7

Vectores

" Ciência são factos; assim como casas são feitas de pedras, a ciênçia é feita de factos, mas uma pilha de pedras não é uma casa e uma coleção de factos não é necessariamente ciênçia"

- Henri Poencaré -

7.1- Motivação

Nos capítulos anteriores, introduzimos os conceitos de variável e tipo elementar de dados. Agora, iremos estudar um novo conceito, o tipo estruturado de dados. Antes porêm, veremos um exemplo ilustrativo que nos mostrará a importância dessa estrutura na programação.

Dada uma sequência de n números inteiros positivos. Desenvolva um programa para imprimi-la na ordem inversa.

Para n= 3, a solução é trivial, ela consiste em:

```
scanf(" %d %d %d ",&a,&b,&c);
printf(" %d %d %d ",c,b,a);
```

Mas, esta solução só é válida para esse valor. Se n for igual a dois ou quatro, essa estratégia não funciona. É evidente, que poderemos desenvolver um programa génerico, que leia um valor n, e em função desse valor, utilize o comando de decisão, para ler uma sequencia de n números e imprimi-los na ordem inversa. Essa abordagem é descrita pelo seguinte segmento de código:

```
if (n == 1)
  {
    scanf(" %d ",&a);
    printf(" %d ",a);
  }
 else if (n== 2)
         {
           scanf(" %d %d " ,&a,&b);
           printf(" %d %d ",b,a);
         }
       else if (n== 3)
              {
                scanf("  %d %d %d ",&a,&b,&c);
                printf(" %d %d %d ",c,b,a);
              }
            else if (n == 4)
                   {
                     scanf(" %d %d %d %d ",&a,&b,&c,&d);
                     printf(" %d %d %d %d ",d,c,b,a);
                   }
                 else
                     printf(" Erro: Mais do que 4 elementos ");
```

Facilmente se constacta, que necessitamos de quatro variáveis e um programa com pelo menos dezaseis linhas para resolver o problema. Como resolveriamos esse problema se n fosse igual a 100? Com essa abordagem, necessitaríamos de 100 variáveis e um programa com pelo menos quatrocentas linhas. Em programação isto não é aceitável, esta estratégia é um absurdo.

As linguagens de programação de alto nível, possuem recursos que permitem associar às n variáveis há um conjunto.

Esse conjunto, denominado por **tipo estruturado de dados** é uma colecção de variás elementos do mesmo tipo. Cada elemento pode ser do tipo elementar ou do tipo estruturado.

7.2- Conceitos

Um **vector** (vulgarmente conhecido por **array**) é um tipo estruturado de dados, cujos elementos são do mesmo tipo, estão organizados em ordem sequencial e possui a seguinte sintaxe:

<tipo> nome [tamanho];

onde:

tipo denota um tipo elementar de dados
nome denota um identificador
tamanho denota o número máximo de elementos

A organização sequencial, estabelece uma relação de ordem entre os seus elementos. Queremos com isso dizer, que o primeiro elemento está antes do segundo, o segundo antes do terceiro, e por aí adiante. Esse tipo de organização permite que qualquer elemento seja identificado por um índice.

Um **índice** é uma variável ou uma expressão do tipo inteiro que indica a posição relativa de qualquer elemento num vector.

O **acesso** à um determinado elemento é feito por uma **variável indexada**, constituída pelo nome do vector, seguido de um índice entre parêntesis rectos. Em termos simbólicos:

nome[indice]

Veremos em seguida, um exemplo ilustractivo que nos permitirá consolidar este conceito. Na declaração:

```
const int n = 10;
int A[n];
```

reservamos dez posições contínuas de memória para armazenar a informação do vector A. Cada posição de memória é um elemento do vector do tipo inteiro.

Por defeito, na linguagem C, o índice do primeiro elemento de qualquer vector é igual a zero. Então, o vector anterior possui a seguinte representação:

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

onde: A[0] é o primeiro elemento, A[1] o segundo, A[2] o terceiro, …, A[9] o décimo elemento.

Em função deste exemplo, podemos concluir que o índice de um vector com n elementos é um número inteiro que possui a seguinte propriedade:

$$0 \leq i \leq n - 1$$

É importante salientar que o número de elementos de um vector é um valor constante. Por esse facto, não é permitido em C, efectuar a seguinte declaração:

```
int max = 10;
int A[max];
```

ou

```
int max;
scanf(" %d ", &max);
int A[max];
```

7.3- Acesso à um elemento

Dada a declaração

```
#define  MAX 10

int aluno[MAX];
```

O segmento de código:

```
{
   aluno[2] = 5;
   aluno[9] = 8;
}
```

armazena o número 5 no terceiro elemento do vector aluno e o número 8 no décimo elemento do mesmo vector. Em termos gráficos:

0	1	2	3	4	5	6	7	8	9
		5							8

O segmento de código:

```
{
   aluno[2] = 5;
   aluno[ aluno[2]+ 1] = 24;
}
```

armazena o número 5 no terceiro elemento do vector aluno. Em seguida, adiciona ao conteúdo do terceiro elemento uma unidade e armazena nessa posição o número 24. Mas essa posição é o sétimo elemento desse vector, pois vejamos: aluno[aluno[2]+1]= 24 ⇔ aluno[5+1]= 24 ⇔ aluno[6]= 24.

O segmento de código

```
{
    aluno[4] = 20;
    aluno[ aluno[4] ] = 12;
{
```

armazena o número 20 no quinto elemento do vector aluno. Em seguida, armazena no vigésimo primeiro elemento desse vector o número 12. Mas o vector aluno só possui 10 elementos, então esse índice faz referência há uma posição inválida.

Tome muito cuidado em não utilizar índices inválidos, ou seja índices negativos ou superiores à dimensão do vector. Como o compilador C não detecta esses erros, os programas com esses índices imprimem resultados imprivisíveis.

7.4- Inicialização de vectores

Na declaração de um vector, o sistema operativo reserva uma quantidade de posições consecutivas de memória para armazenar essa variável e coloca em cada posição um valor indefinido (lixo).

A operação de inicialização tem por finalidade limpar esse lixo. Vejamos um exemplo. Dada a declaração:

```
#define   ZEROS 0
#define   MAX     20

int k;
int  nota1[MAX], nota2[MAX];
```

Inicializar um vector, consiste em armazenar em cada elemento, brancos, se o vector for do tipo caracter, ou zeros se o vector for do tipo inteiro ou real. Esta operação é descrita pelo seguinte segmento de código.

```
k = 0;
while (k< MAX)
 {
   nota1[k] = zeros;
   nota2[k] = zeros;
   k++;
 }
```

7.5 - Comando For

As linguagens de programação de alto nível, possuem uma estrutura de repetição especial, denominado por **comando de repetição automática** que é uma generalização do comando **while** e aplica-se quando estão devidamente determinadas, às condições iniciais, às condições de término e o passo da repetição. Sua sintaxe é descrita por:

```
for (condição inicial; condição de término; passo da repetição)
     < Sequencia de comandos >;
```

e é equivalente à:

```
condição inicial;
while (condição de término)
{
   < Sequencia de comandos>;
   passo da repetição;
 }
```

Com este comando, a iniciliazação dos vectores visto no ponto anterior é descrita pelo seguinte segmento de código:

```
for (k=0; k < MAX; k++)
 {
   nota1[k] = zeros;
   nota2[k] = zeros;
 }
```

Ao utilizar o comando de repetição automática, tome o cuidado em não alterar no interior da sequencia de comandos as variáveis de controlo.

7.6- Operador vírgula

O operador vírgula não tem precedência em relação à qualquer outro operador e tem por finalidade encadear várias acções. Por exemplo:

```
pos=0, ind = 1;
```

é equivalente à

```
pos= 0;
ind= 1;
```

Este operador é muito utilizado para a compactação de código. Por exemplo: O segmento de código:

```
fat=1;
for (i=1; i<= n; i++)  fat = fat * i;
```

que calcula o factorial de um número inteiro positivo é equivalente à:

```
for (fat=1,i=1; i<=n; i++)  fat = fat * i;
```

7.7- Carregamento de dados

O carregamento de dados consiste em preencher os elementos de um vector com alguma informação. Esse carregamento pode ser feito por uma acção de leitura ou pela aplicação de uma fórmula matemática. Por exemplo: Carregar a avalição parcelar de 20 alunos de uma turma de computação consiste em:

```
printf(" \n nº aluno  1ª nota   2ª nota  \n ");
for (k = 0; k < MAX; k++)
    scanf(" %d  %d  %d ",&aluno[k],&nota1[k],&nota2[k]);
```

É importante salientar que estamos a supor que MAX é uma constante simbólica igual à 20.

Se o vector tiver poucos elementos e os valores a carregar forem conhecidos, este processo pode ser feito na sua declaração. Por exemplo:

```
int aluno[4] = {3212,3217,3228,3242};
int nota1[4] = {10,9,17,5};
int nota2[4] = {12,6,14,10};
```

7.8- Impressão de resultados

Imprimir os resultados consiste em mostrar na tela o conteudo dos elementos do vector. Por exemplo, imprimir uma pauta com as notas dos 20 alunos do curso de computação, consiste em:

```
for (k=0; k < MAX; k++)
   prinf(" \n  %d   %d   %d   %d ",k+1,aluno[k], nota1[k],nota2[k]);
```

Se quizessemos imprimir a mesma pauta no sentido inverso bastaria percorrer os vectores do término para o inicio, ou seja:

```
for (k=MAX-1; k >=0; k--)
   prinf(" \n %d  %d  %d   %d ",k+1, aluno[k], nota1[k],nota2[k]);
```

7.9- Utilização de um vector como acumulador

A utilização de um vector como acumulador, consiste em declarar um vector auxiliar em que cada elemento é um acumulador. Por exemplo: armazenar à média das notas dos 20 alunos de computação, consiste em percorrer os vectores que contêm essas notas. Para cada iteração k, armazenar na k_ésima posição do vector auxiliar, à média das notas do aluno que está na k-ésima posição dos vectores notas1 e notas2. Lembre-se que o número de elementos do vector auxiliar deve ser igual ao número de elementos dos vectores que contêm essas notas.

```
float  media[MAX];
for (k= 0; k < MAX; k++)
     media[k]= (nota1[k] + nota2[k])/2.0;
```

7.10 – Cálculo do valor máximo

Calcular o valor máximo de um vector, consiste em supor, que o elemento que se encontra na primeira posição é o máximo. Armazenar o conteúdo desse elemento numa variável auxiliar. Comparar em seguida, o conteúdo dessa variável com os restantes elementos do vector. Se encontrarmos um elemento cujo conteudo for maior do que o valor dessa variável, então o conteúdo desse elemento deverá ser armazenado nessa variável e o processo de comparação continua. Quando não for possível realizar mais comparações o conteúdo da variável máximo representa o elemento com o maior valor no vector. Esta estratégia é descrita pelo seguinte segmento de código.

```
float max_nota;
 int i;
 max_nota = media[0];
for(i =1; i< MAX; i++)
    if (max < media[i])  max_nota= media[i];
```

7.11 – Passagem de parâmetros

Vimos que os vectores são estruturas de dados indexadas que servem para armazenar dados do mesmo tipo. Por exemplo, na declaração

```
int  A[4];
```

cada elemento do vector A é do tipo inteiro e está associado há um endereço de memória. Em termos mais precisos, A[0] está associado a &A[0], ... , A[3] está a associado à &A[3].

Nas linguagens de programação modernas a passagem de parâmetros de estruturas indexadas é feita por **referência,** queremos com isso dizer, que na chamada da função, os argumentos são transferidos para a definição da função como ponteiros.

Na definição da função, os argumentos são declarados, com nome do vector seguida da abertura e encerramento de parêntesis rectos. Por exemplo, a função:

```
float modulo(float v[ ], int n)
{
   int i;
   float r = 0;
   for ( i=0; i < n ; i++)  r + = v[i]*v[i];
   return sqrt(r);
}
```

recebe como argumentos um vector do tipo real e um inteiro n. Essa declaração também pode ser feita pelo operador indirecto (*) que denota um ponteiro.

```
float modulo(float *p, int n)
{
   int i;
   float r = 0;
   for ( i=0; i < n; i++)  r + = p[i]*p[i];
   return sqrt(r);
}
```

A sua chamada pode ser descrita pelo segmento de código.

```
int main()
{
    foat x[6]= {3,2,5};
    int m,comprimento;
    m = 3;
    comprimento= modulo(x,m);          /* Chamada da função    */
    return 0;
 }
```

Observe que não tivemos necessidade de colocar o operador endereço (&) na chamada da função porque na declaração do vector cada elemento está associado ao seu endereço.

O argumento v[] da função modulo, contêm o endereço do elemento X[0] da função main(). Como consequência, qualquer alteração ao conteúdo dos elementos de v reflete-se no conteudo dos correspondentes elementos de x.

*Declarar float v[ ] equivale a declarar *p. A notação *p é a mais utilizada na literatura da ciência de computação*

7.12- Strings

Na programação os vectores têm uma importância crucial no armazenamento e na manipulação de dados do tipo texto.

Strings são vectores do tipo caracter com um código especial a delimitar o seu término. Estes possuem a seguinte sintaxe:

char *<nome>[tamanho];*

Por exemplo: **char** palavra[18]; representa um vector do tipo caracter com dezoito elementos. Suponhamos sem perda da generalidade que esse vector contêm a palavra UCANIANO.

U	C	A	N	I	A	N	O										

Essa palavra ocupa apenas os oito primeiros elementos, os restantes contêm um valor indefinido. Como determinar o término dessa palavra?

Na linguagem C, o término de qualquer string é identificada por um caracter especial, denominado por **NULL** e representado pelo caracter '\0' da tabela ASCII.

Por exemplo, a string anterior possui a seguinte representação:

U	C	A	N	I	A	N	O	'\0'									
								↑									
								delimitador									

7.13- Funções de leitura e impressão

A leitura de uma cadeia de caracteres é feita pela função **scanf**() com o formato "%s". Por exemplo:

```
scanf("%s", endereco);
```

lê uma sequencia de caracteres digitado pelo utilizador e os armazena no vector endereco. O processo termina quando o utilizador digitar um caracter em branco. Nessa altura será armazenado o caracter especial **NULL** que representa o término dessa sequencia. No exemplo a seguir, iremos mostrar como definir uma string para receber um nome digitado pelo utilizador.

```
#include <stdio.h>
#include <stdlib.h>
 int main()
{
   char nome[50];
   system("CLS");
   printf(" \n Digite o seu nome:");
   scanf("%s",nome);
   printf( "\n O seu nome : %s",nome);
   return 0;
}
```

cujo funcionamento descrevemos em seguida:

```
Digite o seu nome:  Carlos Alberto Gomes
 O seu nome : Carlos
```

A função **scanf**("%s",nome); interpreta o espaço em branco como o término de uma cadeia de caracteres. Desse modo só o primeiro nome será armazenado no vector.

Observe que não utilizamos o ponteiror **&** no segundo argumento dessa função porque a variável nome é uma estrutura indexada.

Para leitura de textos, a Linguagem C possui a função **gets**() que está declarada e definida na biblioteca-padrão <stdio.h> e possui a seguinte sintaxe:

```
char gets(char *s);
```

Esta função lê uma cadeia de caracteres digitadas pelo utilizador enquanto não for pressionada a tecla [ENTER]. Todos os caracteres são armazenados na string s incluindo o caracter especial **NULL**. Ao armazenar esse caracter o cursor é movimentado para a próxima linha. Vejamos um exemplo ilustractivo:

```
#include <stdio.h>
#include <stdlib.h>
 int main()
{
   char nome[50];
   system("CLS");
   printf(" \n Digite o seu nome:");
```

```
    gets(nome);
    printf( "\n O seu nome : %s",nome);
    system("PAUSE");
    return 0;
}
```

cujo funcionamento descrevemos em seguida:

```
Digite o seu nome:  Carlos Alberto Gomes
O seu nome : Carlos Alberto Gomes
```

Para imprimir uma string, para além da função **printf**() com o formato "%s", podemos utilizar a função **puts**() que está definida na mesma biblioteca-padrão e possui a seguinte sintaxe:

```
int puts(char *s);
```

Esta função imprime o conteudo da string s. Ao encontrar o caracter especial **NULL** termina a impressão e move o cursor para a próxima linha. Vejamos um exemplo ilustractivo:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char nome[50];
    system("CLS");
    printf("Digite o seu nome:");
    gets(nome);
    puts( " O seu nome :");
    puts(nome);
    return 0;
}
```

cujo funcionamento descrevemos em seguida:

```
Digite o seu nome:  Carlos Alberto Gomes
O seu nome :
Carlos Alberto Gomes
```

Cada string impressa com o **puts**() termina com o cursor na próxima linha. Se quizermos imprimir o nome e o sobrenome de uma pessoa numa única linha, teremos de utilizar o comando **printf**(), ou seja:

```
printf("\n  %s   %s \n", nome, sobrenome);
```

Como qualquer string é um vector os métodos de passagem de parâmetros para os vectores aplicam-se as strings.

7.14- Funções de manipulação de strings

O arquivo cabeçalho <string.h> da biblioteca-padrão definida pela ANSI, possui um conjunto de funções de utilização geral. Estudaremos a seguir, as funções mais comuns:

char strcpy(**char** *destino, **char** *origem); Denominada por *String Copy*, copia a string origem para a string destino incluindo o caracter especial NULL e retorna a string destino. Apresentamos em seguida um programa que cria uma cópia de uma frase digitada pelo utilizador.

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char origem[100],destino[100];
    printf("Entre com uma frase: ");
    gets(origem);
    strcpy(destino,origem);
    printf ("\n  Copia da frase :%s",destino);
    printf ("\n Frase original : %s",origem);
    return 0;
}
```

cujo funcionamento descrevemos em seguida:

```
Entre com uma frase: Quem me dera ser onda
Copia da frase : Quem me dera ser onda
Frase original : Quem me dera ser onda
```

int strlen(**char** *st); Denoninada por *String Length*, devolve o número de caracteres existentes na cadeia de caracteres st, sem o caracter especial NULL. Apresentamos em seguida um programa que imprime o número de elementos de uma frase digitas pelo utilizador.

```
#include <stdio.h>
#include <string.h>
int main ()
{
   int tamanho;
   char str[100];
   printf ("Entre com uma frase: ");
   gets (str);
   tamanho= strlen(str);
   printf ("\n Esta frase possui %d caracteres", tamanho);
   return 0;
}
```

cujo funcionamento é descrito por:

Entre com uma frase: O meu pé de laranja lima
Esta frase possui 24 caracteres

int strcmp(**char** *st1, **char** *st2); Denominada por *String Compare*, compara o conteudo da string st1 com a string st2. Devolve um inteiro menor do que zero se st1 for alfabéticamente menor do que st2; Zero se st1 for alfabéticamente igual a st2 e um número maior do que zero se st1 for alfabéticamente maior do que st2. Apresentamos em seguida um programa que compara duas frases digitadas pelo utilizador.

```
#include <stdio.h>
#include <string.h>
int main ()
{
   char str1[100],str2[100];
   printf ("Entre com uma frase: ");
   gets (str1);
   printf ("\n Entre com frase: ");
   gets (str2);
   if (strcmp(str1,str2))
      printf ("\n As frase são diferentes.");
  else
      printf ("\n As frases são iguais.");
  return 0;
}
```

int strcat(**char** *destino, **char** *origem); Denominada por *String Concat*, coloca a cadeia origem imediatamente a seguir da cadeira destino e devolve a cadeia destino. Esta função não verifica se na cadeia destino existe espaço suficiente para receber a cadeia origem. Apresentamos em seguida um programa que irá acrescentar a frase: " Meu amor da rua 11" uma cadeia com o nome do autor digitado pelo utilizador.

```
#include <stdio.h>
#include <string.h>
int main ()
{
   char destino[100],origem[100];
   printf ("Entre com uma frase: ");
   gets (origem);
   strcpy (destino,"Meu amor da rua 11 ");
   strcat (destino,origem);
   printf ("\n %s",destino);
   return 0;
}
```

cujo funcionamento descrevemos em seguida:

Entre com uma frase: de Almeida Santos
Meu amor da rua 11 de Almeida Santos

Na linguagem C não podemos utilizar o comando de atribuição de valor para armazenar uma string numa outra. Para resolver esta operação teremos de utilizar a função strcpy()

7.15 - Exercícios Resolvidos

Problema 7.15.1: O colégio Leonard de Pisa, pretende informatizar os serviços académicos. A primeira fase desse projecto consiste em desenvolver um subprograma para cadastrar (carregar) as notas finais dos alunos. Suponha que o cadastro de alunos seja constituído pelos seguintes dados: número do aluno, sexo e nota final.

Resolução: Vamos analisar numa primeira fase, toda a informação que o subprograma recebe e devolve.

Pelo enunciado, o subproblema recebe três vectores para armazenar os dados dos alunos e devolver esses vectores actualizados. Logo, a melhor forma de efectuar essa comunicação é declarar esses vectores como argumentos passados por referencia.

Para além disso, temos a necessidade de enviar para o programa que irá utilizar esse subprograma, um sinalizador que informará se o carregamento foi efectuado com ou sem sucesso. Esse sinalização deve ser uma variável do tipo inteiro, que contêm os valores lógicos verdadeiro ou falso e será retornado via valor de retorno.

Isso permite-nos concluir que a melhor forma de implementar esse subproblema é uma função do tipo inteiro.

Agora, vamos utilizar o método de refinamente sucessivo para construir um algoritmo que é a solução do nosso problema. Numa primeira abordagem, temos:

Versão 1

1- Ler os dados do aluno
2- Enquanto há alunos para cadastrar faça
3- Armazenar as notas
4- Ler os dados do próximo aluno

Como não conhecemos o número de alunos que iremos processar, o controlo de fim de leitura será determinado pelas seguintes condições: O número do estudante é igual ao sentinela de fim de leitura ou o vector está cheio.

Para verificar se o vector está cheio, basta declarar uma variável local para controlar em qualquer instante, à posição do próximo elemento livre. Seja plivre essa variável.

```
int plivre;
```

Estamos em condições de refinar a versão anterior, com as seguintes acções: declarar as constantes, as variáveis e definir as condições para o término do processo de leitura.

Versão 2

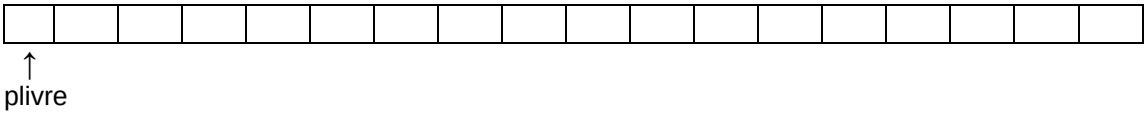
1- Definir as constantes
2- Definir as variáveis
3- Inicializar os vectores
4- Ler os dados do aluno
5- Enquanto (número do aluno ≠ sentinela e vector não está cheio) faça
6- Armazenar os dados do aluno nos vectores
7- Ler os dados do próximo aluno

O próximo passo consiste em escrever com mais rigor, o funcionamento da variável que controla as inserções no vector.

O controlo das inserções é feito pela variável plivre. A primeira acção consiste em dizer que a próxima posição livre encontra-se no primeiro elemento do vector, ou seja

```
plivre = 0;
```

Com essa operação, dizemos que o vector está vazio, em termos gráficos:



Em seguida, armazenamos os dados do aluno nos vectores indexados por plivre e adicionamos uma unidade à essa variável.

Mas, o utilizador pode cometer erros de lançamento. Para evitar que dados inconsistêntes sejam armazenados efectuamos uma prévia operação de validação.

Estamos em condições de refinar a versão anterior com a descrição detalhada do processo de inserção de novos elementos.

Versão 3

1- Definir as constantes
2- Definir as variáveis
3- Inicializar os vectores

4- Inicializar à posição livre
5- Ler os dados do aluno
6- Enquanto (número do aluno ≠ sentinela e vector não está cheio) faça
7- Validar os dados do aluno
8- Se os dados forem consistentes
9- Armazenar os dados do aluno nos vectores
10- Adicionar uma unidade à posição livre
11- ler os dados do proximo aluno
12- no caso contrário
13- imprimir uma mensagem de erro
14- Ler os dados do mesmo aluno

O próximo passo consiste em escrever com mais rigor a validação de dados. Essa validação, consiste em verificar se as regras do problema estão cumpridas, ou seja: Se o número do aluno é positivo, se o sexo é igual a 'M' ou 'F' e se as notas satisfazem a relação 0 ≤ notas ≤ 20.

Para implementar essa estratégia necessitamos de definir um conjunto de variáveis auxiliares, para armazenar os dados digitados pelo utilizador. Em seguida analisar o seu conteudo. Ao encontrar qualquer inconsistência, imprimir uma mensagem de erro e rejeitar esses dados.

Para verificar se o vector está cheio, basta comparar o conteudo da variável plivre com o valor da constante simbólica MAX. Porque?

Ao terminar o processo de carregamento verificar em que condições esse processo foi encerrado. Se o utilizador digitou o sentinela de fim de leitura, devolver o valor verdadeiro, no caso contrário, devolver o valor falso.

Apresentamos em seguida, uma função que é a solução do nosso problema.

```
*-----------------------------------------------------------------------------------------------------
Constantes simbólicas válidas para o programa
 -----------------------------------------------------------------------------------------------------*/
# include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#define MAX 50
#define SENTINELA  -1
#define FALSE  0
#define TRUE 1
#define ZEROS  0
#define MASCULINO  'M'
#define FEMININO  'F'

*-----------------------------------------------------------------------------------------------------
 Objectivo: Carregar as notas finais de uma turma
Parâmetros por referencia: Vectores para armazenar os dados
Valor de retorno: Verdadeiro ou Falso
 -----------------------------------------------------------------------------------------------------*/
```

```
int carregar_dados(int num[ ], int nota[ ], char sexo[ ])
{
  int    aux_num, aux_nota, plivre= 0, ind,consistente;
  char  aux_sexo;
  for (ind=0; ind < MAX; ind++)
     {
        num[ind] = nota[ind] = ZEROS;
     }
  printf(" \n Entre com o número do aluno sexo e nota \n ");
  scanf(" %d  %c  %d ",&aux_num, &aux_sexo, &aux_nota);
  while ((plivre < MAX ) && ( aux_num != SENTINELA))
      {
         consistente= TRUE;
         if (aux_num <= 0)
            {
              printf("  \n Erro: Número do aluno inválido ");
              consistente= FALSE;
            }
         if ((aux_ sexo != MASCULINO) || ( aux_sexo != FEMININO))
            {
              printf("  \n Erro: Sexo do aluno inválido ");
              consistente= FALSE;
            }
         if (( aux_nota < 0 ) || ( aux_nota > 20))
            {
              printf(" \n Erro: nota do aluno inválida ");
              consistente= FALSE;
            }
         if (consistente)
            {
               num[plivre]  = aux_num;
               sexo[plivre]  = aux_sexo;
               nota[plivre]   = aux_nota;
               plivre++;
               printf(" \n Entre com o número do proximo aluno, sexo e nota \n ");
            }
         else
               printf(" \n Entre com o número do mesmo aluno, sexo e nota \n ");
         scanf(" %d  %c  %d ",&aux_num, &aux_sexo, &aux_nota);
      }
  if (aux_num == SENTINELA)
     retrun True;
  else
     return False;
}
```

Vamos estudar este exemplo com mais detalhe. Os dados são armazenados nos vectores de forma compacta não existindo entre eles "buracos". Então, o índice plivre permite que se determine em qualquer instante o que é informação válida e o que é lixo.

Se plivre fizer referencia a primeira posição do vector todos os elementos desse vector contêm lixo, logo o vector está vazio.

Observação: *No inicio do programa anterior, tivemos a necessidade de definir uma série de includes. Para tornar os programas mais elegantes, a Linguagem C, permite que se crie um arquivo separado onde se possa organizar as definições das constantes e macros. Grava-se esse arquivo com um nome, por exemplo: inicio.h. Para utilizar essas constantes, basta escrever a directiva #include inicio.h*

Problema 7.15.2: Desenvolva programa com funções para calcular a soma vectorial.

Resolução: Por definição, dados dois vectores A e B a soma vectorial é um vector C com a mesma dimensão dos vectores A e B que satisfaz a seguinte propriedade:

$$c_i = a_i + b_i. \ \forall_{i>0}$$

Numa primeira abordagem, a solução deste problema consiste nos seguintes passos:

Versão 1:

1- *Carregar os dados nos vectores*
2- *Calcular à soma vectorial*
3- *Imprimir à soma vectorial*

Pelo enunciado, fazem parte das entidades de entrada dois vectores com MAX elementos e fazem parte das entidades de saída, um vector cujos elementos satisfazem à fórmula matemática anterior. Então, podemos declarar as seguintes variáveis:

```
int  a[MAX], b[MAX], c[MAX];
```

onde

```
#define MAX  35
```

O próximo passo consiste em definir com mais rigor, as acções para carregar os dados, calcular à soma vectorial e imprimir o resultado. Mas, antes veremos algumas condições necessárias para efectuar essas operações.

Como não conhecemos o número de alunos da turma, o controlo do término da leitura de dados, será efectuado por um sentinela. Se o número de alunos a carregar for maior do que a dimensão do vector o processo de carregamento deverá ser abortado e deveremos imprimir uma mensagem a notificar tal facto.

Estamos em condições de refinar a versão anterior com as seguintes acções: declarar as constantes, as variáveis, definir as condições necessárias para efectuar o processo de carregamento, o cálculo vectorial e a impressão de resultados.

Versão 2:

1- Declarar as constantes
2- Declarar as variáveis
3- Carregar os dados
4- Se carregamento não terminou
5- Imprimir mensagem de erro
6- No caso contrário
7- Calcular à soma vectorial
8- Imprimir os resultados

O próximo passo consiste em descrever com mais rigor as acções: carregar os dados, verificar se o carregamento terminou, calcular à soma vectorial e imprimir os resultados. Essas acções serão descritas pelos seguintes subprogramas:

Subprograma: Carregar os dados

Comunicação:
Parâmetros por valor: Nada
Parâmetros por referência: Vectores A e B
Valor de retorno: Posição livre
Tipo de Subprograma: Função

Algoritmo:

1- Declarar as variáveis locais
2- Inicializar à variável que controla à posição livre
3- Ler os dados
4- Enquanto há dados e o vector não está cheio
5- Armazenar dados
6- Adicionar uma unidade à posição livre
7- Ler os próximos dados
8- Se dado lido for o sentinela de fim de leitura
9- Devolver à posição livre
10- No caso contrário
11- Devolver Zero

Subprograma: Calcular a soma vectorial

Comunicação
Parâmetros por valor: Posição do último elemento inserido
Parâmetros por Referencia: Vectores A, B e C
Tipo de subprograma: Procedimento

Algoritmo:

1- Declarar as variáveis locais
2- Percorrer os vectores até último elemento inserido
3- Aplicar fórmula matemática

Subprograma: Imprimir os resultados

Comunicação:
Parâmetros por valor: Posição do último elemento inserido
Parâmetros por Referência: Vector C
Tipo de subprograma: Procedimento

Algoritmo:

1- Declarar as variáveis locais
2- Percorrer o vector até ao último elemento inserido
3- Imprimir o conteúdo do elemento

Subprograma: Verificar se o carregamento terminou

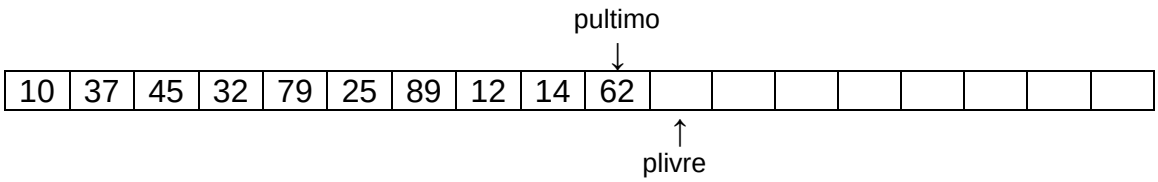
Comunicação:
Parâmetros por valor: Posição livre
Parâmetros por referência: Nada
Valor de retorno: Verdadeiro ou Falso
Tipo Subprograma: Função

Algoritmo:

1- Se a posição livre for igual à zeros
2- Devolver Falso
3- No caso contrário
4- Devolver Verdadeiro

Como determinar em cada instante a posição do último elemento inserido?

Pela figura que descrevemos em seguida:



à posição do último elemento inserido é igual à posição livre menos uma únidade.

Estamos condições de apresentar um programa que é a solução do nosso problema.

Para tomar conhecimento, veremos em primeiro lugar, a implementação dessa solução com variáveis globais.

```
/*-------------------------------------------------------------------------------------------------
 Objectivos: Imprimir à soma vectorial
 -------------------------------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#define  SENTINELA  -1
#define  MAX  35
#define  ZEROS  0
/*-------------------------------------------------------------------------------------------------
 Variáveis glovais
 -------------------------------------------------------------------------------------------------*/
 int a[MAX], b[MAX], c[MAX];
 int plivre;
/*-------------------------------------------------------------------------------------------------
 Protótipos de funções
 -------------------------------------------------------------------------------------------------*/
void  carregar_dados();
void calc_soma_vectorial());
void imprime_resultados());

/*-------------------------------------------------------------------------------------------------
Objectivo: Carregar os dados nos vectores A e B
 -------------------------------------------------------------------------------------------------*/
void  carregar_dados( )
{
   int x, y;
   plivre = 0;
   pritnf(" \n Entre com o primeiro elemento do vector A e do vector B ");
   scanf(" %d %d ",&x,&y);
   while ((plivre < MAX) && (x != SENTINELA))
     {
         a[plivre] =  x;
         b[plivre] = y;
         plivre++;
         pritnf(" \n Entre com um elemento do vector A e do vector B ");
         scanf(" %d %d ",&x,&y);
      }
   if (x ! == SENTINELA)  plivre = ZEROS;
}
/*-------------------------------------------------------------------------------------------------
Objectivo: Calcular a soma vectorial
-------------------------------------------------------------------------------------------------*/
void calc_soma_vectorial()
{
  int i;
  for (i= 0; i <= plivre-1; i++)  c[i] = a[i] + b[i];
}
```

```
/*-------------------------------------------------------------------------------------------------------
Objectivo: Imprimir a soma vectorial
-------------------------------------------------------------------------------------------------------*/
void imprime_resultados()
{
  int i;
  printf(" \n Valor da soma vectorial ");
  for (i = 0; i <= plivre-1; i++)
      printf(" \n valor do elemento %d = %d ", i+1,c[i]);
}

/*-------------------------------------------------------------------------------------------------------
 Função principal
-------------------------------------------------------------------------------------------------------*/
int main()
{
  setlocale(LC_ALL,"Portuguese");
  carregar_dados(x,y);
   if (plivre == ZEROS)
       printf(" \n Erro: Carregamento não concluído com sucesso");
  else
       {
          calc_soma_vectorial();
          imprime_resultados();
       }
   return ZEROS;
}
```

Agora, veremos a mesma implementação com a técnica de passagem de parâmetros. Devemos salientar que essa técnica insere-se nas boas prácticas de programação.

```
/*-------------------------------------------------------------------------------------------------------
 Objectivos: Imprimir à soma vectorial
 -------------------------------------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#define  SENTINELA  -1
#define  MAX  35
#define  ZEROS  0

/*-------------------------------------------------------------------------------------------------------
 Protótipos de funções
 -------------------------------------------------------------------------------------------------------*/
int carregar_dados(int a[ ], int b[ ]);
void calc_soma_vectorial(int ultimo, int a[ ] , int b[ ], int c[ ]);
void imprime_resultados(int ultimo, int c[ ]);
```

```
/*-------------------------------------------------------------------------------------------------
Objectivo: Carregar os dados nos vectores A e B
Parâmetros por referência: Vectores A e B
Valor de retorno: Posição livre
 -------------------------------------------------------------------------------------------------*/
int  carregar_dados(int a[ ], int b[ ])
{
   int x, y, poslivre= 0;
   pritnf(" \n Entre com o primeiro elemento do vector A e do vector B ");
   scanf(" %d %d ",&x,&y);
   while ((poslivre < MAX) && (x != SENTINELA))
     {
         a[poslivre] =  x;
         b[poslivre] = y;
         poslivre++;
         pritnf(" \n Entre com um elemento do vector A e do vector B ");
         scanf(" %d %d ",&x,&y);
     }
   if (x == SENTINELA)
        return poslivre;
   else
       return ZEROS;
}

/*-------------------------------------------------------------------------------------------------
Objectivo: Calcular a soma vectorial
Parâmetros por valor: Última posição inserida e os vectores A e B
Parâmetros por referência: Vector C
-------------------------------------------------------------------------------------------------*/
void calc_soma_vectorial(int ult, int a[ ] , int b[ ], int c[ ])
{
  int i;
  for (i= 0; i <= ult; i++)  c[i] = a[i] + b[i];
}


/*-------------------------------------------------------------------------------------------------
Objectivo: Imprimir a soma vectorial
Parâmetros por valor: Última posição inserida e o vector C
-------------------------------------------------------------------------------------------------*/
void imprime_resultados(int ult, int c[ ])
{
   int i;
   printf(" \n Valor da soma vectorial ");
   for (i = 0; i <= ult; i++)
       printf(" \n valor do elemento %d = %d ", i+1,c[i]);
}
```

```
/*-------------------------------------------------------------------------------------------------
 Função principal
-------------------------------------------------------------------------------------------------*/
int main()
{
   int x[MAX], y[MAX], z[MAX];
   int plivre;
   setlocale(LC_ALL,"Portuguese");
   plivre = carregar_dados(x,y);
   if (plivre == ZEROS)
        printf(" \n Erro: Carregamento não concluído com sucesso");
   else
        {
           calc_soma_vectorial(plivre-1,x, y, z);
           imprime_resultados(plivre-1,z);
        }
    return ZEROS;
}
```

Problema 7.15.3: A Elm production, é uma empresa de produção e comercialização de música em discos compactos. Desenvolva um subproblema para encontrar um determinado álbum que está cadastrado no seu sistema informático. Suponhamos que temos 50 discos e que não há discos repetidos.

Resolução: A resolução deste problema, consiste em analisar numa primeira fase, toda a informação que o subprograma recebe e devolve.

O subprograma recebe: O vector com o cadastro dos discos, o código do disco que procuramos e a posição do último elemento inserido.

```
int  codigo, ultimo;
int  discos[ ];
```

O subprograma devolve: A posição do elemento encontrado.

Isto quer dizer que estamos em presença de uma função que possui a seguinte comunicação:

Parâmetros por valor: código do disco e a posição do último elemento inserido.
Parâmetros por referência: Vector com o cadastro de discos.
Valor de retorno: índice do elemento no vector.

Numa primeira abordagem, a solução deste problema é descrita pelos seguintes passos:

Versão 1

1- *Procurar o código do disco*
3- Devolver o resultado da busca

O próximo passo, consiste em descrever de forma genérica uma acção para procurar o código do disco. Procurar esse código, consiste em percorrer o vector até ao último elemento inserido e verificar se o conteúdo de cada elemento é igual ao valor que procuramos. Lembre-se que o ultimo elemento inserido faz referência ao elemento anterior à posição livre.

Com base nesse conceito, estamos em condições de refinar a versão anterior.

Versão2

1- Percorrer o vector discos até ao último elemento inserido
2- Comparar conteúdo dos elementos com o código do disco
3- Devolver o resultado

O próximo passo, consiste em descrever com mais rigor, um método para procurar o código do disco.

Devemos salientar que não temos qualquer garantia da existência do disco no vector. Logo, o nosso processo de busca termina com duas condições exclusivas: O disco encontra-se no vector e guardamos a sua posição ou o disco não se encontra e guardamos um valor especial a indicar tal facto.

Por definição, um índice é um número maior ou igual a zero. Então, faz sentido indicar à inexistênçia de um elemento no vector, por um número menor do que zero. Seja -1 esse número.

A estratégia para resolver este problema consiste em utilizar uma flag que possui o seguinte funcionamento: Suponhamos que o disco não está no vector. Então, armazenamos na flag o valor -1. Em seguida, percorremos o vector e comparamos o conteúdo de cada elemento, com o valor que procuramos. Se essa comparação for verdadeira, guardamos na flag o valor desse índice.

No fim do processo, a flag indica a localização do elemento. Se o conteudo da flag for igual à -1 o elemento não foi encontrado, caso contrário o elemento foi encontrado.

Estamos em condições de refinar a versão anterior com a descrição de um método para encontrar o disco.

Versão3

1- Armazenar numa flag o valor -1
2- Percorrer o vector discos até ao último elemento inserido
3- Se o conteúdo do elemento for igual ao código do disco
4- Armazenar na flag o valor dessa posição
5- Devolver a flag

Estamos em condições de apresentar uma função que é solução do nosso problema.

```
/*-------------------------------------------------------------------------------------------------
Objectivo: Procurar um determinado disco num vector
Entrada  : código do disco, índice do último elemento inserido, vector discos
Saida    : -1 se não encontrou, caso contrário Índice do elemento no vector
 -------------------------------------------------------------------------------------------------*/
int busca(int ultimo,int codigo,int discos[ ]);
{
  int  ind, pos= -1;
  for (ind = 0; ind <= ultimo; ind++)
      if (discos[ind] == codigo)  posicao=  ind;
  return pos;
}
```

Esta função está correcta? Numa primeira abordagem parece-nos que sim. Mas, vamos analizar o seu comportamento, com os seguintes casos:

1º Caso (O elemento não está no vector): A função armazena na variável pos o valor -1. Em seguida, percorre o vector na totalidade. Em cada iteração compara o conteúdo desse elemento com o valor que procuramos. Como essa comparação é sempre falsa, não se altera o conteúdo dessa variável. O ciclo termina e devolve o valor correcto.

2º Caso (O elemento está na última posição do vector): Na última iteração, o conteudo desse elemento é igual ao valor que procuramos. Então, a variável pos passa a armazenar o valor dessa posição. O ciclo termina e devolve o valor correcto.

3º Caso (O elemento está na primeira posição do vector): Vimos, pelos casos anteriores, que a função inicia o percurso do vector nessa posição. Compara o conteúdo desse elemento com o valor que procuramos. Como o resultado dessa comparação é verdadeiro, à variável pos armazena esse índice. Mas, a função contínua à percorrer os restantes elementos do vector e à compara-los com o valor que procuramos. Isto é um absurdo, essa estratégia não é aceitável.

Um algoritmo eficaz, deve suspender o processo de busca, quando encontrar o elemento.

Para melhorar a versão anterior, temos a seguinte estratégia: Quando o elemento for encontrado, devolver a sua posição através do comando return. Se o vector for percorrido na totalidade, então o elemento não foi encontrado. Nesse caso, deve-se devolver via comando return o valor -1. Apresentamos em seguida a função com essa estratégia.

```
/*-------------------------------------------------------------------------------------------------
Objectivo: Procurar um determinado disco num vector
Entrada  : código do disco, índice do último elemento inserido, vector discos
Saida    : -1 se não encontrou, caso contrário Índice do elemento no vector
 -------------------------------------------------------------------------------------------------*/
```

```
int busca(int ultimo,int codigo,int discos[ ]);
{
  int ind;
  for (ind= 0; ind <= ultimo; ind++)
      if (discos[ind] == codigo)  return  ind;
  return -1;
}
```

Problema 7.15.4: O Instituto Nacional de Meteorologia (INAMET), pretende criar um banco de dados com as estatísticas pluviométricas anuais das estações meteorologicas existêntes no território nacional. Todos os meses, as diversas estações existentes nas capitais de província, enviam para Luanda, os seguintes dados: código da província e quantidade de chuva em milímetros. Pelos estudos realizados, assuminos que nenhuma estação recede mais do que 99 centímetros de chuva anualmente. Desenvolver um programa para determinar as estações que se enquandram nos seguintes grupos: 0 a 9, 10 a 19, 20 a 29, 30 a 39, 40 a 49, …, 90 a 99.

Resolução: Pela descrição do enunciado, não temos qualquer indicação para determinar o término do processo de leitura. Como o código de cada estação meteorológica é um inteiro maior ou igual a zero, podemos utilizar como sentinela de fim de leitura, um valor negativo.

Numa primeira abordagem, a solução deste problema pode ser descrita pelos seguintes passos:

Versão 1

1- Ler o código da estaçao e à quantidade de chuva
2- Enquanto há dados
3- Actualizar as estatisticas das estações por grupo
4- *Ler o código da próxima estação e a quantidade de chuva*

Pelo enunciado, temos como entidade de entrada, o código da estação meteorológica e a quantidade de chuva. Como entidade de saída, um vector com as estatistivas das estações por grupo. Logo, podemos declarar as seguintes variáveis:

```
int  estacao, chuva;
int  grupos[MAX];
```

onde:

```
#define MAX 10
#define SENTINELA  -1
```

O próximo passo consiste em descrever com mais detalhe, a acção enquanto há dados. Essa acção, consiste em processar à quantidade de chuva

coletada pelas estações meteorológicas, enquanto o utilizador não digitar o sentinela de fim de leitura.

Estamos em condições de refinar a versão anterior, com as seguintes acções: declarar as constantes, as variáveis, e formalizar as acções para leitura de dados e o processamento da quantidade de chuva.

Versão 2

1- Declarar as constantes
2- Declarar as variáveis
3- Ler o codigo da estação e a quantidade de chuva
4- Enquanto codigo da estação for diferente de sentinela
5- Actualizar as estatisticas das estações por grupo
6- Ler o codigo da próxima estação e a quantidade de chuva

Agora, vamos escrever com mais rigor, a instrução para actualizar as estatisticas das estações por grupo.

Actualizar essas estatísticas, consiste em determinar um índice que satisfaça à seguinte relação:

Quantidade de chuva	**Grupo**
chuva ≥ 0 e chuva ≤ 9	0
chuva ≥ 10 e chuva ≤ 19	1
chuva ≥ 20 e chuva ≤ 29	2
chuva ≥ 30 e chuva ≤ 39	3
chuva ≥ 40 e chuva ≤ 49	4
chuva ≥ 50 e chuva ≤ 59	5
chuva ≥ 60 e chuva ≤ 69	6
chuva ≥ 70 e chuva ≤ 79	7
chuva ≥ 80 e chuva ≤ 89	8
chuva ≥ 90 e chuva ≤ 99	9

Esse índice determina o grupo da estação. Em seguida, adicionar uma unidade ao elemento do vector grupos, referenciado por esse índice. Mas, antes de iniciar esse processo, é nessário inicializar esse vector.

Estamos em condições de refinar à versão anterior, com a descrição de uma acção para actualizar as estatisticas das estações por grupo.

Versão 3

1- Declarar as constantes
2- Declarar as variáveis
3- Inicializar o vector grupos
4- Ler o codigo da estação e a quantidade de chuva
5- Enquanto codigo da estação for diferente de sentinela
6- Validar os dados de leitura

7- Se dados são válidos
8- Calcular um índice do grupo
9- Adicionar uma unidade ao elemento desse grupo
10- Ler o codigo da próxima estação e a quantidade de chuva
11- No caso contrário
12- Imprimir uma mensagem de erro
13- Ler os mesmos dados

O próximo passo, consiste em descrever com mais detalhe, as acções para inicializar o vector grupos, Validar os dados, calcular um índice do grupo e adicionar uma unidade ao elemento desse grupo.

Inicializar o vector grupo é trivial.

Validar os dados é muito simples e já foi vista nos exemplos anteriores.

Calcular um índice do grupo e adicionar uma unidade à esse grupo, consiste em:

1º Estratégia (Força bruta): Aplicar o comando condicional encadeado, cujo segmento de código descrevemos em seguida:

```
if (chuva < 9)
    grupo[1] = grupo[1] + 1
else if (chuva <  19)
            grupo[2]  = grupo[2] + 1
       else if (chuva < 29)  . .
```

2º Estratégia: Consiste em utilizar uma propriedade inerente ao problema: Os intervalos estão classificados de 10 em 10. Logo, o índice do grupo, pode ser determinado pelo quociente da divisão da quantidade de chuva por 10. Esta estratégia é descrita pelo seguinte segmento de código:

```
i =  chuva/10;
grupo[i+1] = grupo[i+1] + 1;
```

Contudo, estas estratégias só poderão ser aplicadas, se a quantidade de chuva digitada pelo utilizador, estiver no intervalo 0 a 99.

Estamos em condições de apresentar um programa que é a solução do nosso problema.

```
/*-----------------------------------------------------------------------------------------------
 Objectivo: Classificação das estações meteorológicas
 Entrada  : Código da estação, quantidade de chuva registada
 Saida    : Vector com a respectiva classificação por grupos
 -----------------------------------------------------------------------------------------------*/
#include <stdio.h>
#include <locale.h>
```

```
#include <sdtdlib.h>
#define   MAX  10
#define  SENTINELA -1
#define  TRUE  1;
#define  FALSE 0;
int main()
{
    int  estacao, chuva, ind;
    int  grupos[MAX];
    setlocale(LC_ALL,"Portuguese");
    for (ind=0; ind < MAX; ind++)  grupos[ind] = 0;
    printf(" \n  Entre com a estação e a quantidade de chuva: ");
    scanf("  %d  %d ",&estacao,&chuva);
    while (estacao != SENTINELA)
      {
        erro = FALSE;
        if ( estacao < 0 )
          {
            printf( " \n Erro: Código da estação inválido");
            erro = TRUE;
          }
        if ((chuva < 0) || (chuva >  99))
          {
              printf(“ \n Erro: Quantidade de chuva não permitida”);
              erro = TRUE;
          }
        if (!erro)
          {
             ind = chuva/10;
             grupos[ind+1]++;
          }
        printf(" \n  Entre com a estação e a quantidade de chuva : ");
        scanf(" %d  %d ",&estacao,&chuva);
      }
    system("PAUSE");
    return 0;
}
```

Problema 7.15.5: Uma firma de desenvolvimento de software, necessita que se desenvolva um subprograma para inserir um elemento numa determinada posição de um vector.

Resolução: A resolução deste problema, consiste em analisar numa primeira fase, toda a informação que a função recebe e devolve.

O subprograma recebe: O vector, a posição de inserção, o elemento a inserir, e o índice do último elemento inserido.

O subprograma devolve: O vector, a posição do último elemento inserido actualizada e uma notificação a descrever o estado da operação (sucesso ou insucesso).

Isto quer dizer que estamos em presença de uma função que possui a seguinte comunicação:

Parâmetros por valor: Elemento a inserir e a posição de inserção
Parâmetros por referência: Vector e o índice do último elemento inserido
Valor de retorno: Verdadeiro ou Falso

Numa primeira abordagem, a solução deste problema pode ser descrita pelos seguintes passos:

Versão 1

1- Se as condições de inserção estão satisfeitas
2- Inserir o elemento

O próximo passo, consiste em descrever as condições de inserção e o processo de inserção.

Para que a operação de inserção possa realizar-se, é necessário que estejam garantidas as seguintes condições: O vector não pode estar cheio, e a posição de inserção deve estar no intervalo $0 \leq k \leq ultimo + 1$, onde ultimo é o índice do último elemento inserido.

Na chamada da função, deveremos verificar se à primeira condição está satisfeita, enquanto à segunda, que tem à finalidade de preservar a compactação dos elementos inseridos no vector, deverá ser testada no corpo da função.

O processo de inserção, consiste em mover para à posição seguinte, todos os elementos do vector, que estão entre o último elemento inserido e a posição de inserção; Colocar na posição de inserção o elemento e em seguida, actualizar o índice do ultimo elemento inserido.

Estamos em condições de refinar a versão anterior, com a descrição pormenorizada das acções para inserir um elemento na k-ésima posição do vector.

Versão2

1- Declarar uma variável local
2- Se (k < 0 ou k > ultimo+1)
3- Devolver falso
4- Senão
5- Percorrer o vector do ultimo elemento inserido a posição de inserçao
6- Mover o elemento para à posição seguinte
7- Inserir um novo valor na posição de inserção
8- Adicionar uma unidade ao último elemento inserido

9- Devolver verdadeiro

Estamos condições de apresentar uma função que é solução do nosso problema.

```
/*-------------------------------------------------------------------------------------------------
Objectivo: Inserir um elemento na k-ésima posição de um vector
Parâmetros por valor: Elemento a inserir e a posição de inserção
Parâmetros por referência: Vector e o índice do último elemento inserido
Valor de retorno: Verdadeiro ou Falso
-------------------------------------------------------------------------------------------------*/
int  inserir_pos_k(int x, int k, int  vetor[ ], int *ultimo);
{
  int i;
  if ( k < 0  || k > ultimo+1)   return 0;
  else {
          for(i = *ultimo; i >= k; i--)  vetor[i+1] = vetor[i];
          vetor[k]= x;
         *ultimo ++;
         return 1;
       }
}
```

Apresentamos em seguida um trecho do segmento de código que invoca essa função:

```
ultimo = plivre-1;
if (cheio(plivre))
    printf("ERRO: O vector está cheio");
else if (inserir_pos(x,k,a, &ultimo)
          printf("\n Operação concluída com sucesso ");
       else
          printf(" \n Operação não concluída com sucesso");
```

7.16 - Exercícios Propostos

7.16.1-Desenvolva um programa para armazenar num vector 100 números inteiros positivos, digitados pelo utilizador. Imprima em seguida, os números primos e mostre as suas respectivas posições.

7.16.2-Uma imobiliária possui 25 vendedores. Cada venda é feita por um vendedor que regista numa ficha os seguintes dados: Número do vendedor e o valor da venda. Desenvolver um programa para gerar um relatório com o total de vendas por vendedor.

7.16.3-Desenvolva um programa com funções para armazenar num vector n números inteiros. A partir desse vector, construa dois vectores, um com os

números negativos e outro com os números positivos. Este programa também deve mostrar a quantidade de elementos de cada vector.

7.16.4-Desenvolva um programa para armazenar n números inteiros positivos, diferentes, digitados pelo utilizador. Utilize uma função para ordenar esses n elementos e imprimir o conteúdo do vector.

7.16.5- Desenvolva um procedimento que recebe dois vectores com n elementos. Cada vector não possui elementos repetidos e devolva um vector que seja a intercalação desses vectores.

7.16.6- Desenvolva uma função para verificar se duas sequenciais armazenadas em dois vectores são do tipo capicua. Sequencias capicuas são aquelas têm o mesmo valor se forem percorridas da esquerda para a direita ou da direita para a esquerda. Exemplo: ovo, osso, roma e amor

7.16.7- Desenvolva um procedimento que recebe dois vectores com n elementos e devolve um vector que seja a união desses vectores.

7.16.8- Desenvolva um procdimento que recebe dois vectores A e B com n elementos e devolve um vector C com esses elementos ordenados. Lembre-se que os vectores A e B, não estão ordenados.

7.16.9- Desenvolva um programa com funções para ler um conjunto de carros e a quantidade de quilómetros que cada carro faz com um litro de combustível. Em seguida, calcular e imprimir, o modelo mais económico, e quantos litros de combustível cada carro gasta aos 100 quilómetros.

7.16.10- Uma empresa de comésticos possui 50 empregados. A sua administração pretende efectuar um aumento salarial com base nos seguintes critérios: Têm direito à aumento salarial, os empregados que possuem tempo de serviço superior há cinco anos ou um salário inferior à 25.000,00 Kz. Esse aumento é baseado nas seguintes regras: O empregado que satisfizer as duas condições, terá um aumento de 35%; O empregado que apenas satisfizer a condição de tempo de serviço terá um aumento de 25% e o empregado que satisfizer a condição de salário terá um aumento de 15 %. Desenvolva um programa com funções para cadastrar os dados dos empregados (número do empregado, salário e data de admissão) em seguida, imprimir um relatório com os números dos empregados que não terão aumento salarial e um relatório com os números dos empregados e o seu salário actual.

7.16.11- Uma Universidade pretende saber se não há alunos a frequentar as disciplinas de introdução à lógica de computação e introdução à linguagem de programação I. Para efectuar qualquer matrícula, os serviços académicos registam o número do passe e o código da disciplina. Essas informações são armazenadas electronicamente. Desenvolva um programa com funções para carregar os dados dos alunos e imprimir todos os estudantes matriculados as duas disciplinas.

7.16.12- Desenvolva uma função para contar o número de vezes que um elemento aparece num vector. Suponha que o vector possua n elementos.

7.16.13- Desenvolva um procedimento para calcular o maior e o menor elemento do vector. Suponha que o vector possua n elementos.

7.16.14- Desenvolva uma função que recebe dois vectores com n elementos e verifica se eles são iguais.

7.16.15- Desenvolva uma função para armazenar em ordem crescente n números inteiros não negativos digitados pelo utilizador. Preste atenção que o utilizador não entra com os números ordenados.

7.16.17- Desenvolva um programa com funções para armazenar num vector X n números inteiros e num vector Y m números inteiros. Para cada elemento de Y encontre e imprima os seus divisores armazenados em X.

7.16.18- Desenvolva um programa para imprimir um relatório anual de vendas. O relatório deve mostrar os totais mensais e o total anual. Cada venda é registada numa ficha: valor da venda, número do mês. Estas fichas não estão ordenadas.

7.16.19- Uma empresa multinacional possui fábricas em cinco cidades do país. A empresa possui n empregados. A ficha de cada empregado contém os seguintes campos: número do empregado e código da cidade. Estas fichas não estão ordenadas. As informações dos empregados estão em dois vectores: Numeros de empregado e códigos das cidades. Construir um programa com funções que ordene todos os empregados de modo que eles sejam impressos por ordem crescente de empregado dentro de cada cidade.

7.16.20- Desenvolva um procedimento que receba dois vectores com n elementos e devolva um vector que seja a diferença entre eles.

7.16.21- Desenvolva um procedimento que receba dois vectores com n elementos e devolva um vector que seja a sua intersecção.

7.16.22- Desenvolva um procedimento que receba um vector com n elementos e devolve um outro vector que seja o seu inverso.

7.16.23- A Trans-Urbana Ltda é uma empresa de transportes rodoviários, que possui uma frota de cinquenta táxis. Para motivar os motoristas, a adminsitração dessa empresa, criou um prémio mensal para o melhor motorista. O melhor motorista é aquele que tem a melhor receita do mês. O controlo das receitas é feito pelo seguinte procedimento: No inicio e no fim de cada dia faz-se a leitura aos taximetros. Os taxis utilizam uma tarifa única de 250,00 Kz. Desenvolva um programa para determinar o motorista que teve a melhor receita e o motorista que teve a pior receita do dia. Vamos supor que não se apuram receitas iguais.

7.16.24- Para tentar descobrir se um dado está viciado, o dono de um cassino, lançou esse dado n vezes. Desenvolva um programa com funções que determine o número de ocorrências de cada face dado n lançamentos. Um dado tem seis faces.

7.16.25- Desenvolva um procedimento para inserir um elemento num vector ordenado de tal modo que o vector continue ordenado.

7.16.26- Desenvolva um procedimento que recebe um vector com n elementos e devolve um outro vector sem os elementos repetidos

7.16.27- Desenvolva um procedimento para remover num vector com n elementos, um elemento que se encontra na k-ésima posição.

7.16.28- Desenvolva um procedimento que recebe um vector com n elementos e devolve o valor do elemento com o maior número de ocorrências.

7.16.29- Desenvolva um procedimento que recebe dois vectores com n elementos e verifica quantos elementos de um vector pertencem ao outro.

7.16.30- Uma empresa de distribuição alimentar, pretende corrigir os preços de venda de seus artigos em função da taxa de desvalorização da moeda. O Instituto Nacional de Estatistica disponibiliza um vector com a possível desvalorização mensal no ano. Desenvolva um programa com funções para carregar os dados dos artigos (código do artigo, preco de custo, preço de venda, último mês de ajuste e quantidade em stock), em seguida, o utilizador entra com o mês, o programa ajusta os preços dos artigos que têm stock e imprime todos os artigos cuja margem de comercialização tornou-se inferior a X %.

7.16.31- Um clube de vídeo possui 50 clientes que devem ser armazenados num vector. Num segundo vector deverão ser armazenados as quantidades de vídeos que esses clientes alugaram. Sabe-se que para cada dez vídeos, o cliente tem direito a um vídeo de graça. Desenvolva um programa com funções que mostre os todos os clientes que têm condições para receber um vídeo de graça. Esse programa também deverá mostrar a quantidade de vídeos que esses clientes têm direito.

7.16.32- Desenvolva um programa para controlar o stock de uma empresa. Inicialmente o utilizador deve carregar dois vectores com 20 elementos. O primeiro vector corresponde aos códigos dos artigos e o segundo a quantidade de mercadoria em stock existente. Em seguida, o programa deverá ler um conjunto indeterminado de dados que contêm o código do cliente, o código do artigo e a quantidade a comprar. O programa deve verificar:

- Se o código do produto solicitado existe. Se existir tentar atender o pedido; caso contrário emitir a mensagem de erro: *código inexistênte*.
- Cada pedido só poderá ser atendido na íntegra. Caso contrário, emitir a mensagem: *Não temos stock suficiente para esta mercadoria*. Se puder atender o pedido, emitir a mensagem: *pedido atendido, volte sempre*.
- Efectuar a actualização do stock sómente se o pedido for atendido

- Imprimir os códigos dos artigos com stock's actualizados.

7.16.33- Desenvolva um programa para simular o controlo bancário. Inicialmente o programa deve ler um conjunto de 30 contas bancárias e seus respetivos clientes. Os códigos das contas devem ser armazenados num vector de números inteiros e os saldos num vector de números reais. Cada saldo deverá ser cadastrado na mesma posição do código da conta. Depois de efectuar a leitura desses valores mostrar o menu:

1- Depositos
2- Saque
3- Consultas activo banco
1- Finalizar algoritmo

- Para efectuar um depósito deve-se solicitar o código da conta e o valor depositado. Se a conta não estiver cadastrada, mostrar a mensagem de erro: *Conta não cadastrada* e voltar ao menu. Se a conta estiver cadastrada, actualizar o seu saldo.

- Para efectuar um saque deve-se solicitar o código da conta e o valor a ser sacado. Se a conta não estiver cadastrada, mostrar a mensagem de erro: *Conta não cadastrada* e voltar ao menu. Se a conta estiver cadastrada, verificar se o saldo é suficiente para realizar a operação. Estamos a supor que a conta não possa ficar com saldo nulo. Se o saldo for suficiente, realizar a operação e voltar ao menu; caso contrário emitir mensagem de erro *Saldo insuficiente* e voltar ao menu principal.

- Consultar a activo do banco que consiste em somar o saldo de todos os clientes, mostrar esse valor e voltar ao menu.

- O programa só termina quando for digitada essa opção.