Capítulo 4
Vectores

"Existem duas maneiras de construir um projecto de software: uma maneira é fazê-lo tão simples que obviamente não existem deficiências; a outra é fazê-lo tão complicado que não existem deficiências óbvias"

- C. A. R. Hoare -

Sumário:

4.1- Conceitos

O **vector** é a estrutura de dados mais simples para armazenar informação no computador. Nessa estrutura de dados, os elementos são todos do mesmo tipo, e estão armazenados em organização sequencial.

Na organização sequencial, os elementos estão armazenados em posições consecutivas de memória, ou seja, um a seguir ao outro, permitindo desse modo o acesso directo a qualquer elemento do vector.

O acesso a qualquer elemento do vector é feito por uma variável indexada constituída pelo nome do vector e um índice entre colchetes.

Um índice é uma variável do tipo elementar, inteira ou carácter, que faz referência a posição relativa que cada elemento do vector.

Na linguagem C, a sintaxe utilizada para declarar um vector é análoga à sintaxe utilizada para declarar as variáveis do tipo elementar, com a particularidade de termos de especificar, logo apôs o seu nome, à dimensão do vector entre colchetes.

```
tipo  nome [dimensão];
```

Apôs a declaração de um vector, vamos escrever alguns programas para a sua manipulação. Como a linguagem C, não possui uma instrução para ler todos os elementos de um vector, só podemos aceder e processar elemento por elemento.

Seja v um vector com 10 elementos do tipo inteiro. Então a declaração desse vector deve possuir a seguinte sintaxe:

```
int  v[M];
```

onde M é uma constante simbólica. Os elementos desse vector, são as variáveis indexadas v[0], v[1], ..., v[9], que estão armazenadas em posições contínuas de memória. Em termos gráficos:

0	1	2	3	4	5	6	7	8	9
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

Facilmente se constacta que v[0] é o primeiro elemento, v[1] o segundo, v[2] o terceiro, …, v[9] o décimo elemento.

Na linguagem C, o primeiro elemento está armazenado na posição zero, enquanto o último na posição M-1, sendo M a dimensão do vector.

Quando se declara um vector, o compilador C, disponibiliza através do sistema operativo, um conjunto de endereços de memória para alocar os seus elementos, mas deixa nesses endereços lixo.

Mas quando se insere elementos num vector esses elementos devem ser inseridos em posições contínuas, não sendo permitido “buracos” entre essa informação válida.

Seja a um vector de números reais de dimensão M. Suponhamos que tenhamos preenchido os n primeiros elementos desse vector, para $0 \leq n \leq M-1$. Como distinguir a informação válida do lixo residual?

A estratégia mais utilizada, consiste em declarar uma variável do tipo inteiro que dar-nos-á em cada instante o número de elementos inseridos no vector, que denominaremos por nElem. Por exemplo, na figura a seguir, os elementos que estão no intervalo {0,…,6} possuem informação válida, enquanto que os elementos que estão no intervalo {7,…,M-1} possuem lixo residual.

0	1	2	3	4	5	6	7	8	9	…	M-1
5	12	17	8	7	11	12	?	?	?	…	?

↑
nElem

Não é demais salientar que todos os elementos do vector cuja informação é válida, estão nas posições cujo índice está no intervalo {0, .. , nElem-1}.

4.2- Passagem de Parâmetros

Por defeito, na linguagem C, os vectores são passados por referência. Nesta forma de comunicação, os parâmetros da função (parâmetros formais) e os parâmetros da chamada (parâmetros reais) ocupam os mesmos endereços de memória. Logo, qualquer alteração feita ao conteúdo do vector no interior da função é reflectido no seu exterior.

Na linguagem C, os parâmetros de um vector, podem ser declarados na notação vectorial ou na notação por ponteiro.

Nestas notas, utilizaremos a notação vectorial, e nessa notação, no parâmetro formal, um vector é declarado pelo seu nome seguida de abertura e encerramento de colchetes, enquanto que no parâmetro real, um vector é declarado pelo seu nome.

4.3- Operações Iterativas com Vectores

Para desenvolvermos as operações com vectores, vamos declarar a estrutura de dados onde essas operações possam ser realizadas. Utilizaremos um vector vet com M elementos do tipo inteiro. Associado a esse vector, temos uma variável do tipo inteira nElem, que dar-nos-á, em qualquer instante o número de elementos inseridos.

```
#define  M   50
  …
int vet[M];        /* vector com dimensão M */
 int nElem;        /* número de elementos inseridos no vector  */
```

É importante frisar que o número de elementos inseridos no vector, satisfaz a seguinte relação:

$$0 \leq nElem \leq M$$

4.3.1- Imprimir os Elementos de um Vector

A operação para imprimir os elementos de um vector, consiste em percorrer todos os seus elementos, e por cada elemento mostrar na tela o seu conteúdo.

```
void imprimir (int vet[ ] , int nElem )
 {
    int i;
    for ( i = 0 ; i < nElem ; i++)  printf ( " %d% \n" , vet[i] );
 }
```

Observe que nesta função, só mostramos na tela a informação valida, ou seja, a informação, cujo índice está contido no intervalo {0,…, nElem-1}

4.3.2- Inserir um Elemento numa Determinada Posição

A operação para inserir um elemento num vector, dada uma determinada posição, consiste em deslocar para à direita, todos os elementos que vão desde a posição do último elemento inserido até a posição de inserção. Inserir em seguida o elemento na posição de inserção e actualizar o número de elementos inseridos.

Mas, antes de efectuar essa operação é necessário verificar se o vector não está cheio, e se a posição (índice) de inserção está contida no intervalo {0,..,nElem}. Se essas condições forem falsas, devemos emitir as correspondentes mensagens de erro.

```
void  inserirNaPosicao ( int vet[ ], int pos , int *nElem )
{
  int i;
  if (estaCheio(nElem))
      printf ("\n Erro: vector cheio");
   else if (pos >= 0  || pos <= nElem )
            {
              for ( i =  nElem-1 ; i >= pos ; i--)  vet[i+1] = vet[i];
                  vet[i] = x;
               nElem++;
            }
          else
              printf (" \n Erro: Insercao não permitida- fora do intervalo ");
  }
```

4.3.3- Remover um Elemento de uma Determinada Posição

A operação para remover um elemento num vector, dada uma determinada posição, consiste em armazenar o conteúdo do elemento que ocupa essa posição numa variável auxiliar; deslocar para à esquerda todos os elementos que vão desde essa posição até a posição do último elemento inserido, e por fim, actualizar o índice que faz referência ao número de elementos inseridos.

Mais antes de efectuar essa operação é necessário verificar se o vector não está vazio, e se o índice de remoção está contida no intervalo {0,..,nElem-1}. Se essas condições forem falsas, devemos emitir as correspondentes mensagens de erro.

```
void RemoverDaPosicao (int vet[ ], int *nElem, int pos, int *x)
 {
   int i;
   if ( estaVazio (nElem) )
      printf ("\n Erro: Vector Vazio ");
   else if  ( pos >=0 || pos <= nElem – 1 )
            {
              *x = vet[ind];
              for ( i = pos ; i < nElem ; i++ ) vet[i] = vet[i+1];
                  nElem--;
             }
          else
            printf ("\n Erro: Remocao não permitida- Fora do intervalo");
 }
```

4.3.4- Determinar o Elemento máximo de um Vector

A operação para determinar o elemento com o maior valor num vector, consiste em supor que o elemento que encontra-se na primeira posição é o maior. Armazenar o conteúdo desse elemento numa variável auxiliar. Comparar em seguida, o conteúdo dessa variável com os restantes elementos. Se encontrarmos um elemento cujo conteúdo for maior do que o valor dessa variável, o conteúdo desse elemento deverá ser armazenado nessa variável, e o processo de comparação continua. Quando não for possível realizar mais comparações, o conteúdo dessa variável representa o elemento com o maior conteúdo no vector.

```
int  maximoDoVector  (float vet[ ], int n)
 {
   int  i , maior = vet[0];
   for (i =1 ; i <  nElem ; i++)
      if (maior < vet[i])  maior = vet[i];
    return maior;
 }
```

Torna-se evidente que essa operação só pode ser realizada se o vector possuir pelo menos um elemento.

4.4- Operações Recursivas com Vectores

Por definição um vector pode ser definido por:

a) Um conjunto unitário; ou por
b) Um conjunto composto pelo primeiro elemento seguido de um subvector que começa do segundo elemento até ao último.

Com base nesta definição recursiva, vamos implementar algumas operações com a estratégia de decrementar para conquistar.

4.4.1- Imprimir os Elementos de um Vector

Seja T = (t_0, t_1, …, $t_{nElem-1}$) um conjunto de números inteiros armazenados num vector, onde nElem é um índice que dá-nos o número de elementos inseridos nesse vector. A resolução deste problema consiste em separar o conjunto T em dois subconjuntos de tamanhos diferentes: T1= (t_0) e T2= (t_1,t_2, …, $t_{nElem-1}$). A solução para o subconjunto T1 é trivial. Como o subconjunto T2 possui nElem-2 elementos, a sua solução não é imediata. Então, procede-se a sua decomposição em dois subconjuntos T'1 = (t_0) e T'2= (t_1,t_2, …, $t_{nElem-1}$). Como este processo de decomposição gera um conjunto de menor dimensão, então ele é finito, e termina quando o índice de recursão chegar a posição do último elemento inserido.

```
void imprimirRec (int vet[ ] , int nElem , int i )
 {
   if ( i < nElem )
      {
        printf (" % \n " , vet[i]);
        imprimir (vet , n , i + 1);
      }
 }
```

É oportuno salientar, que esta função só irá mostrar os elementos cujo índice está contido no intervalo {0 , .. , nElem-1} , se essa função for invocada com a seguinte sintaxe:

```
imprimirRec (vet, nElem, 0);
```

Esta função mostra que o término da recursão, acontece quando à variável responsável pelo passo da recursão for igual a nElem. Mas, isso só é possível, porque o passo da recursão é feito no sentido crescente, ou seja, de 0 para nElem.

Contudo, na linguagem C, o primeiro elemento ocupa a posição de índice zero. Logo, podemos aproveitar essa propriedade, para utilizar essa posição como o término da recursão. Para esse caso, o passo da recursão deve ser feito no decrescente.

Está estratégia, para além de compactar o código da função, produz um ganho substancial de performance no algoritmo, porque ele reduz o número de parâmetros.

```
void imprimirRec (int vet[ ] , int nElem)
 {
   if (nElem  > = 1)
      {
        printf ( " % \n" , vet[i -1]);
        imprimir (vet, nElem-1);
      }
 }
```

4.4.2- Somar os Elementos de um Vector

Seja T = (t_0, t_1, …, $t_{nElem-1}$) um conjunto de números inteiros armazenados num vector, onde nElem é um índice que faz referencia ao número de elementos inseridos. Por definição, a soma dos elementos de um vector é definida por:

$$S = t(0) + t(1) + t(3) + \ldots + t(nElem-1)$$

A resolução deste problema consiste em separar o conjunto T em dois subconjuntos de tamanhos diferentes: T1= (t_0) e T2= (t_1,t_2,t_3,…, $t_{nElem-1}$). A solução para o subconjunto T1 é trivial, logo, S = t(0), enquanto a solução para o subconjunto T2 não é imediata, ela passa pela decomposição desse subconjunto em dois subconjuntos T'1 = (t_1) e T'2= (t_2,t_3,…, $t_{nElem-1}$). Observe que este processo de decomposição apresenta a seguinte propriedade: O subconjunto à esquerda é unitário e a dimensão do subconjunto à direita é menor do que a dimensão do subconjunto original. Essa propriedade permite-nos concluir que este processo gera estruturas de dados da mesma classe e é finito. Então, o processo termina, quando gerarmos um subconjunto à esquerda unitário e um subconjunto à direita vazio, e a solução do problema original consiste na soma das soluções dos subproblemas gerados pelo processo de decomposição.

```
int somaVectorRec (int vet[ ] , int nElem , int i)
{
   if (i > nElem-1) return 0;
    return vet[i]  + somaVectorRec (vet , nElem , i + 1 );
}
```

Mais uma vez, salientamos, que podemos compactar de código desta função e melhorar o seu desempenho, se passo da recursão for no sentido decrescente

```
int somaVectorRec (int vet[ ] , int nElem )
{
   if (nElem == 1) return v[0];
    return vet[nElem-1]  + somaVectorRec (vet , nElem-1);
}
```

Vamos simular em seguida, esta versão. Seja V um vector constituído pelos seguintes elementos {5,6,9,4}. Na figura a seguir, mostramos a simulação dessa função:

```
somaVectorRec (a,v)
        4 + somaVectorRec (a, v)
            9 + somaVectorRec (a, v)
                6  + somaVectorial (a, v)
                                5
                 6 + 5 = 11
              9 + 11 = 20
          4 + 20 = 24
    24
```

cuja chamada (início da recursão), pode ser feita pelo seguinte segmento de código:

```
if ( ( nElem = tamanhoVector() ) == 0 )
   printf ("\n Erro: Vector vazio");
else {
        total = somaVectorRec (v, nElem);
        printf ("\n Numero de elementos do vector = %d ", total);
       }
```

4.4.3- Determinar o Elemento máximo de um vector

Seja T = (t_0, t_1, …, $t_{nElem-1}$) um conjunto de números inteiros armazenados num vector, onde nElem é um índice que faz referencia ao número de elementos inseridos. A resolução deste problema, consiste em separar o conjunto V em dois subconjuntos de tamanhos diferentes: T1 = (t_0,t_1,t_2,…,$t_{nElem-2}$) e T2 = ($t_{nElem-1}$). Então o elemento máximo do conjunto T é determinado pela comparação entre o elemento máximo do subconjunto T1 com o valor $t_{nElem-1}$. Como a solução do subconjunto T1 não é trivial, procede-se à sua decomposição em dois subconjuntos T'1 = (t_0,t_1,…,t_{n-2}) e T'2 = (t_{n-1}). Agora o elemento máximo é determinado pela comparação do elemento máximo do subconjunto V'1 com o valor $t_{nElem-1}$. Observa-se que este processo de decomposição, apresenta uma propriedade análoga aos problemas anteriores, mas no sentido inverso. Com

base nessa propriedade podemos concluir que o processo gera estruturas de dados da mesma classe, é finito, e termina quando gerarmos um subconjunto à esquerda vazio e um subconjunto à direita unitário. A solução do problema original consiste na comparação dos elementos máximos dos subproblemas gerados por essa decomposição.

```
int MaximoDoVectorRec (int vet[ ] , int nElem)
{
   int max;
   if ( nElem == 1 ) return vet[0];
   max = MaximoDoVectorRec (vet , nElem-1)
    if  (max > vet[nElem-1])  return max;
    return vet[nElem-1];
 }
```

Para consolidação os seus conhecimentos, solicitamos que faça uma versão recursiva equivalente, com o passo da recursão no sentido crescente.

4.5- Recursão Caudal

Os procedimentos que iremos desenvolver a seguir, têm por finalidade mostrar na tela todos os elementos de um vector. Mas antes de continuar a estudar a este assunto, faça uma simulação de cada versão, para entender o que elas realmente fazem.

```
void imprimir1 ( int vet[ ] , int nElem  )
{
   if ( nElem >= 1 )
     {
        printf ( " %d \n ", vet[nElem-1] );
        imprimir ( vet , nElem-1);
      }
}
```

Nesta função, os elementos são impressos a medida que o processo recursivo avança, e, como como consequência, os elementos do vector são impressos na ordem que estão armazenados. Mas na função:

```
void imprimir2 ( int vet[ ], int nElem )
{
   if ( nElem >= 1 )
     {
        imprimir ( vet , nElem-1 );
        printf ( " %d \n ", vet[nElem-1] );
```

```
    }
}
```

a chamada recursiva é executada antes da operação de impressão. Como consequência, os elementos do vector são armazenados numa pilha, e quando o processo recursivo terminar, esses elementos serão mostrados na tela, a medida que a pilha for desempilhada. Como esse dispositivo possui a propriedade do último a entrar ser o primeiro a sair, o processo de impressão dar-se-á no sentido inverso, ou seja, do último elemento para o primeiro.

4.6- Eliminação da Recursão

Em termos gerais, dada a função iterativa:

```
void  p (  int vet[ ] , int nElem )
 {
    int i ;
    for ( i = 0 ; i  < nElem ; i = i + 1)
          /*   resolva o problema    */
 }
```

onde vet é um vector, e nElem o número de elementos inseridos nesse vector. A equivalente função recursiva pode ser obtida se utilizarmos o seguinte esquema:

```
void  pRec (  int vet[ ], int nElem , int i )
{
    if ( i == nElem )
       /*   resolva o problema    */
    pRec ( vet , nElem , i + 1);
 }
```

4.6.1- Calcular o número de Elementos Inseridos num vector

Esta operação consiste em percorrer todos os elementos do vector e adicionar uma unidade em cada elemento percorrido.

```
int   totalElementos ( int vet[ ] , int nElem )
 {
    int total = 0, i;
    for (i =  0 ; i <= nElem , i++)  total += 1;
    return total;
 }
```

Utilizando o esquema anterior, obtemos a seguinte função recursiva:

```
int   totalElementosRec ( int vet[ ] , int nElem , int i )
```

```
{
   If ( i == nElem ) return  0;
   return 1 + totalElementosRec  ( vet , nElem , i +1);
}
```

4.7- Recomendações Bibliográficas

Para o leitor aprofundar os seus conhecimentos sobre vectores e recursão, recomendamos os seguintes livros:

Levitin A.;- *Introduction to the Design and Analysis of Algorithms,* 3rd Edition, Pearson, 2011.

K. N. King. - *C Programming – A Modern Approach*, W. W. Norton & Company, Inc., 2nd edition, 2008.

Roberts E.S.;- *Programming Abstractions in C: a Second Couse in Computer Science,* Addison-Weles, 1998.

Feofiloff P.;- *Algoritmos em Linguagem C*, Editora Campos, Brasil, 2009.

Celes W., Cerqueira R., Rangel J. L., - *Introdução a Estruturas de Dados com Técnicas de Programação em C*, 2ª Edição, Editora Elsevier, Brasil, 2016

4.8- Exercícios Propostos

4.8.1-Desenvolva uma função recursiva que recebe como parâmetro um vector com elementos do tipo inteiro e o número de elementos inseridos. Retornar o produto de todos os números positivos.

4.8.2-Desenvolva uma função recursiva que recebe um vector com elementos do tipo real e o número de elementos inseridos. Devolva esse vector com os elementos na ordem inversa, ou seja, troque o primeiro com o último, o segundo com o penúltimo e assim sucessivamente.

4.8.3-Desenvolva uma função recursiva que parâmetros recebe um vector com elementos do tipo inteiro, e o total de elementos inseridos. Verifique se nesse vector os elementos de índice par contêm um conteúdo impár e vice-versa. A sua função deve retornar 1 se essa condição for verdadeira e 0 no caso contrário.

4.8.4-Desenvolva uma função recursiva que recebe um vector com elementos do tipo real e número de elementos inseridos. Devolva o elemento máximo e o elemento mínimo.

4.8.5-Desenvolva uma função recursiva que entre outros parâmetros recebe um vector com elementos do tipo real, uma determinada posição k, e o número de elementos inseridos. Remova o elemento que ocupa essa posição.

4.8.6-Desenvolva uma função recursiva que entre outros parâmetros recebe um vector com elementos do tipo real, uma determinada posição k, um número real x, e o número de elementos inseridos. Insira o número x nessa posição.

4.8.7-Desenvolva uma função recursiva que recebe dois vectores com elementos do tipo real, e com o mesmo número de elementos inseridos. Verifique se esses vectores são iguais, e retorna 1 se todos os elementos forem iguais e zero no caso contrário.

4.8.8-Desenvolva uma função recursiva que recebe um vector com elementos do tipo inteiro, um número inteiro x, e o número elementos inseridos. Retorne o número de vezes que esse número ocorre no vector.

4.8.9-Desenvolva uma função recursiva que recebe um vector com elementos do tipo real, um elemento x, e o número de elementos inseridos. Verifique se esse elemento encontra-se no vector e retorne a sua posição relativa, no caso afirmativo, e -1 no caso contrário.

4.8.10-Desenvolva uma função recursiva que recebe dois vectores com elementos do tipo inteiro, e o número de elementos inseridos em cada vector. Suponhamos que cada elemento armazene um e apenas um algarismo. Verifique se os vectores são inversos, ou seja, se o primeiro elemento do primeiro vector, possui um conteúdo é igual ao último do segundo vector; se o segundo elemento do primeiro vector, possui um conteúdo igual ao penúltimo do segundo vector; e por aí em diante. A sua função deve retornar o valor 1 se essa propriedade for verdadeira e o valor 0 no caso contrário.

4.8.11-Desenvolva uma função recursiva que recebe dois vectores com elementos do tipo inteiro e os números de elementos inseridos nesses vectores. Devolva a junção desses vectores. Por definição a junção do vector A com o vector B é um vector C formado pelos elementos do vector A seguidos dos elementos do vector B. Essa função também deve retornar o total de elementos inseridos

4.8.12-Desenvolva uma função recursiva que recebe um vector com elementos do tipo real, e o número de elementos inseridos. Verifique se esse vector está ordenado.

4.8.13-Desenvolva uma função recursiva que recebe um vector com elementos do tipo inteiro, um número inteiro x, e o número de elementos inseridos. Devolva dois vectores, um com os elementos que estão antes da posição que faz referencia ao elemento x, e outro com as posições que estão desse elemento. O elemento x deve ser inserido no segundo vector.

4.8.14-Um número é chamado de capicua se a sequência de dígitos do número lidos da esquerda para a direita for igual a sequência de dígitos lidos da direita para a esquerda. Por exemplo: os seguintes números são capicuas: 123454321, 54445, 789987, 121. Desenvolva uma função recursiva que recebe um vector com elementos do tipo inteiro e o número de elementos inseridos. Cada

elemento desse vector contêm um e apenas um algarismo. Verificar se esse número é do tipo capicua.

4.8.15-Desenvolva uma função recursiva que recebe dois vectores com elementos do tipo inteiro, e número de elementos inseridos em cada vector. Suponha que cada elemento desses vectores, armazena um e apenas um algarismo. Devolva num terceiro vector a adição desses números. Por exemplo:

4.8.16-Desenvolva uma função recursiva que recebe um vector com elementos do tipo inteiro, e o número de elementos inseridos. Devolva num segundo vector a soma acumulada dos seus elementos. Por definição, seja X um vector com n elementos. A soma acumulada é um vector S, com o mesmo número de elementos, tais que cada elemento s_i é representado pela soma de todos os elementos x_j para $0 \leq j \leq i$, ou seja:

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

Por exemplo, para o vector X = { 4, 3, 6, 1, 6, 9, 0, 3, 1, 7, 9, 2 } a soma acumulada é o vector S = {4, 7, 13, 14, 20, 29, 29, 32, 33, 40, 49, 51}

4.8.17-Desenvolva uma função recursiva que recebe dois vectores com elementos do tipo inteiro, e o número de elementos inseridos em cada vector. Devolva num terceiro vector a intercalação desses vectores. Para além disso, retorne o número de elementos inseridos nesse vector. Por exemplo, se v1 = {4, 8, 1, 9} e v2 = {2, 5, 7, 3} a função deverá devolver o vector v3 = {4, 2, 8, 5, 1, 7, 9, 3}.

4.8.18-Desenvolva uma função que recebe um vector com elementos do tipo inteiro entre 0 a 149, e o número de elementos inseridos. Devolva um vector, com os seguintes totais: no primeiro elemento, o total de ocorrências entre 0 a 4, no segundo elemento, o total de ocorrências entre 5 a 9, . . ., e no último elemento o total de ocorrências entre145 a 149.