1001350: Implementação de estratégias comuns de uso de vetores

Jander Moreira*

1 Apresentação

Os vetores são tão essenciais à programação quanto a água é para a vida. Este texto mostra vários usos de vetores na linguagem C, trabalhando os seguintes conceitos:

- Declaração de vetores e acesso aos elementos;
- Iniciação dos elementos dos vetores na declaração;
- Uso de vetores de tamanho dinâmico:
  - Declaração com tamanho implícito;
  - Uso parcial do vetor.
- Uso do vetor:
  - Leitura e escrita;
  - Contagens, somas, médias, máximos e mínimos;
  - Substituições;
  - Buscas.

Todos esses conceitos são tratados transversalmente nos diversos exemplos.

Nota importante

É relevante destacar que vários dos exemplos apresentados seriam resolvidos de uma forma mais adequada sem vetores. Entretanto, eles são usados para fins didáticos, procurando exemplificar sua manipulação.

2 Orientações de estudo

Este texto é uma sequência de exemplos, alguns mais simples e diretos, outros um pouco mais sofisticados.

É esperado, para cada exemplo, que o texto seja lido e o programa seja executado. Os códigos fonte ficarão disponíveis. Sugere-se fortemente que se busque entender como cada exemplo funciona e que pequenas modificações, a título de teste, sejam feitas.

3 Usos e aplicações de vetores

Esta seção apresentas os vários usos, com os comentários pertinentes.

3.1 Leitura e escrita

O básico do uso de vetores é saber ler seus valores e apresentar seu conteúdo.

Uma leitura de um elemento de um arranjo usa o mesmo formato da leitura do tipo base, recebendo sempre a referência (endereço) do argumento. Assim, se for `int` o formato é `%d` ou `%i`, se for `float` usa `%f` ou se for `double` é `%lf`, por exemplo. A regra para a escrita com o `printf` é equivalente[1].

O Exemplo 1 mostra os comandos básicos para ler um vetor de `double`s e, logo em seguida, escrever os valores. O código equivalente para leitura de um vetor de `double`s está no Exemplo 2.

Exemplo 1

A leitura e escrita do vetor usam o formato `%d` para formatar a entrada e saída do tipo `int`.

```c
/*
  Leitura e escrita de um vetor com tipo
    base int
  Input: 10 valores inteiros
  Output: Os mesmos 10 valores
*/
#include <stdio.h>

#define TAM_VETOR 10
int main(void){
  int vetor[TAM_VETOR];

  // leitura
  for(int i = 0; i < TAM_VETOR; i++){
    printf("vetor[%2d] = ", i);
    scanf("%d", &vetor[i]);
  }
  printf("\n");

  // escrita
  printf("vetor = [ ");
  for(int i = 0; i < TAM_VETOR; i++)
    printf("%d ", vetor[i]);
  printf("]\n");

  return 0;
}
```

Arquivo: codigo/leitura_escrita_int.c.

Exemplo 2

Os elementos são lidos com `%lf`, que é a especificação para o tipo `double` e escritos com `%g`, que se aplica a ▷

*Jander Moreira – Universidade Federal de São Carlos – Departamento de Computação – *Rodovia Washington Luis, km 235 – 13565-905 - São Carlos/SP – Brasil* – jander@dc.ufscar.br

[1] Veja https://en.wikipedia.org/wiki/Scanf_format_string e https://en.wikipedia.org/wiki/Printf_format_string

valores reais e escolhe a melhor forma de apresentação dependendo do valor.

```c
/*
  Leitura e escrita de um vetor com tipo
    base double
  Input: 10 valores reais
  Output: Os mesmos 10 valores
*/
#include <stdio.h>

#define TAM_VETOR 10
int main(void){
  double vetor[TAM_VETOR];

  // leitura
  for(int i = 0; i < TAM_VETOR; i++){
    printf("vetor[%2d] = ", i);
    scanf("%lf", &vetor[i]);
  }
  printf("\n");

  // escrita
  printf("vetor = [ ");
  for(int i = 0; i < TAM_VETOR; i++)
    printf("%g ", vetor[i]);
  printf("]\n");

  return 0;
}
```

Arquivo: codigo/leitura_escrita_double.c.

3.2 Contagens, somas, médias, máximos e mínimos

Nesta seção são apresentados exemplos de varredura dos dados do vetor.

O primeiro programa (Exemplo 3) exemplifica algumas contagens em um vetor de inteiros que varrem todos os elementos do vetor, enquanto no Exemplo 4 a varredura é parcial.

Exemplo 3

O vetor é fixo e possui dados apenas para exemplificar as contagens. São realizadas três contagens diferentes, cada uma delas feita separadamente.
Naturalmente, todas as contagens poderiam ser feitas simultaneamente em uma única repetição.

```c
/*
  Apresentacao de contagens feitas em um
    vetor
  Input: Nao ha
  Output: As quantidades de valores
    negativos, de presentes no intervalo [0,
     10] e de nulos
*/
#include <stdio.h>

int main(void){
  int dados[] =
    {18, 7, 4, -5, 12, 18, 97, -1, 33, -7,
      9, 38, 0, -43, 0, 0, 13, 847, -9, 12};
  int num_elementos = sizeof dados/sizeof
    dados[0];

  // contagem de negativos
  int quantidade_negativos = 0;
  for(int i = 0; i < num_elementos; i++)
    if(dados[i] < 0)
      quantidade_negativos++;
  printf("%d valores negativos\n",
    quantidade_negativos);

  // contagem no intervalo [0, 10]
  int quantidade_0a10 = 0;
  for(int i = 0; i < num_elementos; i++)
    if(0 < dados[i] && dados[i] < 10)
      quantidade_0a10++;
  printf("%d valores em [0, 10]\n",
    quantidade_0a10);

  // contagem de nulos
  int quantidade_nulos = 0;
  for(int i = 0; i < num_elementos; i++)
    if(dados[i] == 0)
      quantidade_nulos++;
  printf("%d valores negativos\n",
    quantidade_nulos);

  return 0;
}
```

Arquivo: codigo/contagens.c.

Exemplo 4

A contagem da quantidade de valores que precedem o primeiro zero é apresentada. Na ausência de zeros, a quantidade de elementos no vetor é a saída.
Vale notar que a verificação de término do vetor (`i < num_elementos`) precede a verificação do valor (`dados[i] < 0`), uma vez que `i` pode ser um índice inválido.

```c
/*
  Contagem da quantidade de dados ate o
    primeiro valor zero
  Input: Nao ha
  Output: A quantidade de valores não nulos
    precedentes ao primeiro valor zero ou a
    quantidade de valores no arranjo caso
    nenhum nulo esteja presente
*/
#include <stdio.h>

int main(void){
  int dados[] =
    {18, 7, 4, -5, 12, 18, 97, -1, 33, -7,
      9, 38, 0, -43, 0, 0, 13, 847, -9, 12};
  int num_elementos = sizeof dados/sizeof
    dados[0];

  // contagem ate o primeiro zero
  int quantidade_nao_nulos = 0;
  int i = 0;
  while(i < num_elementos && dados[i] != 0){
    quantidade_nao_nulos++;
    i++;
  }
  printf("%d valores ate o primeiro nulo\n",
    quantidade_nao_nulos);

  return 0;
}
```

Arquivo: codigo/contagem_ate_zero.c.

O soma de valores é apresentada pelo programa do Exemplo 5, usando-se neste caso o controle dinâmico do tamanho do vetor.

Exemplo 5

Soma dos valores em um vetor. A quantidade de valores válidos é dada pela variável `num_elementos`, que não pode ser maior que `TAMANHO_VETOR`. A função `assert`, definida em `assert.h` termina o programa caso a condição da asserção seja violada.
A leitura deve ser finalizada digitando-se `Ctlr-D` no Linux ou `Ctrl-Z` e `ENTER` no Windows.

```
/*
  Soma de uma sequencia de valores reais
  Input: Uma sequencia de valores reais
  Output: A soma de todos os valores
*/
#include <stdio.h>
#include <assert.h>

#define TAMANHO_VETOR 100 // máximo
int main(void){
  double dados[TAMANHO_VETOR];

  // entrada de dados
  printf("Digite um valor por linha");
  #ifdef linux
    printf(" (ou Ctrl-D para terminar):\n");
  #else
    printf(" (ou Ctrl-Z e ENTER "
      "para terminar):\n");
  #endif
  int quantidade = 0;
  double elemento;
  while(scanf("%lf", &elemento) != EOF){
    assert(quantidade < TAMANHO_VETOR);
    dados[quantidade] = elemento;
    quantidade++;
  }

  // soma
  double soma = 0;
  for(int i = 0; i < quantidade; i++)
    soma += dados[i];

  if(quantidade > 0)
    printf("\nSoma = %g\n", soma);

  return 0;
}
```

Arquivo: codigo/soma.c.

Baseado no Exemplo 5, o programa da média é apresentado no Exemplo 6.

Exemplo 6

A média é calculada mediante a soma dos valores de entrada dividida pela quantidade de itens. Todos os comentários do Exemplo 5 são válidos aqui.

```
/*
  Media de uma sequencia de valores reais
  Input: Uma sequencia de valores reais
  Output: A soma de todos os valores
*/
#include <stdio.h>
#include <assert.h>

#define TAMANHO_VETOR 100 // máximo
int main(void){
  double dados[TAMANHO_VETOR];

  // entrada de dados
  printf("Digite um valor por linha");
  #ifdef linux
    printf(" (ou Ctrl-D para terminar):\n");
  #else
    printf(" (ou Ctrl-Z e ENTER "
      "para terminar):\n");
  #endif
  int quantidade = 0;
  double elemento;
  while(scanf("%lf", &elemento) != EOF){
    assert(quantidade < TAMANHO_VETOR);
    dados[quantidade] = elemento;
    quantidade++;
  }

  // soma
  double soma = 0;
  for(int i = 0; i < quantidade; i++)
    soma += dados[i];

  if(quantidade > 0)
    printf("\nMedia = %g\n",
      soma/quantidade);

  return 0;
}
```

Arquivo: codigo/media.c.

3.3 Substituições

Nesta seção serão exemplificadas algumas manipulações que modificam os valores do vetor segundo algum critério.

No Exemplo 7 é apresentada uma substituição simples, na qual todos os valores negativos são trocados por zero.

Exemplo 7

A substituição de valores negativos por zero é feita pela varredura do vetor, comparando cada valor individual e fazendo-se a substituição se necessária.

```
/*
  Remocao dos valores negativos de um vetor,
     substituindo-os por zero
  Input: Uma sequencia de valores reais
  Output: Uma sequencia igual a de entrada,
    exceto pelos valores negativos terem
    sido trocados por zero
*/
#include <stdio.h>
#include <assert.h>

#define TAMANHO_VETOR 100 // máximo
int main(void){
  double dados[TAMANHO_VETOR];

  // entrada de dados
  printf("Digite um valor por linha");
  #ifdef linux
    printf(" (ou Ctrl-D para terminar):\n");
  #else
    printf(" (ou Ctrl-Z e ENTER "
      "para terminar):\n");
```

```
  #endif
  int quantidade = 0;
  double elemento;
  while(scanf("%lf", &elemento) != EOF){
    assert(quantidade < TAMANHO_VETOR);
    dados[quantidade] = elemento;
    quantidade++;
  }

  // troca dos negativos por zero
  for(int i = 0; i < quantidade; i++)
    if(dados[i] < 0)
      dados[i] = 0;

  // apresentacao da sequencia modificada
  for(int i = 0; i < quantidade; i++)
    printf("%10g\n", dados[i]);

  return 0;
}
```

Arquivo: codigo/substituicao_negativos.c.

A substituição de valores negativos por zero é trocada pela remoção efetiva dos elementos negativos no programa do Exemplo 8. Neste caso, o tamanho (i.e., seu tamanho lógico) pode ser reduzido.

Exemplo 8

O vetor é varrido linearmente e a cada valor negativo encontrado, todos os elementos subsequentes são movidos, reduzindo-se seu tamanho lógico.
O código implementa o algoritmo seguinte.

Entrada: Uma sequência de valores reais
Saída: : Uma sequencia baseada na de entrada eliminados os valores negativos

Crie um vetor *dados* com os dados de entrada e defina *quantidade* com o número de valores do vetor
Inicie o índice i com zero
enquanto $i <$ *quantidade* **faça**
  se *dados*[i] for negativo **então**
    Desloque os valores das posições de $i+1$ até *quantidade* -1 respectivamente para as posições de i até *tamanho* -2
    Reduza o tamanho do vetor eliminando a última posição e atualizando *quantidade*
  senão
    $i \leftarrow i+1$
  fim se
fim enquanto
Apresente *dados*

```
/*
  Remocao dos valores negativos de um vetor
    com preservacao da ordem relativa e
    reducao do tamanho do vetor
  Input: Uma sequencia de valores reais
  Output: Uma sequencia baseada na de
    entrada eliminados os valores negativos
*/
#include <stdio.h>
#include <assert.h>

#define TAMANHO_VETOR 100 // máximo
int main(void){
  double dados[TAMANHO_VETOR];

  // entrada de dados
  printf("Digite um valor por linha");
  #ifdef linux
    printf(" (ou Ctrl-D para terminar):\n");
  #else
    printf(" (ou Ctrl-Z e ENTER "
      "para terminar):\n");
  #endif
  int quantidade = 0;
  double elemento;
  while(scanf("%lf", &elemento) != EOF){
    assert(quantidade < TAMANHO_VETOR);
    dados[quantidade] = elemento;
    quantidade++;
  }

  // remocao dos negativos
  int i = 0;
  while(i < quantidade)
    if(dados[i] < 0){
      // reposicionamento
      for(int j = i + 1; j < quantidade; j
        ++)
        dados[j - 1] = dados[j];
      quantidade--;
    }
    else
      i++;

  // apresentacao da sequencia modificada
  for(int i = 0; i < quantidade; i++)
    printf("%g ", dados[i]);
  printf("\n");

  return 0;
}
```

Arquivo: codigo/remocao_negativos.c.

Uma permuta de elementos é apresentada no Exemplo 9, ilustrando trocas de itens de posição, preservando todos os valores originais.

Exemplo 9

A posição em que cada par será inserido é controlada por `posicao_par`. Quando um valor par é localizado, é feita a troca dos valores de posição e o `posicao_par` é atualizada. Embora a ordem relativa entre os pares seja mantida, isso não ocorre com os ímpares.

```
/*
  Dada uma coleção de valores inteiros,
    apresentar todos os pares seguidos de
    todos os impares
  Input: Uma sequencia de valores inteiros
  Output: A sequencia de entrada com os
    elementos permutados de forma a todos os
     pares seram apresentados antes dos
    impares, sem preservar a ordem relativa
    entre os elementos

  Definicao de par: todo  inteiro n que, se
    dividido por dois, tem resto zero.
  Definicao de impar: todo inteiro que nao
    seja par.
*/
#include <stdio.h>
```

```c
#include <assert.h>

#define TAMANHO_VETOR 100 // máximo

// prototipo
int e_par(int);

int main(void){
  int dados[TAMANHO_VETOR];

  // entrada de dados
  printf("Digite um valor por linha");
  #ifdef linux
    printf(" (ou Ctrl-D para terminar):\n");
  #else
    printf(" (ou Ctrl-Z e ENTER "
      "para terminar):\n");
  #endif
  int quantidade = 0;
  int elemento;
  while(scanf("%d", &elemento) != EOF){
    assert(quantidade < TAMANHO_VETOR);
    dados[quantidade] = elemento;
    quantidade++;
  }

  // separacao dos pares dos impares
  int posicao_par = 0;
  for(int indice = 0; indice < quantidade;
    indice++)
    if(e_par(dados[indice])){
      int temp = dados[posicao_par];
      dados[posicao_par] = dados[indice];
      dados[indice] = temp;
      posicao_par++;
    }

  // apresentacao dos dados rearranjados
  for(int i = 0; i < quantidade; i++)
    printf("%d ", dados[i]);
  printf("\n");

  return 0;
}


/*
  e_par: retorna verdadeiro se for par;
    falso caso constrario
  Input: um valor inteiro n como parametro
  Output: 1 (verdadeiro) se for par ou 0 (
    falso) se for impar como valor de
    retorno
*/
int e_par(int n){
  return n % 2 == 0;
}
```

Arquivo: codigo/pares_impares.c.

3.4 Buscas

As buscas por dados no vetor pode ter vários formatos. Por exemplo, pode-se querer apenas saber se um dado está ou não presente ou então retornando a posição onde foi localizado.

Os Exemplos 10 e 11 são programas que determinam ausência ou presença de um valor no vetor, fornecendo um resultado "lógico" (ou seja, presente ou ausente).

Exemplo 10

A busca é feita com o auxílio de uma variável lógica (`presente`), que termina com VERDADEIRO ou FALSO. Assume-se a ausência do valor antes do início da busca (`presente = 0`), alterando-se esse valor para verdadeiro (`presente = 1`) caso o valor seja localizado. Caso o item não esteja presente, a repetição termina ao chegar ao fim do vetor e o valor FALSO permanece. O uso do `while` permite que, assim que o valor seja localizado, o laço de repetição já seja interrompido, sem fazer verificações desnecessárias.

```c
/*
  Busca por um dado valor em um vetor
  Input: O valor inteiro a ser buscado no
    vetor
  Output: A mensagem "presente" se o valor
    for localizado ou "ausente" caso
    contrario
*/
#include <stdio.h>

int main(void){
  int dados[] =
    {18, 7, 4, -5, 12, 18, 97, -1, 33, -7,
      9, 38, 0, -43, 0, 0, 13, 847, -9, 12};
  int num_elementos = sizeof dados/sizeof
    dados[0];

  // obtencao do valor a ser buscado
  int valor_busca;
  printf("Valor de busca: ");
  scanf("%d", &valor_busca);

  // busca no vetor
  int presente = 0; // nao presente
  int i = 0;
  while(i < num_elementos && !presente){
    if(dados[i] == valor_busca)
      presente = 1; // achou
    i++;
  }
  if(presente)
    printf("presente\n");
  else
    printf("ausente\n");

  return 0;
}
```

Arquivo: codigo/busca_v1.c.

Exemplo 11

```c
/*
  Busca por um dado valor em um vetor
  Input: O valor inteiro a ser buscado no
    vetor
  Output: A mensagem "presente" se o valor
    for localizado ou "ausente" caso
    contrario
*/
#include <stdio.h>

int main(void){
  int dados[] =
    {18, 7, 4, -5, 12, 18, 97, -1, 33, -7,
      9, 38, 0, -43, 0, 0, 13, 847, -9, 12};
  int num_elementos = sizeof dados/sizeof
    dados[0];
```

▷

```
  // obtencao do valor a ser buscado
  int valor_busca;
  printf("Valor de busca: ");
  scanf("%d", &valor_busca);

  // busca no vetor
  int i = 0;
  while(i < num_elementos && dados[i] !=
    valor_busca)
    i++;
  if(i < num_elementos)
    printf("presente\n");
  else
    printf("ausente\n");

  return 0;
}
```

Arquivo: codigo/busca_v2.c.

A busca é feita usando-se um `while` que termina em duas condições: chegando ao final do vetor ou localizando o elemento procurado.
A decisão se localizou ou não é feita verificando-se se o índice `i` está dentro do intervalo válido para o vetor (caso em que achou) ou se passou do final (isto é, o valor está ausente).

A estratégia de determinar a presença ou ausência de um valor no vetor, muitas vezes, pode ser feita contando o número de ocorrências. Neste caso, se a contagem for igual a zero, está ausente, caso contrário está presente.

Entretanto, esta estratégia gasta mais tempo, uma vez que o vetor é percorrido até o fim todas as vezes, mesmo quando o valor a ser localizado já está presente nas primeiras posições. Assim, terminar antes do final quando possível é a melhor abordagem.

3.5 Tamanho dinâmico

Uma forma de se ter um vetor com tamanho dinâmico é definindo-se um vetor de tamanho fixo, porém usando-o parcialmente. Dessa forma, pode-se criar um vetor com 1000 posições (fixo) e usar somente a quantidade necessária: de zero até 1000. Nesta abordagem, há um tamanho máximo para o vetor que deve ser considerado.

A criação de um vetor ordenado é apresentada no Exemplo 12. A ideia é acrescentar um novo valor ao vetor, porém mantendo os dados em ordem não decrescente.

Exemplo 12

O vetor é criado vazio, o que é indicado por `num_elementos` igual a zero. Cada novo item inserido faz com que todos os valores maiores que ele sejam deslocados em direção ao final do arranjo, de forma a abrir espaço para o item na posição correta. O programa implementa o algoritmo seguinte.

Entrada: Uma sequência de valores reais
Saída: A sequência de entrada com seus elementos permutados de forma que estejam em ordem não decrescente

▷

Inicie o vetor *dados* com *num_ elementos* igual a zero
para cada valor *novo_ item* da entrada **faça**
  Mova todos os elementos maiores que *novo_ item* uma posição em direção ao fim do vetor, aumentando o vetor em um elemento
  Atualize *num_ elementos*
  Atribua o *novo_ item* à posição anterior ao último item deslocado
fim para

Na implementação, o vetor é escrito depois de cada inserção para ilustrar a situação atual do vetor.

```
/*
Criacao de um vetor mantendo os dados em
  ordem nao decrescente
  Input: Uma sequencia de valores reais
  Output: A sequencia de entrada com seus
    elementos permutados de forma que
    estejam em ordem nao decrescente
*/
#include <stdio.h>
#include <assert.h>

#define TAMANHO_VETOR 100 // máximo
int main(void){
  double dados[TAMANHO_VETOR];

  // entrada de dados
  printf("Digite um valor por linha");
  #ifdef linux
    printf(" (ou Ctrl-D para terminar):\n");
  #else
    printf(" (ou Ctrl-Z e ENTER "
      "para terminar):\n");
  #endif
  int num_elementos = 0; // vetor vazio
  double novo_item;
  printf("Novo item: ");
  while(scanf("%lf", &novo_item) != EOF){
    assert(num_elementos < TAMANHO_VETOR);

    // desloca maiores que novo_item
    int i = num_elementos - 1;
    while(i >= 0 && dados[i] > novo_item){
      dados[i + 1] = dados[i];
      i--;
    }

    // coloca novo_item no lugar certo
    dados[i + 1] = novo_item;
    num_elementos++;

    // apresentacao da sequencia modificada
    for(int i = 0; i < num_elementos; i++)
      printf("%g ", dados[i]);
    printf("\n\nNovo item: ");
  }
  printf("\n");

  return 0;
}
```

Arquivo: codigo/insercao_ordenada.c.

Revisão: 8 de setembro de 2020
Dia Internacional da Alfabetização.