Algoritmos e Programação

Turmas: Engenharia de Alimentos e Engenharia Civil

Vetores

Aula 25/05/2015 (Engenharia de Alimentos)

Aula 27/05/2015 (Engenharia Civil)

Vetores (Arranjos, ou Arrays, em inglês)

Em algoritmos e programação, um vetor, também chamado de arranjo, ou *array*, em inglês, é um tipo especial de variável que permite a solução de problemas que envolvem **coleções de valores do mesmo tipo**.

Dessa forma, um vetor é uma variável que referencia uma coleção de valores.

Vetores

Exemplos de problemas:

- Suponha que você queira calcular a média de uma turma com 30 alunos e imprimir a nota de cada aluno com uma mensagem informando se ele está acima da média ou não. Sem o uso de vetores, será necessário declarar 30 variáveis distintas e realizar 30 comandos de leitura.

Vetores - Exemplos de problemas

Alguém pode argumentar que esse problema pode ser resolvido com somente 1 variável para ler as notas dos alunos. Isso seria possível se desejássemos apenas calcular a média de todos os alunos. Mas como desejamos comparar a média (que só pode ser calculada após lermos todas as notas) com a nota de cada aluno, então torna-se necessário armazenar as notas de cada aluno (as 30 variáveis).

Vetores - Outro exemplo

Em estatística, a **moda** é o valor que ocorre com mais frequência num conjunto de dados.

O problema do cálculo da moda de um conjunto de dados também só pode ser resolvido com vetores.

Exemplo:

Entrada: 1 2 3 4 5 1 2 3 4 4

Saída: A moda eh 4

Vetores

Esses dois exemplos ilustram o fato dos vetores serem utilizados na resolução de problemas com coleções de valores. Na matemática, uma coleção de valores onde a ordem dos elementos é relevante, chama-se de **arranjo**.

Vetores

Também, na matemática, uma coleção de valores onde a ordem não é relevante e não há elementos repetidos, dá-se o nome de **conjunto** (*set*, em inglês).

Veremos, posteriormente, que vetores podem ser utilizados na construção de conjuntos.

Vetores (Outros tipos)

Na linguagem C, quando um vetor é utilizado para armazenar caracteres (letras e números, representados pelo tipo **char**), esse vetor é chamado de **string**. Veremos que, a linguagem C não define um **tipo de dados** string, mas que um vetor de caracteres, terminado pelo caracter nulo **\0** constitui uma string.

Vetores (Outros tipos)

Até o momento, mencionamos apenas os vetores unidimensionais (de uma dimensão). Mas um vetor pode ter **n** dimensões. Os vetores de duas dimensões são chamados de **matrizes** (*matrix*, em inglês).

Os vetores podem possuir muitas dimensões (lembre-se que o limite de uma variável pode ser restringido pelo compilador, pelo sistema operacional, pela arquitetura do computador ou pela quantidade de memória do computador).

Vetores (Dimensões)

Um vetor de uma dimensão e tamanho **n** ocupará um espaço de memória da ordem **n**

Um vetor de duas dimensões e tamanho **n** ocupará um espaço de memória da ordem **n^2** (n ao quadrado)

Um vetor de m dimensões e tamanho **n** ocupará um espaço de memória da ordem de **n^m** (n elevado a m)

Vetores (Dimensões)

Quando nos referimos ao espaço de memória **da ordem de n**, isso é porque cada posição **n** ocupa um espaço de memória de acordo com o tipo de dados. Seguem informações sobre alguns tipos:

Vetores (Tamanhos dos principais tipos de dados)

- O tipo char ocupa 1 byte
- O tipo int ocupa 4 bytes
- O tipo float ocupa 4 bytes
- O tipo double ocupa 8 bytes
- O tipo long double ocupa 10 bytes

Questões???

- Para quê servem vetores?
- Vetores armazenam variáveis de quais tipos?
- Quantas variáveis um vetor pode armazenar?
- O que são as dimensões de um vetor?

Declaração de Vetores

Declaração de Vetores

Vetores são um tipo de dados especial para o armazenamento de coleções de valores do **mesmo tipo de dados**.

Podemos ter um vetor de inteiros, um vetor de caracteres, um vetor de números em ponto-flutuante etc.

Mas como todas as variáveis, antes de utilizá-las, devemos declará-las.

Declaração de Vetores

Vimos em nosso Curso de Algoritmos e Programação que declarar uma variável é reservar um espaço de memória, onde será armazenado um valor de determinado tipo. Esse endereço de memória, tipo e valor serão associados ao **nome da variável**. Exemplo

```
1   float media; //reserva 4 bytes para armazenar um
2                //número em ponto-flutuante e
3                //associa o nome media
4                //a esse espaço de memória
5                //que possui valor indeterminado
6                //porque não foi inicializado
```

Declaração de Vetores

A declaração de um vetor é similar, mas ao invés de reservarmos somente uma posição de memória, um vetor reserva:

n posições de memória (quando possui dimensão 1)

ou n1 * n2 * ...n_m posições de memória quando possu
m dimensões de tamanhos n1, n2, ..., n_m.

Declaração de Vetores

A declaração de vetores é similar a de variáveis, com a diferença de que o número de elementos do vetor (seu tamanho) deve ser posto entre colchetes, após o nome do vetor.

Em pseudo-código:

```
1    variaveis
2        v        : vetor[10] de inteiro;
```

Declaração de Vetores

Em linguagem C:

```
1   int  vetor[10]; //declara um vetor de
2                   //inteiros com 10 posições
```

Declaração de Vetores Multidimensionais

- Um vetor com 2 dimensões, com 10 elementos em cada dimensão

Em pseudo-código:

```
1   variaveis
2       v        : vetor[10, 10] de inteiro;
```

- Um vetor com 2 dimensões, com 10 elementos em cada dimensão

Em linguagem C:

```
1   int  vetor[10][10]; //declara um vetor de
2                       //inteiros com 10 elementos
3                       //em cada dimensão
4   /* vetor é apenas um nome. A variável poderia ser
5      qualquer identificador válido utilizado na
6      linguagem C */
```

Tipos de Dados

Um vetor só armazena valores do mesmo tipo de dados fornecidos em sua declaração.

É o tipo de dados que também define o espaço em memória que o vetor vai ocupar.

Tipos de Dados

Nos exemplos anteriores, um vetor de 1 dimensão e 10 elementos e um vetor de 2 dimensões com 10 elementos cada, ambos do tipo inteiro, o espaço utilizado em memória será:

- 40 bytes para o vetor unidimensional (4 * 10);
- 400 bytes para o vetor de 2 dimensões (4 * 10 * 10);

Tipos de Dados

Se os vetores fossem do tipo **char**, ocupariam:

- 10 bytes para o vetor unidimensional (1 * 10);
- 100 bytes para o vetor de 2 dimensões (1 * 10 * 10);

Tipos de Dados

Nesse tipo de declaração, chamada de estática, todo o espaço de memória utilizado deve ser reservado antes de ser utilizada, e portanto, conhecida *a priori*.

Em cursos avançados de algoritmos, aprende-se a fazer *alocação dinâmica de memória*, quando não se sabe *a priori* quanta memória será utilizada, podendo realizar esse processo sob demanda (reservar e liberar memória conforme necessário).

Exercícios

- Declare um vetor de 1 dimensão, do tipo float, chamado notas e capaz de armazenar 65 elementos;

- Declare um vetor de 1 dimensão, do tipo char, chamado gabarito e capaz de armazenar 65 elementos;

Exercícios

- Declare um vetor de 1 dimensão, do tipo char, chamado nome e capaz de armazenar 81 elementos (**lembre-se que na linguagem C é necessário armazenar uma posição extra para o \0. Logo, esse vetor pode armazenar um nome com até 80 letras**);

Endereços de memória

- Uma variável é um nome que aponta para um endereço de memória. Esse endereço não é conhecido em **tempo de compilação**. Ele só é definido em **tempo de execução** (quando o sistema operacional carrega o programa na memória).
- Logo, o endereço **real** de uma variável pode ser distinto em duas execuções, mesmo que consecutivas.

Endereços de memória

Como não é possível conhecer o endereço real de uma variável (essa a razão delas existirem, para evitarmos saber qual o endereço onde um valor está armazenado), usaremos em nossos exemplos o endereço **1000**, para fins de exemplificação.

Endereços de memória

```
1           int x, y;
```

- Neste exemplo, a variável x está na posição de memória 1000 e a variável y está na posição de memória 1004
- x ocupa 4 bytes (1000, 1001, 1002 e 1003).
- y também ocupa 4 bytes (1004, 1005, 1006, 1007)

Endereços de memória

```
1        int x[100], y;
```

- Neste exemplo, a variável x está na posição de memória 1000 e a variável y está na posição de memória 1400
- x ocupa 400 bytes (1000, 1001, 1002.....1399).
- y também ocupa 4 bytes (1400, 1401, 1402, 1403)

Indexação

Até o momento, aprendemos apenas a reservar as posições de memória para uma coleção de elementos do mesmo tipo. Também dizemos que vetores armazenam coleções **homogêneas**.

Quando desejamos acessar (ler ou escrever) apenas um elemento do vetor utilizamos um processo chamado de **indexação**.

Indexação

De fato, o processo de indexação é a única maneira de manipular vetores em pseudo-código e na linguagem C

A indexação é o processo de referenciamento de um elemento do vetor através do uso de um **índice**, que é um valor inteiro e positivo.

Indexação

A indexação é realizada através de um processo também chamado de **subscrição**, onde o índice inteiro e positivo é posto entre colchetes, após o nome da variável.

Somente um elemento do vetor pode ser acessado por vez

```
 1   char gabarito[5]; //declara um vetor,
 2   //do tipo char, com 5 elementos
 3
 4   gabarito[0] = 'E'; //o primeiro elemento de um
 5   //vetor é representado pelo índice 0.
 6
 7   gabarito[1] = 'B'; //1 é o segundo elemento
 8
 9   gabarito[2] = gabarito[0]; //um vetor também pode
10   //ser lido utilizando o mesmo método de subscrição
11
12   gabarito[3] = 'A';
13
14   gabarito[4] = 'C'; //último elemento do vetor
15   //um vetor de tamanho 5, tem índices de 0 a 4
```

Indexação

O processo de indexação é trivial:

- Lembre-se que uma variável é um endereço de memória;
- A variável sempre aponta para o primeiro byte do espaço de memória que ocupa;
- Assim, uma variável **i**, do tipo inteira, ocupa os bytes 1000, 1001, 1002 e 1003, mas **i** só aponta para o valor 1000. Os outros valores estão reservados implicitamente.

Indexação

Recapitulando um exemplo anterior, de um vetor do tipo inteiro com 100 elementos, iniciando no endereço **1000** até o endereço **1399** (ocupando assim 400 bytes)

```
1    x[0]; //aponta para o endereço 1000
2    x[1]; //aponta para o endereço 1004
3    x[2]; //aponta para o endereço 1008
4    x[3]; //aponta para o endereço 1012
```

Indexação

Portanto, a fórmula para calcular o endereço de memória de um elemento do vetor é

endereço de memória elemento = endereço base + (índice do vetor * tamanho do tipo)

endereço de x[2] = 1000 + (2 * 4)

endereço de x[2] = 1000 + 8

endereço de x[2] = 1008

Indexação

Lembre-se que dizer o endereço de x[2] é o mesmo que dizer **&x[2]**

E que o endereço de x[0], expresso por **&x[0]** é o mesmo que somente **x**

O vetor é uma variável que aponta para um endereço, quando os colchetes não são utilizados.

Indexação

A variável do vetor **sem os colchetes** é o endereço base.

Logo, **x** aponta para o endereço base do vetor (primeiro endereço).

Em cursos avançados de algoritmos, estuda-se que esse tipo especial de variável que aponta para um endereço de memória é conhecido pelo nome de apontador ou ponteiro.

Indexação em Vetores Multidimensionais

Em vetores multidimensionais, o processo é semelhante.

Lembremos que uma matriz de 1000 por 1000, possui um total de 1 milhão de elementos.

Logo:

```
1   int m[1000][1000];
2   //essa matriz ocupa 4 Megabytes de memória,
3   //ou 4 milhões de bytes
4   //1 Mega = 1 Milhão
```

Indexação em Vetores Multidimensionais

Em pseudo-código:

```
1   variaveis
2     m           : vetor[1000, 1000] de inteiro;
3   inicio
4     ...
5   m[0, 0]; //primeiro elemento
6   m[0, 1]; //segundo elemento
7   m[0, 999]; //milésimo elemento
8   m[1, 0]; //milésimo primeiro elemento
9   m[999, 999]; //último elemento
```

Algumas linguagens de pseudo-código começam a contar a partir de 1. Isso não importa muito, em pseudocódigo. Só é necessário ser coerente

Indexação em Vetores Multidimensionais

```
1    int m[1000][1000];
2    m[0][0]; //primeiro elemento
3    m[0][1]; //segundo elemento
4    m[0][999]; //milésimo elemento
5    m[1][0]; //milésimo primeiro elemento
6    m[999][999]; //último elemento
```

Indexação em Vetores Multidimensionais

Note que o acesso aos elementos do vetor é feito da dimensão mais externa, até o limite, para depois mudar o índice na dimensão mais interna.

Qualquer elemento do vetor pode ser acessado, a qualquer hora, mas o **acesso sequencial é muito comum, em diversos algoritmos**.

Indexação em Vetores Multidimensionais

Voltando ao cálculo dos endereços, temos o seguinte:

```
1   int m[1000][1000]; //4 milhões de bytes reservados
2   int y; // 4 bytes reservados
3   m[0][0]; //endereço 1000
4   m[0][1]; //endereço 1004
5   m[0][999]; //endereço 4996
6   m[1][0]; //endereço 5000
7   m[999][999]; //endereço 4.000.996
8   y; // endereço 4.001.000
```

Indexação em Vetores Multidimensionais

Suponha um vetor de dimensões *i1, i2, ... ik*.

endereço de memória = endereço base + (ik * tamanho do tipo) + (ik-1 * tamanho dimensao ik * tamanho do tipo) + (ik-2 * (tamanho dimensao ik * tamanho dimensao ik-1 * tamanho do tipo)) + (i1 * (tamanho dimensao ik * tamanho dimensao ik-1 * ... tamanho dimensao i2 * tamanho do tipo))

Não se preocupe com a fórmula anterior, porque na prática, raramente um vetor de mais de 3 dimensões é utilizado.

Exemplos:

k = 2, i1 = 10, i2 = 5, endereço base = 1000. tipo = inteiro (4 bytes).

endereço de x[9][4] = 1000 + (4 * 4) + (9 * 5 * 4)

endereço de x[9][4] = 1000 + (16) + (180)

endereço de x[9][4] = 1000 + 196

endereço de x[9][4] = 1196

Indexação em Vetores Multidimensionais

k = 2, i1 = 10, i2 = 5, endereço base = 1000. tipo = inteiro (4 bytes).

endereço de x[5][2] = 1000 + (2 * 4) + (5 * 5 * 4)

endereço de x[5][2] = 1000 + (8) + (100)

endereço de x[5][2] = 1000 + 108

endereço de x[5][2] = 1108

Indexação em Vetores Multidimensionais

k = 4, i1 = 10, i2 = 5, i3 = 4, i4 = 2, endereço base = 1000. tipo = inteiro (4 bytes).

endereço de x[9][4][3][1] = 1000 + (1 * 4) + (3 * 2 * 4) + (4 * 2 * 4 * 4) + (9 * 2 * 4 * 5 * 4)

endereço de x[9][4][3][1] = 1000 + (4) + (24) + (128) + (1440)

endereço de x[9][4][3][1] = 1000 + 1596 = 2596

Indexação em Vetores Multidimensionais

O programador só precisa saber indexar os elementos. O cálculo e a resolução dos endereços fica a cargo do compilador.

No entanto, o conhecimento dessa fórmula pode ser úti na tentativa de descobrir e entender possíveis erros.

Limites dos Índices

Na linguagem C, os índices dos vetores só podem ser números inteiros e positivos, com contagem a partir de zero e índice máximo **n - 1**, considerando **n** como o tamanho da dimensão do vetor.

```
1   int v[10];
2   v[-1]; //índice inválido
3   //acessa a variável anterior!!! Pode causar quebra
4   //no programa
5   //ou pior, causar erros silenciosamente
6   v[0]; //primeiro elemento
7   v[9]; //último elemento
```

Acessando os elementos do vetor

Normalmente, os elementos de um vetor são acessados (lidos/escritos) utilizando um comando de repetição (**while, do-while ou for**) com uma variável contadora para cada dimensão do vetor.

Exemplos - Ler um vetor com n elementos

```c
//Ler um vetor de tamanho n, no máximo 10
#include <stdio.h>

int main(){
  int i, n, v[10];
  printf("Digite o tamanho do vetor, maximo 10");
  scanf("%d", &n);
  for(i = 0; i < n; i++){
    printf("Digite o elemento do vetor\n");
    scanf("%d", &v[i]);
  }
  printf("Imprimindo vetor na ordem inversa\n");
  for(i = n - 1; i >= 0; i = i -1){
    printf("%d ", v[i]);
  }
  printf("\n"); //poderia ser puts("");
  return 0;
}
```

Exercícios com Vetores

A Lista de Exercícios está disponível no sistema Eureka.

Referências e Links

Arranjos - Wikipédia