os elementos de um vetor usa-se índices;
- O índice define a posição da variável dentro do vetor;
- Todos os vetores tem o primeiro elemento na posição 0 (zero), assim, se tomarmos "n" como sendo o tamanho do vetor, sua última posição é a de índice "n-1";
- Acesso aos elementos dos vetores:

```
float notas[100];
. . .
notas[0] = 4.5;                //insere 4.5 na primeira posição do vetor
notas[10] = 7.8;               //insere 7.8 na décima posição do vetor
```

Vetores

- **Exemplo:** Cálculo da média da nota de n alunos.

```
//cálculo da média aritmética de n alunos
#include <stdio.h>
void main(){
  float notas[100], media = 0.0;
  int i, n;

  printf("Digite a quantidade de alunos\n");
  scanf("%d", &n);
  for(i=0;i<n;i++){
    printf("Entre com a %d a nota:  ",i+1);
    scanf("%f",&notas[i]);
  }
  for(i=0;i<n;i++)
    media += notas[i];
  media/=n;
  printf("\nMédia das notas da turma %3.2f\n",media);
  printf("Nota de cada aluno:\n");
  for(i=0;i<n;i++)
    printf("aluno %d \t nota %3.2f\n", i, notas[i]);
}
```

Vetores

- **Exemplo:** Melhoramento do programa de cálculo da média da nota de *n* alunos.

```
//cálculo da média aritmética de n alunos
#include <stdio.h>
void main(){
 float notas[100], media = 0.0;
 int i, n=0;

 while(n<1 || n>100){
   printf("Digite a quantidade de alunos\n");
   scanf("%d", &n);
 }
 for(i=0;i<n;i++){
   printf("Entre com a %d a nota:  ",i+1);
   scanf("%f",&notas[i]);
 }
 for(i=0;i<n;i++)
   media += notas[i];
 media/=n;
 printf("\nMédia das notas da turma %3.2f\n",media);
 printf("Nota de cada aluno:\n");
 for(i=0;i<n;i++)
   printf("aluno %d \t nota %3.2f\n", i, notas[i]);
}
```

Vetores

- A quantidade de memória necessária para guardar um vetor está diretamente relacionada com seu tamanho e seu tipo;
- Para uma matriz unidimensional (vetor) o tamanho total em bytes é calculado da forma;

```
totalBytes = sizeof(tipo) * tamanhoMatriz;
```

- C não tem verificação de limites em vetores, fica a cargo do programador!
- Quando o programador não toma o devido cuidado com os limites de um vetor, pode ocorrer de sobrescrever dados que estão em memória ativa.

Vetores

- Exemplo de utrapassagem de limites:

```
#include <stdio.h>

void main(){
  int vet[10], i;

  printf("Elemnetos do vetor:  ");
  for(i=0; i<100; i++){
    vet[i] = i*3;
    printf("%d  ",vet[i]);
  }
}
```

- A compilação NÃO exibirá um erro, porém a execução dará falha de segmentação pois tenta fazer o acesso à memória que não foi reservada.

Vetores

- O índice do vetor pode ser uma expressão aritmética;

```
#include <stdio.h>
void main(){
  float x[80];
  int i;

  //inicializando o vetor
  for(i=100; i<180; i++)
    x[i-100] = 0;
}
```

- Ao utilizar uma expressão, deve-se ter certeza que o resultado será um índice válido para o vetor declarado.

Exemplo utilidade de vetores

- Exemplo de como os vetores podem ser úteis: um vetor pode guardar os coeficientes de um polinômio de grau n;

```
...
float P[100];
int n;
```

- Dessa forma $p[0], p[1], ..., p[n]$ guardam os coeficientes de um polinômio $p(x) = p_0 + p_1 * x + ... + p_n * x^n$;
- É claro que nesse caso particular n não pode ser maior que 99.

Strings

- Em C uma string é defina como uma matriz de caracteres que é terminada por um nulo;
- Um nulo é especificado como '\0';
- Dado a forma de término de uma string sempre é necessário declarar vetores de caracteres de tamanho maior que a maior string que ela pode armazenar;
- Exemplo: a declaração de uma string que pode armazenar até 10 caracteres deve ser feita da forma:

```
char str[11];
```

Strings

- Dado a forma de término de uma string sempre é necessário declarar vetores de caracteres de tamanho maior que a maior string que ela pode armazenar.

0	1	2	3	4	5	6
J	O	Ã	O	\0		

Strings

- Para declarar uma string, aloca-se um espaço na memória definindo um vetor de caracteres;
- Para algumas operações, uma string se parece como uma variável simples, como por exemplo impressão;

```
char nome[20];
...
printf("%s\n", nome);
```

- Porém algumas operações com strings estão ausentes na linguagem C, como: Atribuição a uma variável, concatenação, comparação, etc.

Strings

- Embora o C não tenha o tipo de dado string a linguagem permite constantes string;
- Uma constante string é uma lista de caracteres entre aspas "Aula de C";
- Não é necessário adicionar o nulo no final das constantes string manualmente, o compilador C faz isso automaticamente;
- C suporta uma ampla gama de funções para manipulação de strings.

Strings

- Limitações na manipulação de strings:

```
char curso[20], nome[20];
...

curso = "linguagem C"; // atribuição ilegal em C
curso < nome // comparação ilegal em C
curso = curso + nome; // concatenação ilegal em C
```

Strings

- A biblioteca padrão string da linguagem C contém várias funções de manipulação de strings. Para usar essas funções, o seu programa deve incluir string.h;

```
#include <string.h>
```

- **strlen(s1):** recebe uma string e retorna o seu comprimento;
- **strcpy(s1, s2):** recebe duas strings e copia o conteúdo da segunda para a primeira;
- **strcat(s1, s2):** concatena a segunda string ao final da primeira.

Strings

- A biblioteca padrão string da linguagem C contém várias funções de manipulação de strings. Para usar essas funções, o seu programa deve incluir string.h;

```
#include <string.h>
```

- **strcmp(s1, s2):** compara duas strings, retorna 0 se são iguais, menor que zero se $s1 < s2$ e maior que zero se $s1 > s2$;
- **strchr(s1, c):** procura um caracter em uma string, retorna um ponteiro para a primeira ocorrência de c em s1;
- **strstr(s1, s2):** procura uma string dentro de outra string, retorna um pontteiro para a primeira ocorrência de s2 em s1.

Strings

- Exemplo:

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

void main(){
 char s1[80], s2[80];
 int tam1, tam2;

 printf("Digite duas strings\n");
 gets(s1);
 gets(s2);

 tam1 = strlen(s1);
 tam2 = strlen(s2);
 printf("\ncomprimentos: s1 = %d  s2 = %d\n",tam1, tam2);
 if(!strcmp(s1, s2))
   printf("As strings são iguais!\n");
 strcat(s1,s2);
 printf("concatena: %s\n",s1);
 if(strchr("s1", 'o'))
     printf("O cacracter 'o' está em s1\n");
 if(strstr("ola aqui estamos", "tamos"))
     printf("tamos foi encontrado!\n");
}
```

Strings

- Saída da execução:

```
Digite duas strings
o mercado
está vazio faz tempo

comprimentos: s1 = 9  s2 = 21
concatena: o mercado está vazio faz tempo
tamos foi encontrado!
```

Matrizes com duas dimensões

- Como os vetores, as matrizes são estruturas de dados homogêneos;
- Podem ser construídas dos diversos tipos básicos primitivos;
- A principal diferença é a dimensão, onde matrizes possuem uma ou mais dimensões;
- **Exemplo:** jogos como xadrez, campo minado, jogo da velha, etc, problemas matemáticos, etc.

Matrizes com duas dimensões

- A matriz bidimensional é uma matriz de matrizes unidimensionais;
- Exemplo de declaração de matrizes bidimensionais:

```
int mat[10][20];   //matriz 10x20

double matR[5][5];      //matriz 5x5
```

- Similar ao vetor, a forma de acesso de uma determinada posição é feita informando em cada colchete a linha e coluna desejada;

```
a = mat[1][3];
```

Matrizes com duas dimensões

- Matrizes bidimensionais são armazenadas em uma matriz linha-coluna, onde o primeiro índice indica a linha e o segundo a coluna;
- Sendo assim o índice mais a direita varia mais rapidamente que o mais a esquerda, quando o acesso é realizado na ordem em que eles estão armazenados na memória;
- O cálculo do tamanho de uma matriz bidimensional pode ser feito da forma:

```
bytes = número_de_linhas * número_de_colunas * sizeof(tipo_de_dado)
```

- Para uma matriz de inteiros com dimensão $10x5$ tem-se: $10 * 5 * 2 = 100$ bytes alocados.

Matrizes com duas dimensões

Exemplo

Memória

Colunas

Linhas	0	1	2	3	4
0	65	10	39	34	20
1	55	69	29	33	65
2	43	34	77	40	35
3	91	24	40	97	11
4	88	61	21	70	100

COLUNAS

LINHAS

25 Espaços

Matrizes com duas dimensões

- Exemplo de leitura e impressão de uma matriz bidimensional.

```
#include <stdio.h>
#define TAM 5

 void main(){
  int mat[5][5], i, j;

  printf("Entre com os valores de uma matriz %dX%d\n",TAM,TAM);
  for(i=0; i<TAM; i++)
    for(j=0; j<TAM; j++)
      scanf("%d", &mat[i][j]);

  printf("\nMatriz somando 1:\n");
  for(i=0; i<TAM; i++){
    for(j=0; j<TAM; j++)
      printf("%d  ",mat[i][j]+1);
    printf("\n");
  }
}
```

Matrizes com duas dimensões

- Resultado da execução.

```
Entre com os valores de uma matriz 5X5
1 2 3 4 5
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
6 6 6 6 6

Matriz somando 1:
2   3   4   5   6
2   2   2   2   2
3   3   3   3   3
4   4   4   4   4
7   7   7   7   7
```

Matrizes bidimensionais como parâmetro

- Quando uma matriz bidimensional é usada como argumento para uma função, apenas um ponteiro para o primeiro elemento é realmente passado;
- Uma função que recebe uma matriz bidimensional como parâmetro deve definir pelo menos o comprimento da segunda dimensão;
- O compilador C precisa saber o comprimento de cada linha para poder indexar a matriz corretamente;

```
//Exemplo funcao que recebe uma matriz de inteiros 20x10

void func(int mat[][10]){
...
}
```

Matrizes bidimensionais como parâmetro

- O comprimento da primeira dimensão pode ser especificada, mas não é necessário;
- O compilador precisa saber a segunda dimensão para poder trabalhar com sentenças do tipo mat[10][5].

```
//Exemplo funcao que recebe uma matriz de inteiros 20x10

void func(int mat[][10]){
...
}
```

Matrizes com duas dimensões

- Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h> //testar caracter
#define CLASSE 3
#define GRADES 30

int grade[CLASSE][GRADES];

//digita a nota dos alunos
void enterGrades(){
  int i,j;

  for(i=0;i<CLASSE; i++){
    printf("Turma # %d:\n", i+1);
    for(j=0;j<GRADES;j++)
      grade[i][j] = getGrade(j);
  }
}

//continua...
```

Matrizes com duas dimensões

- Exemplo:

```c
//...continuação

//lê uma nota
getGrade(int num){
  char s[80];
  printf("Digite a nota do aluno # %d: \n",num+1);
  gets(s);
  return(atoi(s));
}

//Mostra as notas
void dispGrades(int g[][GRADES]){
  int i,j;

  for(i=0;i<CLASSE; i++){
    printf("Turma # %d:\n", i+1);
    for(j=0;j<GRADES;j++)
      printf("Aluno #%d é %d\n", j+1, g[i][j]);
  }
}

//continua ...
```

Matrizes com duas dimensões

- Exemplo:

```
//...continuação

void main(){
  char ch, str[80];
  for(;;){
    do{
      printf("D. Digitar notas\n");
      printf("M. Mostrar notas\n");
      printf("S. Sair\n");
      gets(str);
      ch = toupper(*str);
    }while(ch!='D' && ch != 'M' && ch != 'S');
    switch(ch){
      case 'D':
       enterGrades();
       break;
      case 'M':
       dispGrades(grade);
       break;
      case 'S':
       exit(0);
    }
  }
}
```

Referências

1. C Completo e Total, Herbert Schidt; Pearson Makron Books; 3a. Ed., 1997.
2. Linguagem C. DAMAS, Luis. 10a. Edição. LTC, 2014.