AULA TEÓRICA+PRÁTICA 14

FUNÇÕES E VETORES

Nas aulas práticas e teóricas vimos funções, argumentos de funções e passagem de parâmetros para funções na linguagem C, onde trabalhamos apenas com argumentos de tipos básicos. Observe que chegamos a passar o valor de um elemento de um vetor como um argumento para uma função, por cópia e por referência. Nesta aula veremos como realizar passagem de parâmetros de tipos complexos, mais especificamente de parâmetros que são variáveis compostas homogêneas unidimensionais ou vetores. Esta aula é baseada nas referências [9, 6].

14.1 Vetores como argumentos de funções

Assim como fizemos com variáveis de tipos básicos, é possível passar um vetor todo como argumento em uma chamada de uma função. O parâmetro correspondente deve ser um vetor com a mesma dimensão. Vejamos um exemplo no programa 14.1.

Programa 14.1: Um programa com uma função que tem um vetor como parâmetro.

```
#include <stdio.h>

/* Recebe um vetor com 10 números inteiros e devolve o menor deles */
int minimo(int A[10])
{
   int i, min;

   min = A[0];
   for (i = 1; i < 10; i++)
      if (A[i] < min)
         min = A[i];
   return min;
}

/* Recebe 10 números inteiros e mostra o menor deles na saída */
int main(void)
{
   int i, vet[10];

   for (i = 0; i < 10; i++)
      scanf("%d", &vet[i]);
   printf("O menor valor do conjunto é %d\n", minimo(vet));
   return 0;
}
```

Observe a interface da função `minimo` do programa acima. Essa interface indica que a função devolve um valor do tipo inteiro, tem identificador `minimo` e tem como seu argumento um vetor contendo 10 elementos do tipo inteiro. Referências feitas ao parâmetro `A`, no interior da função `minimo`, acessam os elementos apropriados dentro do vetor que foi passado à função. Para passar um vetor inteiro para uma função é necessário apenas descrever o identificador do vetor, sem qualquer referência a índices, na chamada da função. Essa situação pode ser visualizada na última linha do programa principal, na chamada da função `minimo(pontos)`.

É importante destacar também que a única restrição de devolução de uma função é relativa às variáveis compostas homogêneas. De fato, um valor armazenado em uma variável de qualquer tipo pode ser devolvido por uma função, excluindo variáveis compostas homogêneas.

14.2 Vetores são parâmetros passados por referência

Nesta aula, o que temos visto até o momento são detalhes de como passar vetores como parâmetros para funções, detalhes na chamada e na interface de uma função que usa vetores como parâmetros. No entanto, esses elementos por si só não são a principal peculiaridade dos vetores neste contexto. Vejamos o programa 14.2.

Programa 14.2: Um vetor é sempre um parâmetro passado por referência para uma função.

```
#include <stdio.h>

/* Recebe um vetor com n números inteiros e
   devolve o vetor com seus valores dobrados */
void dobro(int vetor[100], int n)
{
   int i;
   for (i = 0; i < n; i++)
      vetor[i] = vetor[i] * 2;
}

/* Recebe n números inteiros, com 1 <= n <= 100, e mos-
   tra o dobro de cada um desses valores informados */
int main(void)
{
   int i, n, valor[100];

   printf("Informe a quantidade de elementos: ");
   scanf("%d", &n);
   for (i = 0; i < n; i++) {
      printf("Informe o %d-ésimo valor: ", i+1);
      scanf("%d", &valor[i]);
   }
   dobro(valor, n);
   printf("Resultado: ");
   for (i = 0; i < n; i++)
      printf("%d ", valor[i]);
   printf("\n");

   return 0;
}
```

Perceba que a função `dobro` do programa acima modifica os valores do vetor `valor`, uma variável local do programa principal que é passada como parâmetro para a função `dobro` na variável `vetor`, uma variável local da função `dobro`. É importante observar que, quando usamos vetores como argumentos, uma função que modifica o valor de um elemento do vetor, modifica também o vetor original que foi passado como parâmetro para a função. Esta modificação tem efeito mesmo após o término da execução da função.

Nesse sentido, podemos dizer que um vetor, uma matriz ou uma variável composta homogênea de qualquer dimensão é *sempre* um parâmetro de entrada e saída, isto é, é sempre passado por referência a uma função.

No entanto, lembre-se que a modificação de elementos de um vetor em uma função se aplica somente quando o vetor completo é passado como parâmetro à função e não elementos individuais do vetor. Esse último caso já foi tratado em aulas anteriores.

14.3 Vetores como parâmetros com dimensões omitidas

Na definição de uma função, quando um de seus parâmetros é um vetor, seu tamanho pode ser deixado sem especificação. Essa prática de programação é freqüente e permite que a função torne-se ainda mais geral. Dessa forma, a função `dobro`, por exemplo, definida na seção anterior, poderia ser reescrita como a seguir:

```
/* Recebe um vetor com n números inteiros e
   devolve o vetor com seus valores dobrados */
void dobro(int vetor[], int n)
{
   int i;

   for (i = 0; i < n; i++)
      vetor[i] = vetor[i] * 2;
   return;
}
```

Observe que a dimensão do parâmetro `vetor` da função `dobro`, que é um vetor, foi omitida. A duas definições da função `dobro`, nesta seção e na seção anterior, são equivalentes. No entanto, a definição com a dimensão omitida permite que a função `dobro` possa ser chamada com vetores de quaisquer dimensões como argumentos.

Por exemplo, se temos dois vetores de inteiros A e B, com dimensões 50 e 200, respectivamente, a função `dobro` pode ser chamada para computar o dobro de cada coordenada do vetor A e, do mesmo modo, também pode se chamada para computar o dobro de cada coordenada do vetor B. Ou seja, a função `dobro` pode ser chamada com vetores de dimensões diferentes como parâmetros.

```
dobro(A, 50);
dobro(B, 200);
```

Exercícios

14.1 (a) Escreva uma função com a seguinte interface:

```
int subconjunto(int A[MAX], int m, int B[MAX], int n)
```

que receba como parâmetros um vetor A de números inteiros com m elementos e um vetor B de números inteiros com n elementos, ambos representando conjuntos, e verifica se A está contido em B ($A \subset B$).

(b) Escreva um programa que receba dois vetores de números inteiros U e W, com u e w elementos respectivamente, $1 \leqslant u, w \leqslant 100$, e verifique se os dois conjuntos são iguais ($U = W$ se e somente se $U \subset W$ e $W \subset U$). Use a função do item (a).

Programa 14.3: Solução do exercício 14.1.

```
#include <stdio.h>

#define MAX        100
#define VERDADEIRO 1
#define FALSO      0

/* Recebe um vetor A com m números inteiros e um vetor B com
   n números inteiros e verifica se A está contido em B */
int subconjunto(int A[MAX], int m, int B[MAX], int n)
{
   int i, j, contido;
   contido = VERDADEIRO;
   for (i = 0; i < m && contido; i++) {
      for (j = 0; j < n && B[j] != A[i]; j++)
         ;
      if (j == n)
         contido = FALSO;
   }
   return contido;
}

/* Recebe dois conjuntos de números inteiros e verifica se são iguais */
int main(void)
{
   int tamU, i, U[MAX], tamW, j, W[MAX];
   scanf("%d", &tamU);
   for (i = 0; i < tamU; i++)
      scanf("%d", &U[i]);
   scanf("%d", &tamW);
   for (j = 0; j < tamW; j++)
      scanf("%d", &W[j]);
   if (subconjunto(U, tamU, W, tamW) && subconjunto(W, tamW, U, tamU))
      printf("U == W\n");
   else
      printf("U != W\n");
   return 0;
}
```

14.2 Em um programa na linguagem C, um conjunto pode ser representado por um vetor da seguinte forma: `V[0]` contém o número de elementos do conjunto; `V[1], V[2], ...` são os elementos do conjunto, sem repetições.

(a) Escreva uma função com a seguinte interface:

```
void intersec(int A[MAX+1], int B[MAX+1], int C[MAX+1])
```

que dados dois conjuntos de números inteiros A e B, construa um terceiro conjunto C tal que $C = A \cap B$. Lembre-se de que em `C[0]` a sua função deve colocar o tamanho da intersecção.

(b) Escreva um programa que leia um número inteiro $n \geqslant 2$ e uma seqüência de n conjuntos de números inteiros, cada um com no máximo 100 elementos, e construa e imprima um vetor que representa a intersecção dos n conjuntos.

Observação: não leia todos os conjuntos de uma só vez. Leia os dois primeiros conjuntos e calcule a primeira intersecção. Depois, leia o próximo conjunto e calcule uma nova intersecção entre esse conjunto lido e o conjunto da intersecção anterior, e assim por diante.

14.3 (a) Escreva uma função com a seguinte interface:

```
void ordena_insercao(int A[MAX], int m)
```

que receba um vetor A de m números inteiros, com $1 \leqslant m \leqslant 100$, e ordene os elementos desse vetor em ordem crescente usando o método da inserção.

(b) Escreva uma função com a seguinte interface:

```
void intercala(int A[MAX], int m, int B[MAX], int n,
               int C[2*MAX], int *k)
```

que receba um vetor A de números inteiros em ordem crescente de dimensão m e um vetor B de números inteiros em ordem crescente de dimensão m e compute um vetor C contendo os elementos de A e de B sem repetição e em ordem crescente.

(c) Escreva um programa que receba dois conjuntos de números inteiros e distintos X e Y, com no máximo 100 elementos, ordene cada um deles usando a função do item (a) e intercale esses dois vetores usando a função do item (b), obtendo como resultado um vetor de números inteiros em ordem crescente.