MC102 - Algoritmos e Progração de Computador

Prof. Alexandre Xavier Falcão

9° Aula: Vetores

1 Vetores

Até agora, vimos que uma variável simples está associada a uma posição de memória e qualquer referência a ela significa um acesso ao conteúdo de um pedaço de memória, cujo tamanho depende de seu tipo. Nesta aula iremos ver um dos tipos mais simples de estrutura de dados, denominada **vetor**, que nos possibilitará associar um identificador a um conjunto de variáveis simples de um mesmo tipo. Naturalmente, precisaremos de uma sintaxe apropriada para acessar cada variável deste conjunto de forma precisa.

```
int main()
{
  tipo identificador[número de variáveis];
}
```

Antes de iniciarmos, considere que desejamos ler 10 notas de alunos e imprimir as notas acima da média. Note que para executar a tarefa, precisamos calcular a média primeiro e depois comparar cada nota com a média. Portanto, precisamos armazenar cada nota em uma variável. Agora imagine que a turma tem 100 alunos. Como criar 100 variáveis sem tornar o acesso a elas algo complicado de se programar?

Um **vetor** é um conjunto de posições consecutivas de memória, identificadas por um mesmo nome, individualizadas por índices e cujo conteúdo é do mesmo tipo. Assim, o conjunto de 10 notas pode ser associado a apenas um identificador, digamos nota, que passará a identificar não apenas uma única posição de memória, mas 10.

```
int main()
{
  float nota[10]; /* vetor de 10 variáveis do tipo float */
}
```

A referência ao conteúdo da n-ésima variável é indicada pela notação nota$[n-1]$, onde n é uma expressão inteira ou uma variável inteira.

A nota de valor 7.0 na Figura 1, que está na quarta posição da seqüência de notas é obtida como nota[3]. Assim, um programa para resolver o problema acima fica:

```
#include <stdio.h>

int main()
```

```
{
  float nota[10],media=0.0;
  int i;

  printf("Entre com 10 notas\n");
  for (i=0; i < 10; i++) {
    scanf("%f",&nota[i]);
    media += nota[i];
  }
  media /= 10.0;
  printf("media %5.2f\n",media);

  for (i=0; i < 10; i++) {
    if (nota[i] > media)
      printf("nota[%d]=%5.2f\n",i,nota[i]);
  }

  return 0;
}
```

Figura 1: Vetor de 10 notas representado pelo identificador nota.

O identificador é uma variável do tipo apontador, cujo conteúdo é o endereço de memória do primeiro elemento do vetor. Considere, por exemplo, o trecho de código abaixo.

```
int main()
{
  int v[5]={7,2,1,4,10}; // inicialização

  printf("Endereco de v[0]: %d=%d\n",&v[0],v);
  printf("Conteudo de v[0]: %d=%d\n",v[0],*v);
  printf("Conteudo de v[1]: %d=%d\n",v[1],*(v+1));
  printf("Conteudo de v[4]: %d=%d\n",v[4],*(v+4));
  printf("Endereco de v[4]: %d=%d\n",&v[4],(v+4));

  return 0;
}
```

Na memória, os elementos do vetor são armazenados um após o outro. O endereço de $v[i]$ pode ser obtido por $\&v[i]$ ou $(v+i)$, já que v guarda o endereço de $v[0]$ e, portanto, $(v+i)$ guarda o endereço de $v[i]$. Ao somarmos $(v+i)$, pulamos na memória $i * sizeof(int)$ bytes a partir do endereço de $v[0]$.

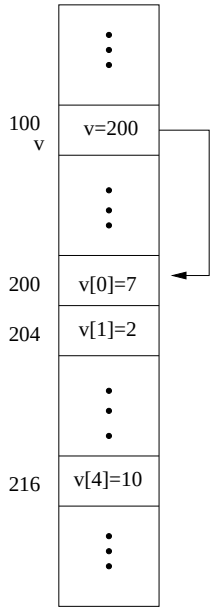

Figura 2: Como o vetor fica armazenado na memória. Os números ao lado indicam o endereço de memória na pilha. As variáveis e os respectivos conteúdos são indicados nas gavetas.

2 Busca em Vetores

Um problema comum quando se manipula vetores é a necessidade de encontrar um elemento com um dado valor. Uma forma trivial de fazer este acesso é percorrer do índice inicial ao índice final todos os elementos do vetor até achar o elemento desejado. Esta forma de busca é chamada linear, pois no pior caso o número de comparações necessárias é igual ao número de elementos no vetor.

2.1 Busca Linear

Suponha, por exemplo, que desejamos saber se existe uma nota x no vetor lido.

```
#include <stdio.h>

int main()
{
  float nota[11],x; /* vetor criado com uma posição a mais */
  int i;

  printf("Entre com 10 notas\n");
  for (i=0; i < 10; i++) {
    scanf("%f",&nota[i]);
  }

  while(1) {
    printf("Digite a nota procurada ou -1 para sair do programa\n");
```

```
    scanf("%f",&x);

    if (x==-1.0)
      break;

    /* busca linear */

    nota[10] = x; /* elemento sentinela */
    i = 0;
    while (nota[i] != x) /* busca com sentinela */
      i++;

    if (i < 10)
      printf("nota %5.2f encontrada na posição %d\n",nota[i],i);
    else
      printf("nota %5.2f não encontrada\n",x);
  }

  return 0;
}
```

Imagine agora que nosso vetor tem tamanho 1024. O que podemos fazer para reduzir o número de comparações? Quanto maior for a quantidade de informação sobre os dados, mais vantagens podemos tirar para agilizar os algoritmos.

2.2 Busca Binária

A busca binária ,por exemplo, reduz o número de comparações de n para $\log_2(n)$ no pior caso, onde n é o tamanho do vetor. Ou seja, um vetor de tamanho $1024 = 2^{10}$ requer no pior caso 10 comparações. No entanto, a busca binária requer que o vetor esteja ordenado. Esta ordenação também tem um custo a ser considerado, mas se vamos fazer várias buscas, este custo pode valer a pena. A idéia básica é que a cada iteração do algoritmo, podemos eliminar a metade dos elementos no processo de busca. Vamos supor, por exemplo, que o vetor de notas está em ordem crescente.

```
#include <stdio.h>

typedef enum {false,true} bool;

int main()
{
  float nota[10],x;
  int i,pos,inicio,fim;
  bool achou;

  printf("Entre com 10 notas em ordem crescente\n");
  for (i=0; i < 10; i++) {
    scanf("%f",&nota[i]);
```

```
  }

  while(1) {
    printf("Digite a nota procurada ou -1 para sair do programa\n");
    scanf("%f",&x);

    if (x==-1.0)
      break;

    /* busca binária */

    inicio = 0;
    fim    = 9;
    achou  = false;

    while ((inicio <= fim)&&(!achou)){
      pos = (inicio+fim)/2;
      if (x < nota[pos])
        fim = pos-1;
      else
        if (x > nota[pos])
          inicio = pos + 1;
        else
          achou = true;
    }

    if (achou)
      printf("nota %5.2f encontrada na posição %d\n",nota[pos],pos);
    else
      printf("nota %5.2f não encontrada\n",x);
  }

  return 0;
}
```

Algoritmos para ordenar vetores serão vistos na próxima aula.