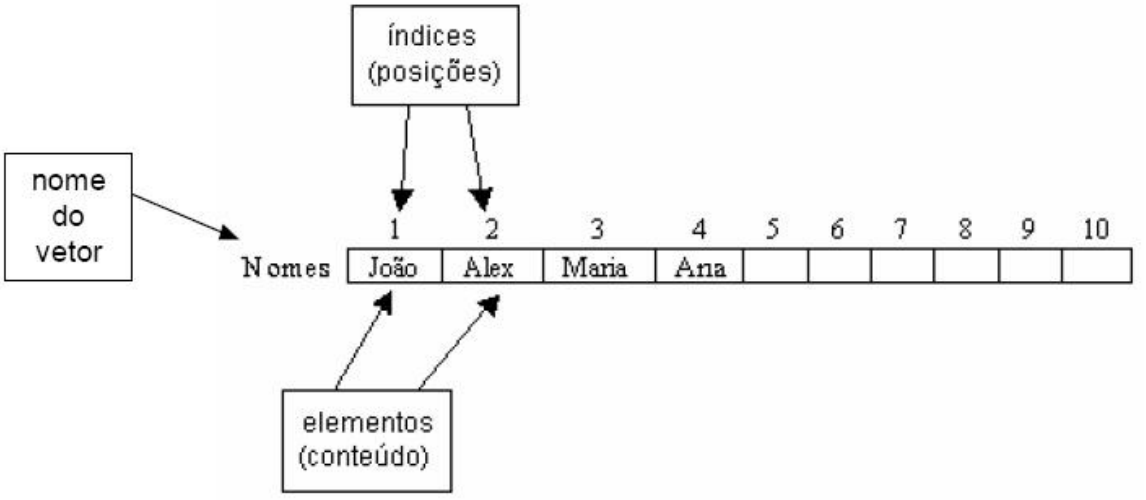
Programação II
AULA 03 – Vetores

7 Vetores

Podemos definir um Vetor como uma variável dividida em vários "pedaços", em várias "casinhas", onde cada pedaço desses é identificado através de um número, referente à posição de uma determinada informação no vetor em questão. O número de cada posição do vetor é chamado de índice.

Conceito: Vetor é um conjunto de variáveis, onde cada uma pode armazenar uma informação diferente, mas todas compartilham o mesmo nome. São associados índices a esse nome, que representam as posições do vetor, permitindo assim, individualizar os elementos do conjunto.

Podemos imaginar que na memória do computador o vetor seja mais ou menos da seguinte forma:



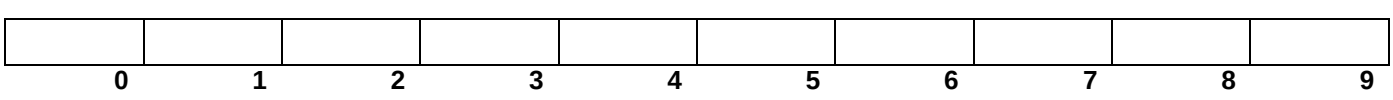
Declaração de Vetor em C:

Sintaxe: *Tipo nome_vetor[tamanho];*

O tipo deve ser especificado de acordo com o tipo de informação que será armazenado no vetor (ex. int float, char,...). E o tamanho representa a quantidade de elementos que este vetor irá conter. É importante dizer que na linguagem C as matrizes começam pelo índice 0 que guarda o primeiro elemento da matriz. Para entender melhor, considere que seja necessário declarar um vetor do tipo inteiro que contenha 10 elementos. Isto é feito da seguinte forma:

```
int vetor_exemplo[10];
```

Isso porque o “vetor_exemplo” vai de 0 a 9, ou seja, contém 10 elementos:



Também é possível inicializar o vetor no momento de sua declaração. Para isso veja a sintaxe abaixo:

Sintaxe: *Tipo nome_vetor[tamanho]={lista_de_valores};*

Sendo que todos os elementos da lista de valores devem ser separados por vírgula e serem todas do mesmo tipo de dados especificado. A seguir temos a declaração do "vetor_exemplo" com os valores atribuídos.

```
int vetor_exemplo[10]={0,1,2,3,4,5,6,7,8,9};
```

Atribuição

Para atribuir um valor em um vetor é necessário escrever em cada um de seus elementos individualmente. Para acessar um elemento de um vetor basta utilizar o identificador do vetor e a sua posição (índice), na seguinte sintaxe:

Sintaxe: *<identificador>[<indice>];*

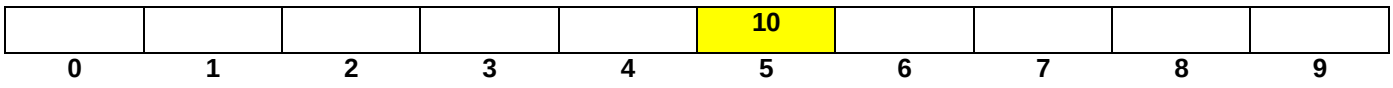
Como por exemplo:

v[5];

No nosso exemplo, v[5] é o sexto elemento do vetor v, lembre-se que o primeiro elemento está no índice 0. Esse elemento é uma variável do tipo inteiro, porque declaramos, em nosso exemplo, o vetor v de tipo inteiro. Assim como é possível atribuir valores, ler e escrever no conteúdo de variável inteira, também é possível atribuir, ler e escrever no conteúdo de v[5]. Por exemplo, se executarmos a seguinte atribuição:

v[5]=10;

então o valor 10 é atribuído ao elemento de índice 5 do vetor v:



Como PREENCHER um Vetor

Para preencher um vetor com informações (dados), ou seja, armazenar informações em um vetor, é necessária uma estrutura de repetição, pois um vetor possui várias posições e temos que preencher uma a uma. A estrutura de repetição normalmente utilizada para vetores é o FOR, então veja no exemplo abaixo como ler um vetor de 10 posições:

Exemplo:

```
for (i=0; i<10; i++)
{
      v[i]=i*10;
}
```

Como ESCREVER um Vetor

Para escrever um vetor, ou seja, para escrever o conteúdo de cada posição de um vetor, também precisamos utilizar uma estrutura de repetição, já que os vetores possuem mais de um conteúdo (mais de uma posição). Normalmente utiliza-se a estrutura FOR também para escrever o vetor. Veja no exemplo abaixo, como escrever um vetor de 10 posições, isto é, como escrever o conteúdo de cada uma das 10 posições do vetor:

Exemplo:

```
for (i=0; i<10; i++)
{
     printf(”%i”, v[i]);
}
```

Exemplo de um programa completo utilizando vetores e outros conceitos estudados até o momento:

```
#include<stdio.h>
#include<conio.h>
int main() {
     int vetor1[5]={1,2,3,4,5}; //declaração e inicialização do vetor
     int vetor2[5]={6,7,8,9,0}; //declaração e inicialização do vetor
     int vetorSoma[5]; //declaração do vetor que armazenará o resultado da soma dos dois
     vetores
     int x;
     printf("Programa que soma os dois vetores abaixo:\n");
     printf("vetor1={1,2,3,4,5}\n");
     printf("vetor2={6,7,8,9,0}\n");
     printf("\nVetor resultante da soma:\n");
     for (x=0; x<5; x++)
     {
          vetorSoma[x]=vetor1[x]+vetor2[x]; //soma os valores e armazena no vetorSoma
          printf("vetorSoma[%d]: %2d\n",x,vetorSoma[x]); //exibe na tela os valores
        armazenados no vetorSoma
     } //fim do for
     getch();
     return 0;
} //fim do programa
```

Os vetores são muito usados para criar uma string de caracteres, pois em C não existe nenhum tipo de dados para definir uma string. A declaração de um vetor contendo uma string sempre deve ser maior que o número de caracteres, pois o compilador acrescenta automaticamente no final da string um espaço nulo que indica o seu término. Este segundo exemplo é muito simples, mostra apenas como podem ser feitas a declaração e inicialização de um vetor de string.

```
#include<stdio.h>
#include<conio.h>
int main()
{
     char c1 = 'a';
     char vetor1[30]="Aprendendo a mexer com string\n";
     /*Imprimindo os dados na tela*/
     printf("O tipo de dado char guarda apenas um caractere\n");
     printf("A variavel c1 do tipo char contem o caractere: %c\n",c1);
     printf("\n");
     printf("Para trabalhar com uma string deve ser declarado um vetor do tipo char");
     printf("\nO vetor do tipo char contem a string: %s",vetor1);
```

```
    getch();
    return 0;
} /*fim do programa*/
```

Vetores são estruturas estáticas

Os vetores são chamados de estruturas estáticas, pois têm o seu tamanho definido durante a codificação do programa. Durante a execução do programa, o tamanho do vetor não pode ser alterado, ou seja, não pode ser aumentado nem diminuído. Uma vez que nós não sabemos quantas notas serão fornecidas pelo usuário, é difícil estipular previamente um tamanho para o vetor. Tudo isso pode acarretar dois problemas para o programa:

1. Durante a codificação, o programador especificou um tamanho para o vetor que é muito maior do que o necessário. Nesse caso, tem-se memória alocada para o vetor que está sendo desperdiçada.
2. Também, o programador pode especificar um tamanho para o vetor que é menor do que o necessário. Nesse caso, o vetor não será suficiente para armazenar todos os valores que o usuário deseja fornecer.

O segundo caso é sempre pior do que o primeiro. Caso o programador não queira ter nenhum dos dois problemas, então ele terá que recorrer a uma estrutura que não seja um vetor. Por exemplo, listas encadeadas. Essas estruturas serão estudadas mais à frente e são muito mais complexas do que os vetores.

Portanto, temos que utilizar um pouco de bom senso para escolher o tamanho do vetor e tratarmos o código para por exemplo, evitar que ocorra o problema 2.

EXERCÍCIO PARA LABORATÓRIO

EL_07) Faça um algoritmo e um programa capaz de ler em um vetor A um número não previamente determinado de valores inteiros positivos. Esses valores devem ser fornecidos pelo usuário até que ele entre um valor especial. Depois de ler esses valores, copie-os para um outro vetor B. Essa cópia deve ser feita de forma que o valor armazenado no primeiro elemento do vetor A seja copiado para o último elemento do vetor B. O segundo elemento do vetor A seja copiado para o penúltimo elemento do vetor B. E assim por diante.

EL_08) Faça um algoritmo e um programa capaz de ler 100 valores e armazená-los em um vetor A. Depois ler mais outros 100 valores e armazená-los em um vetor B. Faça com que um vetor C armazene a soma dos vetores A e B. O primeiro elemento do vetor C deve armazenar a soma do primeiro elemento do vetor A com o primeiro elemento do vetor B. O segundo elemento do vetor C deve armazenar a soma do segundo elemento do vetor A com o segundo elemento do vetor B. E assim por diante.

EL_09) Faça um algoritmo e um programa que leia 500 números reais do teclado e armazene em um vetor. Encontre e imprima o menor e o maior valor do vetor.