Introdução aos vetores em *C*

Como explicado no texto introdutório ao conceito de vetores, um vetor é uma sequência de dados ocupando posições consecutivas de memória e por isso existe uma ordem natural entre seus elementos, o primeiro elemento, o segundo e assim por diante. A grande vantagem de usar vetor é poder trabalhar com muitas variáveis utilizando um único nome para esse "agregado" de variáveis. Para isso existe uma sintaxe especial para pegar cada elemento do vetor, em posições específicas, como o primeiro elemento ou o décimo elemento.

A linguagem *C* trata o conceito de *vetor* de um modo bastante próximo ao modo como é possível implementar o conceito. Ilustraremos isso mostrando que podemos usar *apontadores* para pegar elementos dos vetores.

Neste texto: definir/imprimir vetor; definir "string"; funções para "string"; vetor e apontador; definir/imprimir matriz; vetor/matriz dinâmicas.

1. Definindo e imprimindo um vetor

Pode-se construir um vetor em *C* fornecendo todos seus elementos, como em: `int vet[] = {21, 22, 23, 24};`.

Pode-se também construir de modo "dinâmico", dependendo dos valores que variáveis assumem em meio ao código. Para isso deve-se declarar o máximo de posições que o vetor ocupará (como `#define MAX 10 ... int vet[MAX];`) e depois define-se cada elemento, por exemplo, usando um comando de atribuição, como: `vet[i] = 10+i;`, que define o valor do elemento na posição *i* como o valor `10+i`.

No exemplo a seguir construímos um vetor com *MAX* elementos, iniciando no *10+i* e seguindo até o *10+n-1*. Nesse exemplo indicamos, em um comentário, como usar apontador (**(v+i)=10+i*) para definir os elementos de um vetor (usando *aritmética de ponteiros*).

Exemplo 1. Construir um vetor e imprimir seus elementos.

```
#include <stdio.h>
// Declarar uma constante (para ser usada como dimensao maxima do vetor)
#define MAX  10

// Funcao para imprimir elementos de vetor em uma mesma linha
// Nao precisa saber do total alocado para o vetor efetivo, pois para pegar o elemento i basta
// saber a primeira posicao, digamos 'apont' e pegar '*(apont + i)'
void imprime_vetor1 (int vet[], int tam) {
  int i;
  for (i=0; i<tam; i++)
    printf("%d ", vet[i]);
  printf("\n"); // ao final quebre a linha
  }

// Esta funcao define os valores para o vetor (passagem por referencia/endereco)
// Lembrando: todo vetor e' passado por referencia 'a funcao (logo
// alterar seus valores localmente implica em alterar os valores do
// parametro efetivo!
// Por ser vetor como parametro, nao precisa declarar a dimensao (nao precisa reservar espaco, e'
apenas uma referencia)
// Mas se fosse matriz precisaria saber dimensao coluna
```

```
void definirVetor (int v[], int tam) {
  int i;
  for (i=0; i<tam; i++) // definir vetor/lista com: (10,11,12,13,...10+n-1)
    v[i] = 10+i; // Aqui altera' o 'parametro efetivo'! Tambem pode-se usar como apontador:
*(v+i) = 10+i;
  }

int main (void) {
  int n=10, vet[MAX];    // alocar MAX posicoes para o vetor 'vet' (pode-se usar menos)

  definirVetor(vet, n); // parametro efetivo 'vet' por ser vetor, se alterado na funcao, altera o
efetivo!

  // Imprime o vetor/lista de uma so' vez (na mesma linha)
  printf("\nO vetor: ");
  imprime_vetor1(vet, n); // em C NAO e' possivel imprimir todos os dados de um vetor => fazer
uma funcao...
  return 0;
  }
```

Importante: Em *C*, usar um *vetor como parâmetro (formal) em função* implica que estará usando **passagem por referência (ou endereço)**!
Portanto, dentro da função, se alterar o valor de qualquer elemento do vetor (*parâmetro formal*), isso implicará que, durante a execução, o seu *parâmetro efetivo* (aquele que é *passado efetivamente* para função) terá o elemento correspondente alterado. Por este motivo, no exemplo acima, ao fazer uma atribuição dentro da função (`v[i] = 10+i;`), estamos na verdade alterando os valores do *parâmetro efetivo* `vet`, por esse motivo usamos o nome **passagem por referência (ou endereço)**, é como se o *parâmetro formal* `v[]` fosse uma **referência** (um "apelido") para o *parâmetro efetivo* `vet[]`.
Se tiver dúvidas, reveja o texto sobre parâmetros de funções.

2. Defininido uma *cadeia de caracteres* ("string")

Em *C* é possível definir um vetor com caracteres, formando uma palavra, ou seja, uma *cadeia de caracteres*, que abreviadamente é denominada "*string*". Para isso pode-se fazer uma atribuição como constante ou ler os dados como palavra. Isso é ilustrado no exemplo abaixo.

Exemplo 2. Trabalhando com "strings". Uma "string" fixa e digitar uma frase, imprimindo o número de caracteres nela.

```
#include <stdio.h>
#include <string.h> // para usar a funcao 'strlen(...)'
void main (void) {
  int i;
  char string[] = "0123456789abcdefghih"; // (1) - define a "string"
  // Imprimir cada caractere da string com seu codigo ASCII
  printf("Imprime caracteres e seus codigos ASCII na forma de tabela\nColuna 1 => caractere;
coluna 2 => seu codigo ASCII\n");
  for (i=0; i<strlen(string); i++)
    printf("%20c : %3d\n", string[i], string[i]); // em C inteiro pode ser "lido" como caractere
e vice-versao (via codigo ASCII)
  // Pode-se imprimir como uma frase toda usando o formatador %s
  printf("Frase: %s\n", string);
  // A string tem 20 caracteres, testando os caracteres 19 'h' e 20 '\n'
  printf("A string tem 20 caracteres, testando os caracteres 19 '%c' e 20 '\\n' em ASCII=%d\n",
string[19], string[20]);
  }
```

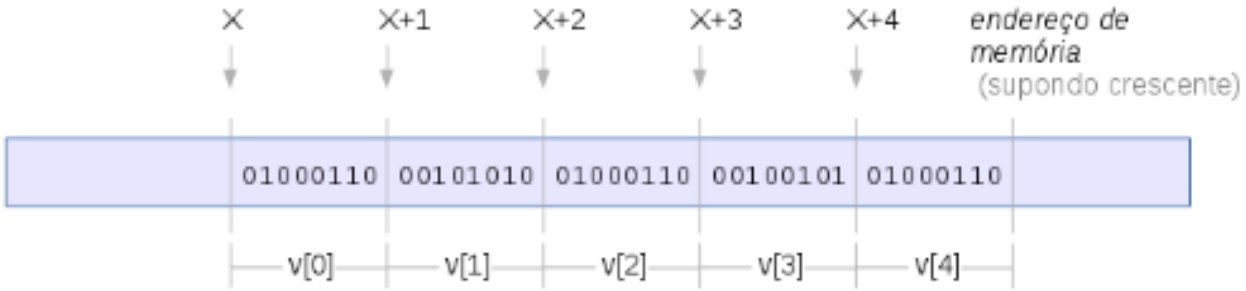
Deve-se notar que em *C*, ao fazer a leitura de uma "string" (por exemplo, via teclado com `scanf("%s", &string);`), automaticamente é inserido o caractere '\0' ao final da "string".

Assim, no exemplo acima ao declarar a "string" com o comando `char string = "0123456789abcdefghih";`, essa terá 20 caracteres "úteis" (a última posição "útil" será a posição 19, com o caractere 'h') e na posição 20 existirá o caractere '\0' (cujo código ASCII é o zero - 0).

3. Vetores e apontadores

Uma vantagem didática da utilização da linguagem *C* em um curso introdutório de programação é que essa é uma linguagem que não esconde muitos detalhes de como os conceitos são implementados. Isso possibilita mostrarmos os fundamentos da computação, os princípios elementares que permitem a construção de sistemas mais sofisticados. Nesse sentido o conceito de vetor é ilustrativo, pois sua implementação depende da existência de apontadores.

Ao declarar um vetor, significa que associamos um nome de variável (o "nome do vetor") a uma sequência de posições de memória, todas de mesmo tipo (logo de mesmo tamanho). Na verdade essa variável guarda o endereço inicial do vetor, o endereço de seu primeiro elemento.



Por exemplo, ao declarar o vetor com `int v[5];`, são reservadas 5 posições de memória, cada uma com o número de *bytes* suficientes para armazenar um dado do tipo *int* (cada *int* geralmente ocupa 2 *bytes* ou 16 *bits*). Vamos supor que *sizeof(int)* devolve o número de *bytes* ocupados por qualquer número inteiro (essa função existe precisamente com este sentido na biblioteca `stdlib.h`). Assim, se o primeiro elemento do vetor está na posição de memória *X*, então o endereço do segundo será *X+sizeof(int)* e assim por diante até o último desses 5 elementos, que estaria na posição *X+5*sizeof(int)*.

O nome de um vetor em *C* é na verdade uma variável *apontadora* e esse fato retoma a observação de *C* não esconder detalhes. Além disso a linguagem permite calculemos a posição de um elemento de vetor somando ao seu endereço inicial o número de elementos que desejamos pular (e.g. `v+3`). Fazendo isso, automaticamente o compilador traduz essa expressão deslocando o número de *bytes* do tipo envolvido e por isso, fazer `v+3`, é o mesmo que pegar o elemento da posição 3 ou `v[3]`.

Como a declaração de uma variável apontadora é feita usando um asterisco (como em `int *apont;`), vamos apresentar um código simples que declara um vetor e depois usamos um apontador para mostrar que, com ele, também podemos "passear" por todos os elementos do vetor.

Exemplo 3. Construindo um vetor e imprimindo seus elementos via um apontador.

```
#include <stdio.h>
void imprimeVetor (int *apont, int n) {
  int i;
  printf("Vetor:");            // supondo X ser o primeiro endereco em 'apont'
  for (i=0; i<n; i++)
    printf(" %d", *(apont+i)); // pegar elemento na posicao 'X + i*sizeof(int) - isso e' o mesmo
que apont[i]
  printf("\n");
  }

int main (void) {
  int vet1[] = { 5, 4, 3, 2, 1 }; // define vetor com 5 posicoes de 0 ate' 4
  int *apont;
  int N = 5;
  imprimeVetor(vet1, N); // passa para "apont" o endereco inicial do vetor
  return 0;
  }
```

Assim, também é possível alterar os valores em qualquer posição do vetor a partir do endereço. Por exemplo, se fizermos

```
  int vet1[5], i; // no lugar de linha 1 da main
  apont = vet1; // ou apont = &vet1; ou apont = &vet1[0];
  for (i=0; i<5; i++) *(apont+i) = 5-i;
```

geraríamos novamente o vetor `vet1 = { 5, 4, 3, 2, 1}`. Experimente!

4. Funções úteis da biblioteca *string.h*

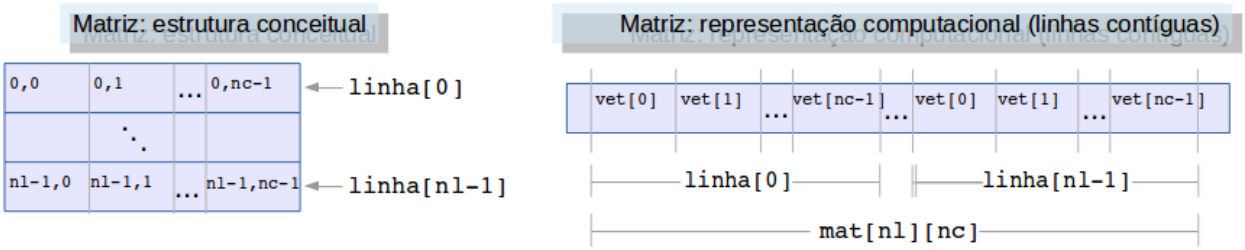
A implementação da função `strlen` (que está na biblioteca `string.h`) usa esse fato, a existência de um caractere especial que só pode ser utilizado em "string" como finalizador para encontrar o tamanho da "string". Assim, se fizer um comando como `string[10] = '\0'; printf("%s\n", string);`, será impresso na tela `0123456789`.

Existem outras funções na biblioteca `string.h`, uma delas é a `strcpy(str1,str2)` que copia o conteúdo da "string" *str2* sobre a *str1*, desde que a declaração delas seja compatível (ou seja, *str2* não pode ter mais caracteres do que a declaração inicial de *str1*). Outra função bastante útil é a `strcmp` que compara lexicograficamente (ordem dos dicionários) duas "strings", do seguinte modo, o comando `strcmp(str1, str2)` devolve

- negativo, se *str1* menor lexicograficamente que *str2*
- zero, se *str1* igual a *str2*
- positivo, se *str1* maior lexicograficamente que *str2*

5. Defininido vetor de vetor (ou matriz)

Uma vez que um vetor/lista é armazenado consecutivamente na memória, então podemos armazenar vários vetores também consecutivamente para obter algo que represente (e seja tratado como) uma **matriz**. Como na imagem abaixo, em que cada alocamos *nl* vetores consecutivos na memória, cada vetor com *nc* posições de memória, desde `vet[0]` até `vet[nc-1]`. Assim, podemos ver este agregado de dados como uma matriz `mat[][]`, cuja primeira linha é o primeiro vetor e assim por diante. Isso está ilustrado na imagem abaixo.



Desse modo obtemos uma estrutura com dois índices que pode ser manipulada como uma matriz (como em *mat[i][j]*). No exemplo abaixo ilustramos isso de dois modos, imprimindo as linhas da matriz e mostrando que pode-se passar como parâmetro uma linha de matriz para uma função que tenha como *parâmetro formal* um vetor.

Exemplo 4. Construindo matriz como vetor de vetor e ilustrando que cada linha dela é um vetor, podendo portanto ser usado para chamar uma função que tem como parâmetro um vetor.

```
#include <stdio.h>

#define MAXL 10 // declarar constantes (para ser usada como dimensoes maximas da matriz)
#define MAXC 10

// Funcao que soma elementos no vetor (precisa do numero de elementos no vetor para saber quando
parar)
int soma_linha (int linha[], int n) { // usando como parametro um vetor (NAO precisa da dimensa,
explicacao abaixo)
  int i, soma = 0;
  for (i=0; i<n; i++)
    soma += linha[i];
  return soma;
  }

// Copiar aqui a funcao do exemplo 1: imprime_vetor1

// Funcao para imprimir linhas de matriz (linha por linha): usa 'imprime_vetor1(...)' para
imprimir uma linha
// Precisa do numero de colunas pois na pratica os elementos estao em linhas de MAXC elementos
void imprime_matriz (int mat[][MAXC], int numLinhas, int numColunas) {
  int i;
  for (i=0; i<numLinhas; i++)
    imprime_vetor1(mat[i], numColunas); // para pegar inicio de 'mat[i]' usa-se o valor de MAX
  }

void main (void) {
  int mat1[MAXL][MAXC]; // alocar MAXL*MAXC posicoes para a matriz 'mat1' ('mat1' e' um vetor de
vetor)
  int n = 10, i, j, nl, nc, prim, seg;

  // Vetor de vetor ou matriz
  nl = 3; nc = 4;
  for (i=0; i<nl; i++)    // foram alocados na memoria MAXL*MAXC
    for (j=0; j<nc; j++)  // mas usaremos apenas nl*nc
      mat1[i][j] = i*nc + j;

  // Pegando o primeiro e segundo elemento da terceira linha de mat1: mat1[2][1]
  prim = mat1[2][0]; seg = mat1[2][1]; // pegar primeiro e segundo elemento da linha mat1[2]
  printf("O primeiro e segundo elemento da terceira linha da matriz: %d e %d\n", prim, seg);

  // Imprime a matriz
  printf("\nA matriz: \n"); // informe o que vira impresso a seguir (na mesma linha devido ao
parametro end=""
  imprime_matriz(mat1, nl, nc);

  // Verifique que cada linha da matriz e' de fato um vetor
  // Usar a funcao 'somalinha(...)' que soma elementos em vetor
```

```
  for (i=0; i<nl; i++)
    printf("Soma da linha %2d = %3d\n", i, soma_linha(mat1[i], nc)); // note que cada linha
'mat1[i]' tem 'nc' elementos uteis
  }
```

6. Defininido vetor de vetor (ou matriz) de modo dinâmico.

Uma vez que um vetor é armazenado consecutivamente na memória, então podemos armazenar vetores de modo consecutivo para obter algo que represente (e seja tratado como) *matriz*. Neste caso obtemos uma estrutura com dois índices (como em *mat[i][j]*). No exemplo abaixo ilustramos isso de dois modos, imprimindo as linhas da matriz e mostrando que pode-se passar como parâmetro uma linha de matriz para uma função que tenha como *parâmetro formal* um vetor.

Exemplo 5. Construindo matriz como vetor de vetor e ilustrando que cada linha dela é um vetor, podendo portanto ser usado para chamar uma função que tem como parâmetro um vetor.

```
#include <stdio.h>
#include <stdlib.h> // para 'malloc' e 'free'

#define MAXL 10 // declarar constantes (para ser usada como dimensoes maximas da matriz)
#define MAXC 10

// Copiar aqui a funcao do exemplo 1: imprime_vetor1

// Funcao para imprimir elementos de vetor em uma mesma linha – usando aritmetica de ponteiro
void imprime_vetor2 (int *pont, int tam) {
  int i;
  for (i=0; i<tam; i++)       // "*(pont+i)" significa, pegar a posicao do primeiro elemento
    printf("%d ", *(pont+i)); // (em 'linha') e deslocar i posicoes (do tipo 'int')
  printf("\n"); // ao final quebre a linha
  }

// Funcao que soma elementos no vetor: outra versao tratando o vetor como apontador
int soma_linha (int *linha, int n) { // usando como parametro apontador para inteiro
  int i, soma = 0;
  // Uma vez que um vetor e' na verdade uma sequencia de posicoes de memoria, C
  // simplesmente anota o endereco da primeira posicao no nome do vetor 'linha',
  // por isso e' a mesma coisa que declarar como parametro formal o apontador 'int *linha'
  for (i=0; i<n; i++)
    soma += *(linha + i);
  return soma;
  }

// Funcao para imprimir linhas de matriz (linha por linha): usa funcao 'imprime_vetor1(...)'
// Precisa do numero de colunas pois na pratica os elementos estao em linhas de MAXC elementos
void imprime_matriz (int mat[][MAXC], int numLinhas, int numColunas) {
  int i;
  for (i=0; i<numLinhas; i++)
    imprime_vetor2(mat[i], numColunas); // para pegar inicio de 'mat[i]' usa-se o valor de MAX
  }

void main (void) {
  int *vet2; // declara um apontador para inteiro que depois apontara' para o vetor dinamico
  int mat1[MAXL][MAXC];   // alocar MAX*MAX posicoes para a matriz 'mat1' ('mat1' e' um vetor de
vetor)
  int (*apontador)[MAXC]; // declara um apontador para linhas com MAXC elementos, se usar apenas
'int *apontador',
                          // '*(apontador[i]+j) = i*nc+j;' poderia resultar advertencia indicada
em (1)
  int n = 10, i, j, nl, nc, prim, seg;

  // Vetor de vetor ou matriz
  // Definir a amtriz de 2 modos (com indices de matriz ou como apontador)
  nl = 3; nc = 4;
```

```
  apontador = mat1; // pegue o endereco inicial da matriz - (1) se usar "int *apontador", na
declaracao
                    // acima, poderia resultar advertencia: "warning: assignment from
incompatible pointer type"
  for (i=0; i<nl; i++)
    for (j=0; j<nc; j++) // ilustra 2 forma de manipular matriz/vetor
      if (j%2) // se j impar => definir com indices
        mat1[i][j] = i*nc + j;
      else     // se j par   => definir com apontador
        *(apontador[i] + j) = i*nc + j; // deslocar i linhas (i*MAXC) e depois deslocar j
elementos

  // Pegando o primeiro e segundo elemento da terceira linha de mat1: mat1[2][1]
  prim = mat1[2][0]; seg = mat1[2][1]; // pegar primeiro e segundo elemento da linha mat1[2]
  printf("O primeiro e segundo elemento da terceira linha da matriz: %d e %d\n", prim, seg);

  // Imprime a matriz
  printf("\nA matriz: \n"); // informe o que vira impresso a seguir (na mesma linha devido ao
parametro end=""
  imprime_matriz(mat1, nl, nc);

  // Verifique que cada linha da matriz e' de fato um vetor
  // Usar a funcao 'somalinha(...)' que soma elementos em vetor
  for (i=0; i<nl; i++)
    printf("Soma da linha %2d = %3d\n", i, soma_linha(mat1[i], nc)); // note que cada linha
'mat1[i]' tem 'nc' elementos uteis

  // Agora definindo vetor com tamanho dinamico
  n  = 6;
  vet2 = malloc(sizeof(int) * n); // aloca dinamicamente n posicoes para inteiros ('Memory
ALLOCation')

  // Define vetor com n elementos exatamente
  for (i=0; i<n; i++)
    *(vet2+i) = 10 + i;
  // Imprime o vetor/lista de uma so' vez (na mesma linha)
  printf("\nO vetor definido via alocacao dinamica: ");
  imprime_vetor1(vet2, n); // em C NAO e' possivel imprimir todos os dados de um vetor => fazer
uma funcao...

  free(vet2); // depois de usar pode-se devolver as posicoes para sistema operacional
  }
```

Leônidas de Oliveira Brandão
http://line.ime.usp.br

Alterações