Capítulo 7: Vetores

Waldemar Celes e Roberto Ierusalimschy

29 de Fevereiro de 2012

1 Representação de conjunto de dados

Nos exemplos anteriores, temos usado variáveis simples para armazenar valores usados por nossos programas. Em várias situações, precisamos armazenar não alguns poucos valores simples, mas um conjunto de valores. Para exemplificar, vamos retomar o exemplo do capítulo anterior em que calculamos o valor da médias das notas de uma disciplina armazenadas em um arquivo. A média de um conjunto com n valores x_i é definida como sendo:

$$m = \frac{\sum x_i}{n}$$

No exemplo que discutimos, o somatório dos valores era computado a medida que líamos os valores do arquivo. Como só estávamos interessados no cálculo da média dos valores, esta abordagem era simples e suficiente.

Vamos agora considerar que, além da média, também estamos interessados em calcular a variância do conjunto dos números. A variância v de um conjunto de valores x_i é definida como sendo:

$$v = \frac{\sum (x_i - m)^2}{n}$$

onde m representa a média dos valores. Neste caso, a estratégia de acumular a soma a medida que lemos os valores deixa de ser adequada, pois precisamos do valor da média para poder fazer o somatório para o cálculo da variância. Uma solução seria ler o arquivo para o cálculo da média e então ler novamente o arquivo para o cálculo da variância, mas esta não é uma estratégia satisfatória pois a leitura de dados em arquivos é uma operação computacionalmente cara. O problema seria agravado se o nosso programa estivesse lendo os valores do teclado: o usuário teria que entrar com a mesma seqüência de valores duas vezes, o que seria impraticável.

A solução para este problema é usarmos um mecanismo que nos permita armazenar um conjunto de valores na memória do computador. Desta forma, podemos ler os valores do arquivo (ou do teclado) e armazená-los na memória. Posteriormente, estes valores podem ser livremente processados de forma eficiente, pois já estariam na memória do computador.

Podemos armazenar um conjunto de valores na memória do computador através do uso de *vetores* (*arrays*, em inglês). O vetor é a forma mais simples de organizarmos dados na memória do computador. Com vetores, os valores são armazenados na memória do computador em seqüência, um após o outro, e podemos livremente acessar qualquer valor do conjunto. Na linguagem C, quando *declaramos um vetor* (conceito análogo ao de declaração de uma variável simples) devemos informar a *dimensão do vetor*, isto é, o número *máximo* de elementos que poderá ser armazenado no espaço de memória que é reservado para o vetor. Devemos também informar o tipo dos valores que serão armazenados no vetor (por exemplo, `int`, `float` ou `double`). Num vetor, só podemos armazenar valores de um mesmo tipo. Assim, se declaramos um vetor de `int`'s, só podemos armazenar valores inteiros; se declaramos um vetor de `float`'s, só podemos armazenar valores reais de simples precisão, e assim por diante.

Em C, a sintaxe usada para declarar um vetor é similar à sintaxe para declaração de variáveis simples, mas devemos especificar a dimensão do vetor entre colchetes logo após o nome da variável que representa o vetor. A dimensão do vetor deve ser uma constante inteira, isto é, não podemos dimensionar um vetor usando um valor armazenado numa variável. Assim, para declarar uma variável que representa um vetor de até 10 inteiros com o nome x, fazemos:

```
int x[10];
```

Esta declaração reserva um espaço de memória para armazenar 10 valores inteiros e este espaço de memória é referenciado pelo nome x. Após a declaração de um vetor, podemos escrever e ler cada valor do conjunto. A linguagem C não oferece mecanismos para processarmos todos os valores do conjunto ao mesmo tempo – só podemos acessar e processar elemento a elemento. Um elemento do vetor x é acessado escrevendo-se $x[i]$, onde i representa o *índice* do elemento. O primeiro elemento do conjunto representado pelo vetor é acessado pelo índice 0 (zero). Conseqüentemente, o último elemento de um vetor com dimensão n é acessado pelo índice $n - 1$. No caso da declaração mostrada, o último elemento do vetor é acessado pelo índice 9 (nove).

Por exemplo, após a declaração acima, podemos atribuir valores a alguns elementos do vetor como a seguir:

```
x[0] = 5;
x[1] = 11;
x[4] = 0;
x[9] = 3;
```

Este trecho de código atribui o valor 5 ao primeiro elemento do vetor, o valor 11 ao segundo, o valor 0 ao quinto e o valor 3 ao último elemento (décimo elemento do vetor). Os demais elementos do vetor permanecem com valores indefinidos. A Figura 1 ilustra a área de memória associada ao vetor em questão, com os respectivos índices e valores dos elementos.

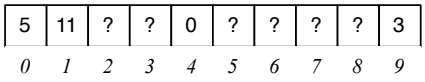

Figura 1: Vetor de inteiro de dimensão 10 com alguns elementos atribuídos.

Como dissemos, a partir da declaração de um vetor, podemos acessar livremente seus elementos. É válido declarar um vetor com uma determinada dimensão n e armazenar valores nas primeiras m posições, desde que m seja menor ou igual a n. Se tentarmos acessar elementos além da dimensão do vetor, acessaremos uma área de memória inválida, pois não está reservada para o vetor em questão. Neste caso, dizemos que estamos *invadindo* memória, isto é, acessando uma memória que não está reservada para nosso uso. Assim, se declaramos um vetor com n elementos, acessar o elemento de índice n (ou maior) é uma operação inválida – só podemos acessar os elementos com índices entre 0 e $n - 1$.

Podemos declarar vetores de diferentes tipos. O trecho de código a seguir ilustra declarações válidas de variáveis simples e variáveis vetor.

```
int a, b[20];          /* declara uma variável simples e um vetor */
float c[10];           /* declara um vetor */
double d[30], e, f[5]; /* declara dois vetores e uma variável simples */
```

Ainda, assim como podemos inicializar os valores das variáveis simples na declaração, podemos inicializar os valores dos elementos dos vetores na declaração. Os valores iniciais dos

elementos dos vetores devem ser listados entre abre e fecha chaves, separados por vírgula, como ilustrado a seguir:

```
int v[5] = {12, 5, 34, 32, 9};
```

Para acessar, um a um, todos os elementos do vetor, podemos codificar uma construção de repetição, usando uma variável inteira como índice do elemento do vetor. Por exemplo, o programa abaixo declara e inicializa um vetor de números reais e, em seguida, exibe os valores de seus elementos na tela:

```
#include <stdio.h>

int main (void)
{
   int i;
   float v[6] = {2.3, 5.4, 1.0, 7.6, 8.8, 3.9};

   for (i=0; i<6; i++) {
      printf("%f", v[i]);
   }

   return 0;
}
```

Já o programa abaixo declara o mesmo vetor e exibe na tela o somatório de seus elementos:

```
#include <stdio.h>

int main (void)
{
   int i;
   float s = 0.0;
   float v[6] = {2.3, 5.4, 1.0, 7.6, 8.8, 3.9};

   for (i=0; i<6; i++) {
      s = s + v[i];
   }

   printf("%f", s);

   return 0;
}
```

2 Cálculo da média e da variância

Agora que conhecemos um mecanismo para armazenar conjuntos de valores na memória do computador, podemos implementar o código que exibe a média e a variância de um conjunto de valores armazenadas em um arquivo. No exemplo do arquivo com notas de uma disciplina, cada linha do arquivo com nome "notas.txt" continha uma nota, como ilustrado a seguir:

7.5
8.4
9.1
4.0
5.7
4.3

Nosso objetivo é escrever um código para exibir a média da turma e o respectivo valor da variância das notas. Para tanto, podemos ler os valores do arquivo e armazená-los em um vetor e, em seguida, efetuar o cálculo da média e da variância. Como precisamos fornecer uma dimensão para o vetor, devemos assumir um número máximo de notas no arquivo. No nosso código, vamos assumir que o número de notas no arquivo não excede 50. O código abaixo ilustra uma implementação deste programa.

```
#include <stdio.h>

int main (void)
{
  int i;
  int n;            /* número de notas lidas */
  float m;          /* média dos valores */
  float v;          /* variância dos valores */
  float notas[50];  /* vetor com as notas */
  FILE *f;

  f = fopen("notas.txt","r");

  if (f==NULL){
    printf("Erro na abertura do arquivo.\n");
    return 1;
  }

  /* Lê valores do arquivo e armazena no vetor */
  n = 0;
  while (fscanf(f,"%f", &notas[n])==1) {
    n++;
  }
  fclose(f);

  /* Calcula média dos valores */
  m = 0.0;
  for (i=0; i<n; i++) {
    m = m + notas[i];
  }
  m = m / n;

  /* Calcula variância dos valores */
  v = 0.0;
  for (i=0; i<n; i++) {
    v = v + (notas[i] - m) * (notas[i] - m);
  }
  v = v / n;

  printf("\nMedia = %f e Variancia = %f\n", m,v);

  return 0;
}
```

Com este código, se existirem menos do que 50 valores no arquivo, o programa funciona perfeitamente. É verdade que reservamos um espaço de memória maior do que estaria sendo usado, mas isso não é necessariamente um problema. Por outro lado, se existirem mais do que 50 notas no arquivo, o programa não funcionará, pois tentaremos acessar elementos fora do limite do vetor. Podemos evitar que ocorra esta "invasão" de memória limitando a leitura para até 50 valores (eventuais valores adicionais não serão considerados). Para tanto, alteramos o laço que lê os valores:

```
...
/* Lê valores do arquivo e armazena no vetor */
n = 0;
while ( n<50 && fscanf(f,"%f", &notas[n])==1)) {
  n++;
}
...
```

3 Vetores passados para funções

Quando discutimos passagem de variáveis simples para funções, dissemos que os valores das variáveis simples são copiados para os parâmetros das funções. Por isso, alterar os valores dos parâmetros dentro da função não altera os valores das respectivas variáveis originais. Além de variáveis simples, também podemos passar vetores como parâmetros de funções. No caso de vetores, no entanto, a linguagem C não faz uma cópia dos elementos do vetor na chamada da função. Quando passamos um vetor como parâmetro, a função chamada recebe uma *referência* para o vetor. Isto significa que a função chamada, quando acessa os elementos, acessa as mesmas posições de memória que a função que declarou o vetor. Por esta razão, se, dentro da função, atribuirmos um valor a um elemento do vetor passado como parâmetro, este elemento também é alterado no vetor original. Portanto, podemos declarar um vetor em uma função e chamar uma outra função auxiliar para acessar e/ou modificar seus elementos.

A referência do vetor passada para a função indica apenas o início do espaço de memória a partir do qual os elementos estão armazenados. Para a função chamada, é indiferente o tamanho com o qual o vetor foi dimensionado – a função sempre recebe apenas uma referência para o início da área de memória. Portanto, é possível escrevermos uma função que receba como parâmetro um vetor de dimensão qualquer. Não precisamos particularizar a função para um vetor de determinada dimensão, mas para isso precisamos receber, também como parâmetro, o número de elementos do vetor.

Com o uso de funções auxiliares, podemos re-escrever o código para cálculo da média e da variância de forma mais organizada. Podemos criar três funções auxiliares: uma para ler os valores do arquivo e armazená-los no vetor; uma para calcular a média dos valores; e uma para calcular a variância.

A função para ler os valores e armazená-los no vetor pode receber como parâmetros o arquivo, a dimensão do vetor e a referência para o vetor. Um parâmetro que representa uma referência para um vetor é expresso através de abre e fecha colchetes após o nome da variável (que é livremente escolhido, como para qualquer outro parâmetro). Não colocamos a dimensão do vetor entre os colchetes, pois o que está sendo passado é apenas uma referência para o vetor original. Assim, um parâmetro de nome `v` representando uma referência para um vetor de `float` é expresso por `float v[ ]`. Nossa função retorna o número de notas efetivamente lidos do arquivo. Uma possível implementação desta função é mostrada a seguir:

```
int captura (FILE *f, int dim, float v[ ])
{
```

```
  int n = 0;
  while ( n < dim && (fscanf(f,"%f",&v[n])==1)) {
    n++;
  }
  return n;
}
```

Note que esta função funciona para qualquer vetor de `float`, por isso a necessidade de passar como parâmetro a dimensão do vetor original. Outra possibilidade seria assumir que a função serve apenas para vetores dimensionados com 50 elementos e não receber a dimensão como parâmetro, substituindo `dim` por 50 na condição do `while`, mas assim estaríamos limitando nossa função sem necessidade e sem vantagens consideráveis.

A função para calcular a média pode receber como parâmetros o número de valores armazenados no vetor e uma referência para o vetor. Esta função retorna a média calculada:

```
float media (int n, float v[ ])
{
  int i;
  float med = 0.0;
  for (i=0; i<n; i++) {
    med = med + v[i];
  }
  return med / n;

}
```

Note que esta função serve para calcular a média dos valores armazenados em qualquer vetor de `float`. A função para o cálculo da variância é similar, mas recebe um parâmetro adicional representando a média dos valores:

```
float variancia (int n, float v[ ], float med)
{
  int i;
  float var = 0.0;
  for (i=0; i<n; i++) {
    var = var + (v[i] - med) * (v[i] - med);
  }
  return var / n;
}
```

O programa se completa com a codificação da função *main*. Com as funções auxiliares, o código da função *main* fica mais estruturado:

```
int main (void)
{
  int n;              /* número de notas lidas */
  float m;            /* média dos valores */
  float v;            /* variância dos valores */
  float notas[50];    /* vetor com as notas */
  FILE *f;

  f = fopen("notas.txt","r");
  n = captura(f,50,notas);
  fclose(f);
```

```
  m = media(n,notas);
  v = variancia(n,notas,m);

  printf("\nMedia = %f e Variancia = %f\n", m,v);

  return 0;
}
```

4 Exemplos com vetores

Para ilustrar a manipulação de vetores com a linguagem C, esta seção apresenta alguns exemplos que operam sobre valores armazenados em vetores.

4.1 Valor máximo (ou mínimo)

Como primeiro exemplo, podemos codificar uma função que tem como valor de retorno o *valor máximo* armazenado em um dado vetor.

Para determinar o valor máximo de um conjunto de valores, podemos inicializar uma variável que representará o valor máximo (por exemplo, $vmax$) com o menor valor possível do conjunto. Por exemplo, se temos um vetor que armazena as notas em uma disciplina, sabemos que o valor mínimo é 0 (zero). Devemos então acessar cada elemento do vetor; para cada elemento, verificamos se ele é maior que o valor atribuído à variável $vmax$; se for, atribuímos o valor do elemento à variável. Fazendo este procedimento para todos os elementos do vetor garante que, ao final, a variável armazenará o maior valor do conjunto. A implementação de uma função que calcula e retorna o valor máximo de um vetor que armazena notas em uma disciplina é ilustrada abaixo.

```
float maximo (int n, floar v[ ])
{
   int i;
   float vmax = 0.0;
   for (i = 0; i < n; i++) {
      if (v[i] > vmax) {
         vmax = v[i];
      }
   }
   return vmax;
}
```

Apesar de correta, esta função pode ser escrita de forma mais genérica, pois, como está, assume-se que os elementos do vetor não podem ser negativos (se todos os elementos forem negativos, a função retornaria o valor zero, que não seria o valor máximo do conjunto). Para generalizar a função, não podemos assumir que o valor mínimo do conjunto seja zero. Uma solução simples consiste em inicializar a variável com o primeiro valor do conjunto. Consequentemente, no laço, não precisamos verificar o primeiro elemento (acessa-se os elementos de índice 1 até $n-1$):

```
float maximo (int n, floar v[ ])
{
   int i;
   float vmax = v[0];
   for (i = 1; i < n; i++) {
      if (v[i] > vmax) {
```

```
            vmax = v[i];
        }
    }
    return vmax;
}
```

Assumindo o arquivo de notas usado nos exemplos anteriores, podemos escrever uma função que lê os valores do arquivo e exibe, além da média, a maior da nota:

```
int main (void)
{
  int n;              /* número de notas lidas */
  float m;            /* média dos valores */
  float M;            /* maior nota */
  float notas[50];    /* vetor com as notas */
  FILE *f;

  f = fopen("notas.txt","r");
  n = captura(f,50,notas);
  fclose(f);

  m = media(n,notas);
  M = maximo(n,notas);

  printf("Media = %f\n", m);
  printf("Maior nota = %f\n", M);

  return 0;
}
```

Para o cálculo do *valor mínimo*, usa-se uma estratégia similar, checando se existem elementos do vetor *menores* que o valor atribuído à variável que representará o valor mínimo.

4.2 Média ponderada

O cálculo da média ponderada é útil em várias situações. A *média ponderada* consiste em calcular a média aritmética de um conjunto de valores considerando que cada valor tem um *peso* para o cálculo da média. Considerando x como sendo o conjunto de valores e w os respectivos pesos associados aos elementos de x, a média ponderada destes valores é definida como sendo:

$$m = \frac{\sum x_i w_i}{\sum w_i}$$

A implementação de uma função que efetua o cálculo da média ponderada de um conjunto de valores não apresenta maiores desafios. Devemos calcular dois somatórios, um para o numerador e outro para o denominador da fórmula. Estes dois somatórios podem ser calculados em um mesmo laço de repetição, que percorre todos os elementos dos vetores (logicamente, os dois vetores devem ter o mesmo número de elementos). Após o cálculo dos somatórios, basta calcular a razão entre eles.

Uma possível implementação desta função é mostrada a seguir. A função recebe como parâmetros os dois vetores, o vetor dos valores, x, e o vetor de pesos, w, além do número de elementos nos vetores, n.

```
float media_ponderada (int n, float x[ ], float w[ ])
{
   int i;
```

```
    float num = 0.0;  /* numerador da fórmula */
    float den = 0.0;  /* denominador da fórmula */
    for (i = 0; i < n; i++) {
       num = num + x[i]*w[i];
       den = den + w[i];
    }
    return num/den;   /* retorna a média: razão entre os somatórios */
}
```

4.3 Soma acumulada

Neste exemplo, vamos processar um vetor de entrada gerando um outro vetor como resultado. Para tanto, vamos considerar o cálculo de um vetor que represente a soma acumulada dos elementos do vetor de entrada. O problema é definido da seguinte forma: dado um conjunto de valores armazenados em um vetor, x, objetiva-se preencher um outro vetor, a, onde cada elemento a_i represente a soma dos elementos x_j, sendo j menor ou igual a i:

$$a_i = \sum_{j=0}^{i} x_j$$

A Figura 2 ilustra a obtenção do vetor com as somas acumuladas para um conjunto de valores inteiros.

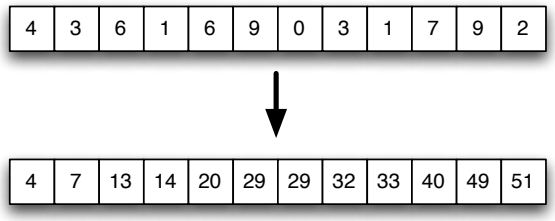

Figura 2: Cálculo da soma acumulada dos elementos de um vetor: um vetor de entrada (acima) e respectivo vetor de saída (abaixo).

Em uma primeira implementação, podemos escrever um código que traduza diretamente a fórmula para obtenção dos elementos a_i mostrada: para cada valor de a calcula-se o somatório dos elementos anteriores x_i. Uma implementação deste código é mostrada a seguir. Esta função recebe dois vetores como parâmetros. O primeiro vetor, x, tem os valores de entrada, e o segundo vetor, a, será preenchido dentro da função. Este portanto é um exemplo onde a função chamada preenche os elementos de um vetor declarado na função que chama. A função recebe também o número de elementos armazenados em x e assume que o vetor a foi declarado com uma dimensão maior ou igual ao número de elementos fornecido. Neste exemplo, vamos considerar vetores de valores inteiros.

```
void soma_acumulada (int n, int x[ ], int a[ ])
{
    int i, j;
    for (i = 0; i < n; i++) {       /* para cada elemento de a */
       a[i] = 0;
       for (j = 0; j <= i; j++) {  /* percorre os elementos de x */
          a[i] = a[i] + x[j];      /* calcula somatório */
       }
    }
```

```
}
```

Apesar de correto, este código não calcula o vetor com as somas acumuladas de forma muito eficiente, pois para calcular cada valor de a, faz-se um laço percorrendo os elementos de x. Assim, um mesmo valor x_i é acessado várias vezes. Existe uma forma mais eficiente para calcular os elementos do vetor de saída. Basta verificar que cada valor do vetor de saída pode ser obtido somando-se a soma anterior com o valor correspondente do vetor de entrada. A Figura 3 ilustra este procedimento.

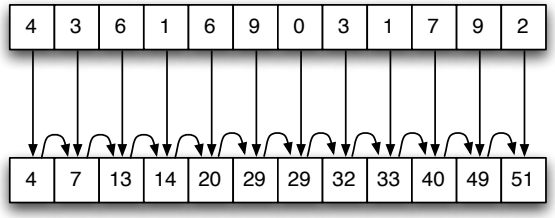

Figura 3: Procedimento para cálculo da soma acumulada.

Portanto, cada valor do vetor de saída pode ser expresso por:

$$a_i = \begin{cases} x_0 & \text{se } i = 0, \\ a_{i-1} + x_i & \text{se } i > 0. \end{cases}$$

Com isso, podemos re-escrever o código que calcula a soma acumulada, agora de forma mais eficiente. Acessamos cada elemento do vetor de entrada apenas uma vez.

```
void soma_acumulada (int n, int x[ ], int a[ ])
{
   int i;
   a[0] = x[0];
   for (i = 1; i < n; i++) {
      a[i] = a[i-1] + x[i];
   }
}
```

Um variação destas implementações é armazenar a soma acumulada no próprio vetor de entrada, isto é, os elementos do vetor original, x_i, são substituídos pelos valores que representam a soma acumulada. Neste caso, a função altera o vetor de entrada fornecido:

```
void soma_acumulada_2 (int n, int x[ ])
{
   int i;
   for (i = 1; i < n; i++) {
      x[i] = x[i-1] + x[i];
   }
}
```

5 Cálculo de histogramas

Como exemplo adicional do uso de vetores, vamos considerar o cálculo do *histograma* de um dado conjunto de valores. Para ilustrar, vamos considerar novamente o arquivo com as notas

dos alunos de uma disciplina. Queremos calcular o histograma para este conjunto de valores. Um histograma mostra a distribuição dos valores em um conjunto. No nosso caso, o histograma pode ser computado calculando quantas notas estão entre 0.0 e 1.0, quantas estão entre 1.0 e 2.0, entre 2.0 e 3.0, e assim por diante.

Um histograma mede a freqüência de ocorrência de valores em diferentes intervalos. Portanto, um histograma pode ser representado por um conjunto de valores reais, indicando a percentagem de valores em cada intervalo. A dimensão do vetor será o número de intervalos escolhidos para subdividir o domínio. No nosso exemplo, o histograma será representado por um vetor dimensionado com 10 elementos.

Podemos considerar a implementação de uma função geral, capaz de computar o histograma de qualquer conjunto de valores reais. Para que isso seja possível, além do conjunto de valores do qual queremos calcular o histograma, precisamos saber em quais intervalos dividiremos o domínio. Assumindo intervalos iguais, a subdivisão pode ser definida pelos valores mínimo, min, máximo, max, e pelo número de intervalos, ni. Assim, a "largura" de cada intervalo é dada por $\delta = (max - min)/ni$. O primeiro intervalo é definido por $[min, min + \delta)$, o segundo por $[min+\delta, min+2\delta)$, e assim por diante. O último intervalo pode ser considerado fechado nos dois extremos a fim de representar também os valores iguais ao máximo: $[min+(ni-1)\delta, max]$. No caso das notas de uma disciplina, por exemplo, o valor mínimo é 0.0 o máximo é 10.0 e o número de intervalos que queremos é 10. O último intervalo inclui as notas que variam entre 9.0 e 10.0, inclusive.

A função que calcula um histograma tem que percorrer os valores do conjunto dado e contar o número de ocorrências em cada intervalo. Para cada valor, deve-se verificar em qual intervalo ele se encontra e incrementar o valor do histograma em uma unidade. Ao final da repetição, o vetor de histograma conterá o número de ocorrências em cada intervalo. Para se obter a percentagem (freqüência), basta dividir pelo número de valores no conjunto.

Para saber o intervalo que contém um determinado valor, não precisamos testar intervalo por intervalo. Existe uma forma simples e eficiente: para cada valor v_i do conjunto, computamos o valor $(v_i - min)/\delta$. A parte inteira desta computação indica o intervalo a que pertence o valor: $0, 1, 2, ..., ni - 1$. Para o caso particular do v_i ser igual a max, o resultado será ni, que é um intervalo inválido. Neste caso, assumimos que o valor pertence ao último intervalo $(ni - 1)$.

A implementação de uma função para calcular o histograma de um conjunto de valores é mostrada abaixo. A função é responsável por preencher o vetor que representa o histograma, com as freqüências dos valores em cada intervalo.

```
void histograma (int n, float v[ ], float min, float max, int ni, float h[ ])
{
  int i, j;
  float delta = (max - min) / ni;

  /* inicializa vetor de histograma */
  for (i=0; i<ni; i++) {
    h[i] = 0;
  }
  /* calcula número de ocorrências em cada intervalo */
  for (j=0; j<n; j++) {
    i = (int) ((v[j]-min) / delta);
    if (i == ni) {
      i = ni-1;
    }
    h[i]++;
  }
  /* calcula freqüência */
  for (i=0; i<ni; i++) {
    h[i] = h[i] / n;
```

```
  }
}
```

Para testar nossa função, podemos implementar uma função *main* que captura as notas dos alunos armazenadas em um arquivo, calcula e exibe o histograma correspondente na tela. Para ler os valores do arquivo, usamos a função `captura` definida anteriormente.

```
int main (void)
{
  int i;
  int n;              /* número de notas lidas */
  float hist[10];     /* histograma */
  float notas[50];    /* vetor com as notas */
  FILE *f;

  f = fopen("notas.txt","r");
  n = captura(f,50,notas);
  fclose(f);

  histograma(n,notas,0.0,10.0,10,hist);

  printf("Histograma calculado:\n");
  for (i=0; i<n; i++) {
     printf("%f\n", hist[i]);
  }

  return 0;
}
```

6 Representação de polinômios

Um polinômio é uma função matemática definida por:

$$a(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + ... + a_gx^g$$

onde $a_0, a_1, a_2, a_3, ..., a_g$ são números reais, denominados *coeficientes do polinômio*, e g é um número inteiro que representa o *grau do polinômio*. Um polinômio é portanto representado por seu grau e seus coeficientes. Se tivermos que manipular polinômios em um programa de computador, podemos usar vetores para representá-los. Um polinômio de grau g pode ser representado por um vetor v com $g + 1$ elementos, cujos valores representam os coeficientes do polinômio: $v[i] = a_i$. Assim, o polinômio $2x^3 + 8x + 5$ pode ser representado por um vetor com quatro elementos: $v[0] = 5$, $v[1] = 8$, $v[2] = 0$ e $v[3] = 2$.

Assumindo esta representação de polinômios com vetores, podemos considerar a implementação de diversas funções que operam sobre polinômios. Estas funções podem ser agrupadas em um arquivo e servir como uma biblioteca para manipular polinômios. Nas seções subseqüentes, discutimos a implementação de algumas destas funções.

6.1 Avaliação de polinômios

Avaliar um polinômio significa avaliar o valor numérico do polinômio, $y = a(x)$, para um determinado x. O valor numérico de um polinômio pode ser expresso matematicamente por:

$$y = \sum_{i=0}^{g} a_i x^i$$

Portanto, a codificação para a avaliação de um polinômio se traduz no cálculo de um somatório. A função para avaliar um polinômio deve receber como entrada o polinômio e o valor de x, tendo como retorno o valor avaliado. Nos nossos exemplos, vamos assumir que os parâmetros que representam o polinômio são o número de coeficientes, n, e os valores destes coeficientes armazenados em um vetor, v. Portanto, estamos assumindo que o polinômio tem grau $n-1$. Uma possível implementação desta função é mostrada a seguir:

```
float avalia (int n, float v[ ], float x)
{
  int i;
  float y = 0.0;
  for (i=0; i<n; i++) {
    y = y + v[i] * pow(x,i);
  }
  return y;
}
```

6.2 Igualdade de polinômios

Dois polinômios de graus g, $a(x) = a_0 + a_1x + a_2x^2 + ... + a_gx^g$ e $b(x) = b_0 + b_1x + b_2x^2 + ... + b_gx^g$, são iguais se os valores de seus coeficientes forem iguais, isto é, a_i tem que ser igual a b_i para qualquer $i : 0 \leq i \leq g$.

Uma função para testar se dois polinômios de mesmo grau são iguais pode receber como parâmetros o número de coeficientes dos polinômios e os vetores com seus valores. Uma possível implementação desta função é mostrada a seguir. A função retorna 0 se os polinômios forem diferentes e 1 se eles forem iguais.

```
int igualdade (int n, float a[ ], float b[ ])
{
  int i;
  for (i=0; i<n; i++) {
    if (a[i] != b[i]) {
      return 0;
    }
  }
  return 1;
}
```

6.3 Soma de polinômios

A soma de dois polinômios de graus g, $a(x) = a_0 + a_1x + a_2x^2 + ... + a_gx^g$ e $b(x) = b_0 + b_1x + b_2x^2 + ... + b_gx^g$, é dada por $c(x) = (a_0 + b_0) + (a_1 + b_1)x + (a_2 + b_2)x^2 + ... + (a_g + b_g)x^g$. Esta operação pode ser matematicamente expressa por:

$$c(x) = a(x) + b(x) = \sum_{i=0}^{g} c_i x^i, \text{ onde } c_i = (a_i + b_i)$$

A implementação de uma função que efetua esta operação é simples – basta codificar a soma de dois vetores. A função deve receber como parâmetros o número de coeficientes dos polinômios e três vetores que armazenam os valores destes coeficientes: dois vetores representando os polinômios de entrada e um representando o polinômio de saída. Na chamada desta função, o programador deve garantir que o vetor que armazenará os coeficientes da soma é dimensionado com número de elementos suficiente. Uma implementação desta função é mostrada a seguir:

```
void soma (int n, float a[ ], float b[ ], float c[ ])
{
  int i;
  for (i=0; i<n; i++) {
    c[i] = a[i] + b[i];
  }
}
```

6.4 Produto de polinômios

Um exemplo mais elaborado consiste em codificar uma função que efetua o produto entre dois polinômios. O produto de dois polinômios, $a(x)$ e $b(x)$, com grau g, é dado por:

$$c(x) = a(x)b(x) = c_0 + c_1x + c_2x^2 + ... + c_{2g}x^{2g}$$

onde:

$$c_k = a_0b_k + a_1b_{k-1} + a_2b_{k-2} + ... + a_{k-1}b_1 + a_kb_0$$

ou, de forma mais concisa, por:

$$c_k = \sum_{i=0}^{k} a_ib_{k-i}$$

sendo que assume-se $a_i = 0$ e $b_i = 0$ para $i > g$.

Podemos então codificar uma função que efetua o produto entre dois polinômios. Esta função pode receber como parâmetros o número de coeficientes dos polinômios de entrada e três vetores de coeficientes: dois de entrada e um de saída. Novamente, o programador é responsável por passar para a função um vetor com dimensão suficiente para armazenar os coeficientes do produto. Como o grau do polinômio resultante é $2g$ e sabemos que n, o número de coeficientes dos polinômios de entrada, é igual a $g + 1$, o número de coeficientes do vetor de saída é $2g + 1 = 2(n - 1) + 1 = 2n - 1$. Uma possível implementação desta função é mostrada a seguir. Note que na avaliação de cada termo c_k acessamos os valores a_i e $b_k - i$. Para estes acessos serem válidos, devemos garantir que $i < n$ e $k - 1 < n$. Quando estamos fora desta condição, o termo assume valor zero e não contribui para o somatório.

```
void produto (int n, float a[ ], float b[ ], float c[ ])
{
  int i, k;
  int m = 2*n - 1; /* número de coeficientes de saída */

  for (k=0; k<m; k++) {
    c[k] = 0.0;
    for (i=0; i<=k; i++) {
      if (i<n && k-i<n) {
        c[k] = c[k] + a[i]*b[k-i];
      }
    }
  }
}
```

6.5 Derivada de polinômio

Também podemos codificar uma função que gera a derivada de um polinômio. Se o polinômio de entrada tem grau g, sua derivada é um polinômio de grau $g-1$. Se o polinômio de entrada é dado por:

$$a(x) = a_0 + a_1 x + a_2 x^2 + ... + a_g x^g$$

sua derivada é expressa por:

$$d(x) = a_1 + 2a_2 x + 3a_3 x^2 + ... + g a_g x^{g-1}$$

Os termos da derivada podem então ser expressos por:

$$d_i = (i+1)a_{i+1}, \;\; 0 \leq i < g$$

A função que implementa o cálculo da derivada pode então receber como parâmetros o número de coeficientes do polinômio de entrada, n, e dois vetores de coeficientes: um de entrada e um de saída. O programador é responsável por passar para esta função um vetor de saída dimensionado com pelo menos $n-1$ elementos. Uma possível implementação desta função é mostrada a seguir:

```
void derivada (int n, float a[ ], float d[ ])
{
  int i;
  for (i=0; i<n-1; i++) {
    d[i] = (i+1)*a[i+1];
  }
}
```

Fica como exercício a implementação de funções de testes para estas funções. Para os testes, é útil a implementação de uma função que exibe na tela os coeficientes de um dado polinômio.

Exercícios

1. Escreva uma função que retorne o valor mínimo armazenado em um vetor. A função deve ter o seguinte cabeçalho: `float mínimo (int n, float v[ ])`. Escreva um programa para testar sua função.

2. A média harmônica, H_n, de um conjunto de valores é dada por:

$$\frac{1}{H_n} = \sum_{i=0}^{n-1} \frac{1}{v_i}$$

   Escreva uma função para calcular e retornar a média harmônica de um conjunto de valores. A função deve ter o seguinte cabeçalho: `float harmonica (int n, float v[ ])`. Note que o valor retornado deve ser H_n e não $1/H_n$. Escreva um programa para testar sua função.

3. A média geométrica, G_n, de um conjunto de valores é dada por:

$$G_n = \sqrt[n]{\prod_{i=0}^{n-1} v_i}$$

   Escreva uma função para calcular e retornar a média geométrica de um conjunto de valores. A função deve ter o seguinte cabeçalho: `float geometrica (int n, float v[ ])`. Escreva um programa para testar sua função.

4. Considere *histogramas* como sendo o número de ocorrências de valores em diferentes intervalos. Considere ainda um experimento laboratorial onde foram colhidos n medições, todas elas maiores ou iguais a 0 e menores que 1. Escreva uma função para preencher um vetor com 10 elementos que represente o *histograma* destas medidas. O primeiro elemento do vetor deve armazenar o número de medidas maiores ou iguais a 0 e menores que 0.1, o segundo elemento deve armazenar o número de medidas maiores ou iguais a 0.1 e menores que 0.2, e assim por diante.

   A função deve receber o vetor, v, com as n medidas do experimento e deve preencher o vetor h que, sabe-se, tem dimensão igual a 10. Por exemplo, se for passado como entrada o vetor:

$$v = \{0.11, 0.2, 0.03, 0.56, 0.323, 0.345, 0.234, 0.56, 0.6546, 0.123, 0.123, 0.999\}$$

   a função deve preencher o vetor h como:

$$h = \{1, 3, 2, 2, 0, 2, 1, 0, 0, 1\}$$

   A função deve seguir o cabeçalho a seguir:

```
void histograma (int n, float v[ ], int h[ ])
```

5. Re-escreva as funções para calcular a soma e o produto de polinômios representados por vetores para considerar que os dois polinômios de entrada possam ter graus diferentes. Os coeficientes não existentes no polinômio de menor grau têm valores nulos, mas não são representados no vetor.