MAC 2166 – Introdução à Computação

POLI - PRIMEIRO SEMESTRE DE 2007

Material Didático

Prof. Ronaldo Fumio Hashimoto

VETORES E PONTEIROS

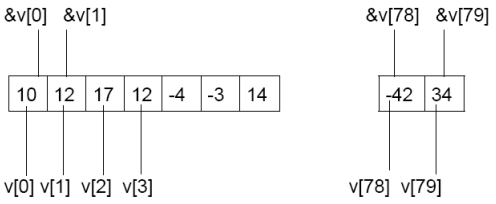
Objetivo

O objetivo desta aula é relacionar o tipo **vetor** com ponteiros.

Vetores

Vimos na aula anterior que **vetores** são estruturas indexadas utilizadas para armazenar dados de um mesmo tipo: **int**, **char**, **float** ou **double**. Por exemplo, a declaração

```
int v[80]; /* declara um vetor de inteiros de nome v com 80 casas */
```

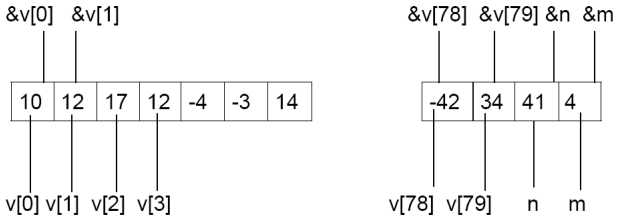


Cada casa do vetor `v` (ou seja, `v[0]`, `v[1]`, ..., `v[79]`) é um inteiro. Além disso, cada casa tem um endereço associado (ou seja, `&v[0]`, `&v[1]`, ..., `&v[79]`).

Uma pergunta que poderíamos fazer é como um vetor fica armazenado na memória. A organização das variáveis na memória depende de como o sistema operacional faz gerenciamento da memória. Em geral, para ser mais eficiente, o sistema operacional tende a colocar as variáveis sucessivamente. Assim, a alocação do vetor na memória é feita de forma sucessiva, ou seja, da maneira como ilustrada na figura acima: `v[0]` antes de `v[1]`, que por sua vez antes de `v[2]` e assim por diante. Assim, as variáveis declaradas como

```
int v[80]; /* declara um vetor de inteiros de nome v com 80 casas */
int n, m;
```

poderiam ser alocadas de forma sucessiva como



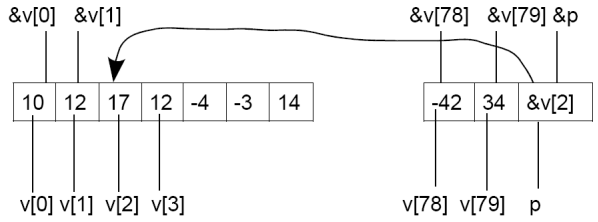
Na linguagem C não existe verificação de índices fora do vetor. Quem deve controlar o uso correto dos índices é o programador. Além disso, o acesso utilizando um índice errado pode ocasionar o acesso de outra variável na memória. No exemplo acima, `v[80]` acessaria a variável `n`. Se o acesso à memória é indevido você recebe a mensagem "segmentation fault".

Vetores e Ponteiros

A implementação de vetores em C está bastante interligada com a de ponteiros visando a facilitar a manipulação de vetores. Considere a seguinte declaração de variáveis:

```
int v[80]; /* declara um vetor de inteiros de nome v com 80 casas */
int *p;
```

que aloca na memória algo do tipo:



Podemos utilizar a sintaxe normal para fazer um ponteiro apontar para uma casa do vetor:

```
p = &v[2]; /* p aponta para a casa de índice 2 de v */
```

Mas podemos utilizar a **sintaxe especial para ponteiros e vetores**, junto com as operações para ponteiros:

- Podemos fazer um ponteiro apontar para o início do vetor `v` fazendo

  ```
  p = v;
  ```

  É a única situação em que o nome do vetor tem sentido sem os colchetes. O comando acima equivale a fazer `p = &v[0];`

- Podemos usar a sintaxe de vetores (`nome_do_vetor[índice]`) com o ponteiro. Assim, se fizermos

```
p = v;
```

  podemos acessar o elemento que está na casa `i` de `v` fazendo `p[i]`, ou seja, ambos `p[i]` e `v[i]` acessam a casa `i` do vetor `v`. Exemplo:

```
i = 3;
p = v; /* p aponta para v[0]. Equivale a fazer p = &v[0] */
p[i] = 4; /* equivale a fazer v[i] = 4 */
```

  Mas se fazemos

```
p = &v[3];
```

  então, `p[0]` é o elemento `v[3]`, `p[1]` é o elemento `v[4]`, `p[2]` é o elemento `v[5]`, e assim por diante.

- Podemos fazer algumas operações com ponteiros. Considere a seguinte declaração:

```
int *p, *q, n, v[50];
float *x, y[20];
```

  1. Quando somamos 1 a um ponteiro para **int** (por exemplo, `p`) ele passa a apontar para o endereço de memória logo após a memória reservada para este inteiro. Exemplo, se `p = &v[4]`, então `p+1` é o endereço de `v[5]`, `p+2` é o endereço de `v[6]`, `p+i` é o endereço de `v[4+i]`.
     Dessa forma, `*p` (vai para onde o `p` está apontando) é o `v[4]`. Portanto, `v[4] = 3` é a mesma coisa que fazer `*p = 3`. Como `p+1` é o endereço de `v[5]`, então `*(p+1)` é `v[5]`. Assim, `v[5] = 10` é a mesma coisa que fazer `*(p+1) = 10`.
  2. Se somamos 1 a um ponteiro para **float** (por exemplo `x`) ele avança para o endereço após este **float**. Por exemplo, se `x=&y[3]`, então `x+1` é o endereço de `y[4]` e `x+i` é o endereço de `y[3+i]`.
  3. Somar ou subtrair um inteiro de um ponteiro:

```
p = &v[22]; q = &v[30];
p = p - 4; q++;
*(p+2) = 3; *q = 4;
```

     Qual índice de `v` recebe 3? Qual índice de `v` recebe 4? [1]
  4. Subtrair dois ponteiros:

```
p = &v[20]; q = &v[31];
n = q - p; /* número inteiro: a diferença entre os índices, neste caso, 11.
```

Vetores como Parâmetro de Funções

Quando se declara uma função que tem como parâmetro um vetor, este vetor é declarado somente com abre e fecha colchetes. Exemplo:

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

[1] As respostas são: `v[20]` recebe 3 e `v[31]` recebe 4.

Esta função recebe um vetor de reais `v` com `n` elementos e devolve o seu módulo via **return**. A declaração acima é equivalente a

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

Na verdade, a declaração no argumento **float** `v[]` é a mesma coisa que **float** `*p`, ou seja, `v` é um ponteiro.

```
1   # include <stdio.h>
2   # include <math.h>
3
4
5   float modulo (float v[], int n) {
6     int i;
7     float r = 0;
8     for (i=0; i<n; i++) {
9       r = r + v[i]*v[i];
10    }
11    r = sqrt (r);
12    return r;
13  }
14
15  int main () {
16    float x[100], comprimento;
17    int m;
18
19    m = 3;
20    x[0] = 2; x[1] = -3, x[2] = 4;
21
22
23    comprimento =  modulo (x, m);
24
25    printf ("Comprimento = %f\n", comprimento);
26
27    return 0;
28  }
```

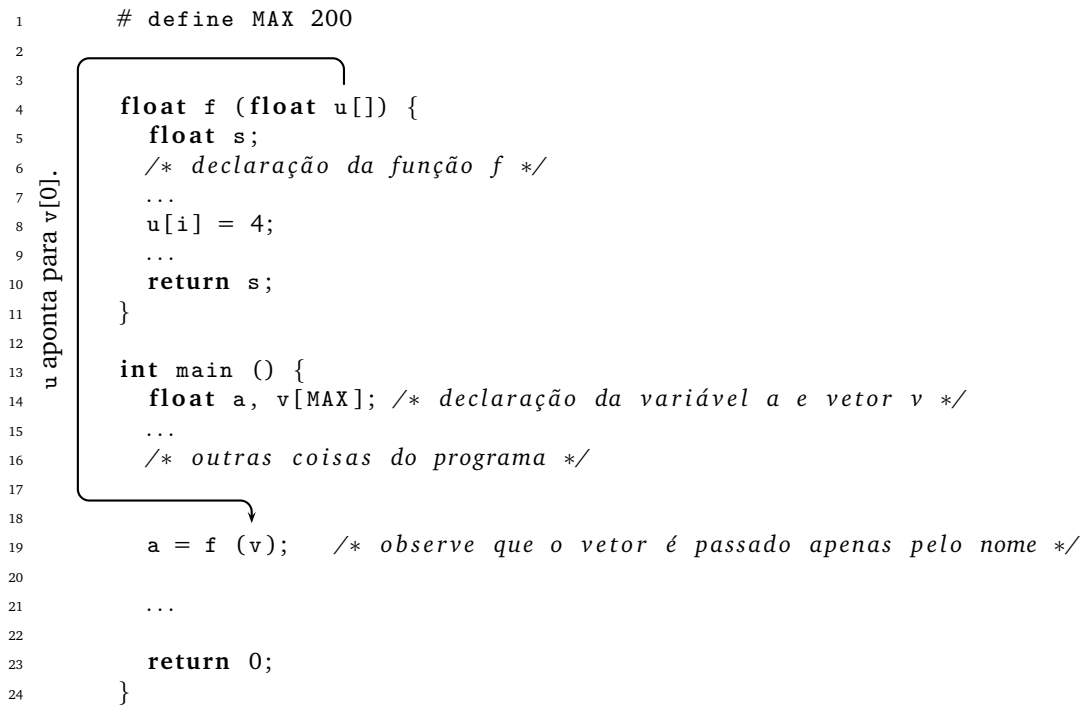
v aponta para x[0]. Então v[i] é x[i]

O parâmetro `v` da função `modulo` aponta para a variável `x[0]` da função `main`. Então `v[i]` na função `modulo` é exatamente `x[i]` da função `main`.

Exemplo de Função com Vetor como Parâmetro

O nome de um vetor dentro de parâmetro de uma função é utilizado como sendo um ponteiro para o primeiro elemento do vetor na hora de utilizar a função.

Exemplo de declaração de funções com vetores como parâmetros:




```
1   # define MAX 200
2
3
4   float f (float u[]) {
5     float s;
6     /* declaração da função f */
7     ...
8     u[i] = 4;
9     ...
10    return s;
11  }
12
13  int main () {
14    float a, v[MAX]; /* declaração da variável a e vetor v */
15    ...
16    /* outras coisas do programa */
17
18
19    a = f (v);   /* observe que o vetor é passado apenas pelo nome */
20
21    ...
22
23    return 0;
24  }
```

Na Linha 19, a chamada da função `f` faz com que o ponteiro `u` receba `&v[0]`, ou seja, faz com que o ponteiro `u` aponte para `v[0]`.

Na Linha 8, temos o comando `u[i] = 4`. Como `u` está apontando para `v[0]`, então o comando `u[i] = 4` é o mesmo que fazer `v[i] = 4`. Assim, na Linha 8, dentro da função `f`, estamos mudando o conteúdo da casa de índice `i` do vetor `v` da função `main`.

Problema

(a) Faça uma função que recebe dois vetores de tamanho n e retorna o seu produto escalar.

O protótipo dessa função seria:

```
float ProdutoEscalar (float u[], float v[], int n);
```

A função recebe como parâmetros os vetores u e v, e um inteiro n. Uma possível solução para esta função seria:

```
float ProdutoEscalar (float u[], float v[], int n) {
  int i;
  float res = 0;
  for (i=0; i<n; i++)
  res = res + u[i] * v[i];
  return res;
}
```

(b) Faça um programa que leia dois vetores reais de tamanho $n < 200$ e verifique se eles são vetores ortogonais. Dois vetores são ortogonais se o produto escalar entre eles é zero. Considere EPS igual a 0.001 o valor do erro para comparar se o produto escalar é zero.

```c
#include <stdio.h>

#define MAX 200
#define EPS 0.001

float ProdutoEscalar (float u[], float v[], int n) {
  int i;
  float res = 0;
  for (i=0; i<n; i++)
  res = res + u[i] * v[i];
  return res;
}

int main () {
  int n, i;
  float a[MAX], b[MAX];
  float prod;

  /* leitura dos vetores */
  printf("Digite o tamanho dos vetores: ");
  scanf("%d", &n);

  printf("Entre com os valores do 1o vetor\n");
  for (i=0; i<n; i++) scanf("%f",&a[i]);

  printf("Entre com os valores do 2o vetor\n");
  for (i=0; i<n; i++) scanf("%f",&b[i]);

  prod = ProdutoEscalar(a, b, n);

  /* cuidado com a comparação com zero usando reais!!! */
  if (prod < EPS && prod > -EPS)
    printf("Os vetores sao ortogonais.\n");
  else
    printf("Os vetores nao sao ortogonais.\n");

  return 0;
}
```

Observe que a função `ProdutoEscalar` não modifica o vetor u nem o vetor v.

Agora, vamos considerar o caso quando temos uma função que deve retornar um vetor.

(c) Faça uma função que recebe dois vetores de tamanho 3 e retorna o seu produto vetorial. O produto vetorial de dois vetores de dimensão três $u = (u_0, u_1, u_2)$ e $v = (v_0, v_1, v_2)$ é dado por $w = (u_1v_2 - u_2v_1, u_2v_0 - u_0v_2, u_0v_1 - u_1v_0)$.

Primeiro como deve ser o protótipo dessa função? Nós sabemos que a função deve receber 2 vetores de entrada e devolver 1 vetor como resultado. Sendo assim, temos o seguinte protótipo:

```c
void ProdutoVetorialTRI (float u[], float v[], float w[]);
```

onde os vetores u e v são entradas e w é o vetor de saída. Uma solução para esta função possível seria:

```c
void ProdutoVetorialTRI (float u[], float v[], float w[]) {
  w[0] = u[1]*v[2] - u[2]*v[1];
  w[1] = u[2]*v[0] - u[0]*v[2];
  w[2] = u[0]*v[1] - u[1]*v[0];
}
```

Observe que a função `ProdutoVetorialTRI` modifica o vetor w. Este vetor w é na verdade um ponteiro para algum vetor da função `main`. E é este vetor que na realidade vai ser modificado.

(d) Faça um programa que leia dois vetores de dimensão três e calcula o seu produto vetorial, e mostra que o produto vetorial é ortogonal aos dois vetores de entrada.

```
#include <stdio.h>

#define MAX 20

float ProdutoEscalar (float u[], float v[], int n) {
  int i;
  float res = 0;
  for (i=0; i<n; i++)
  res = res + u[i] * v[i];
  return res;
}

void ProdutoVetorialTRI (float u[], float v[], float w[]) {
  w[0] = u[1]*v[2] - u[2]*v[1];
  w[1] = u[2]*v[0] - u[0]*v[2];
  w[2] = u[0]*v[1] - u[1]*v[0];
}

int main () {
  int n, i;
  float a[MAX], b[MAX], c[MAX];
  float prod;

  n = 3; /* os vetores têm dimensão 3 */

  printf("Entre com os valores do 1o vetor\n");
  for (i=0; i<n; i++) scanf("%f",&a[i]);

  printf("Entre com os valores do 2o vetor\n");
  for (i=0; i<n; i++) scanf("%f",&b[i]);

  /* Observe a chamada da função ProdutoVetorialTRI    */
  /* O produto vetorial de a e b é colocado no vetor c */
  /* via ponteiro w da função ProdutoVetorialTRI       */
  ProdutoVetorialTRI (a, b, c);

  printf ("Produto vetorial (a x b) = (%.2f, %.2f, %.2f)\n", c[0], c[1], c[2]);

  prod = ProdutoEscalar (a, b, n);
  printf("Produto escalar de a e b: %.2f \n", prod);

  prod = ProdutoEscalar (a, c, n);
  printf("Produto escalar de a e c: %.2f \n", prod);

  prod = ProdutoEscalar (b, c, n);
  printf("Produto escalar de b e c: %.2f \n", prod);

  return 0;
}
```

Observe que a função `ProdutoVetorialTRI` modifica o vetor w. Este vetor w é na verdade um ponteiro para o vetor c da função `main`. E é este vetor que na realidade vai ser modificado, ou seja, o produto vetorial fica armazenado no vetor c.

Outro Problema

(a) Faça uma função que recebe um inteiro $n > 0$ e um vetor de números reais a (que armazena os coeficientes de um polinômio $p(x) = a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n$ de grau n) e devolve a derivada de $p(x)$ no próprio vetor a. Além disso, devolve via **return** o grau do polinômio derivada.

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

(b) Faça uma função que recebe um inteiro $n > 0$, um vetor de números reais a (que armazena os coeficientes de um polinômio $p(x) = a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n$ de grau n) e um real y devolve $p(y)$, ou seja, o valor do polinômio no ponto y.

```
float valor (int n, float a[], float y) {
  float soma = 0, poty = 1;
  int i;
  for (i=0; i<=n; i++) {
    soma = soma + a[i] * poty;
    poty = poty * y;
  }
  return soma;
}
```

(c) Faça um programa que leia um inteiro $m > 0$, os coeficientes reais de um polinômio $p(x) = a_0 + a_1 \cdot x + \ldots + a_m \cdot x^m$ de grau m e um real y e imprime $p'(p(y) - 2) + p''(y + 2)$.

```c
int main () {
  float a[200], y, r, s;
  int m, i;

  printf ("Entre com o grau do polinomio: ");
  scanf ("%d", &m);

  printf ("Entre com os coeficientes do polinomio\n");
  for (i=0; i<=m; i++) {
    printf ("Entre com o coeficiente a[%d] = ");
    scanf ("%f", &a[i]);
  }

  printf ("Entre com o ponto y: ");
  scanf ("%f", &y);

  /* calculando p(y) */
  r = valor (m, a, y);

  /* calculando a derivada de p(x) */
  m = derivada (m, a);

  /* calculando p'(r-2) */
  r = valor (m, a, r-2);

  /* calculando a derivada de p'(x) */
  m = derivada (m, a);

  /* calculando p''(y+2) */
  s = valor (m, a, y+2);

  /* imprimindo resposta final */
  printf ("resposta = %f\n", r+s)

  return 0;
}
```

Observação

É possível também declarar o tamanho `MAX` do vetor nos parâmetros de função, por exemplo, da seguinte forma:

```c
float ProdutoEscalar (float u[MAX], float v[MAX], int n);
void ProdutoVetorialTRI (float u[MAX], float v[MAX], float w[MAX]);
```

Note que o tamanho do vetor é irrelevante na definição da função, no entanto, alguns programadores preferem colocar explicitamente o tamanho `MAX` dos vetores.

Resumo

Vetor quando declarado como parâmetro de função é um **ponteiro**!