LINGUAGEM C++
VARIÁVEIS COMPOSTAS – *Arrays – Aula I*

Prof. Bruno E. G. Gomes

Uma variável em um algoritmo pode ser vista como uma gaveta. A declaração de uma variável reserva uma gaveta (posição) de um certo tipo em um arquivo (a memória do computador). Toda gaveta deve ter um nome, assim é possível identificá-la unicamente. A inicialização atribui um certo valor à gaveta, ou seja, à variável. Dessa forma, ela pode ser referenciada de alguma forma no decorrer do algoritmo.

Até o momento, estudou-se o armazenamento e recuperação de informações em variáveis de tipos primitivos (*int, bool, char, float* e *double*). É intuitivo pensar que o armazenamento apenas em variáveis e constantes simples não é suficiente para se resolver todo tipo de problema que se possa expressar por meio de algoritmos. Por exemplo, se for necessário representar o armazenamento das notas de 60 alunos em uma turma, seria necessário criar 60 variáveis, uma para cada aluno.

Nesta aula, tem início a construção de novos tipos a partir dos tipos básicos (primitivos). Esses novos tipos são denominados de estruturas de dados, que definem como os tipos primitivos serão organizados para se formar o novo tipo.

Iremos introduzir as variáveis compostas homogêneas (de mesmo tipo), que compreendem os vetores, matrizes e cadeias de caracteres e as variáveis compostas heterogêneas (de tipos distintos), definidas por meio de registros e classes.

Vetores – Arrays de um índice

É possível tratar uma variável como um elemento e um *array* como um conjunto. Quando uma *array* é composto por elementos (variáveis) de um mesmo tipo primitivo, temos um conjunto homogêneo de dados, ao qual damos o nome de *vetor.*

Um vetor é composto por uma sequência de valores do mesmo tipo. Cada valor que ele armazena pode ser acessado através do nome da variável mais a posição (índice) do valor que se quer acessar.

Voltando à analogia da memória como uma série de gavetas, com variáveis simples, uma gaveta pode armazenar apenas um dado. Uma variável composta pode ser vista como uma gaveta que pode armazenar uma quantidade sequencial e finita de dados do mesmo tipo, referenciados pelo mesmo nome.

VETORES

Uma variável composta de uma dimensão é denominada de **vetor**. Abaixo, vê-se a declaração de um vetor:

<tipo> <identificador>[<numero_de_elementos>];

Onde:

<tipo> - Qualquer tipo de dados válido na linguagem;

<identificador> - Nome da variável vetor;

<número_de_elementos> - Representa a quantidade de elementos que o vetor pode ter. Esta quantidade é definida na declaração do vetor e não pode ser modificada posteriormente no programa.

Obs. 1: Após declaração, ficam reservadas na memória do computador a quantidade de posições correspondentes ao tamanho do vetor. Se o vetor não for inicializado na declaração ou, posteriormente, por atribuição ou leitura (*cin*), então esses valores são indeterminados, podendo ser qualquer lixo que esteja armazenado na memória da máquina naquele momento.

Obs. 2: Não é obrigado inicializar todas as posições de um vetor, apenas as que forem necessárias ao seu uso. Porém, deve-se lembrar que o tamanho do vetor é fixo, e que não é correto tentar acessar um elemento que esteja fora desta quantidade (menor que 0 e maior que *tamanho - 1*).

Exemplos de declaração

```
float notas[30]; //declara um vetor de reais de tamanho 30.
float vetor[20]; //declara um vetor de reais de tamanho 20.
int pares[25], impares[10];  //declara dois vetores de inteiros.
bool resultados[8]; //declara um vetor de booleanos (true ou false).
char nome[80]; /*declara um vetor para armazenar um nome de até 80
      caracteres.*/
```

OPERAÇÕES COM VETORES

Não é possível operar diretamente com um vetor da mesma forma que com variáveis simples. P. ex: pares = 10 (ERRADO!). As operações devem ser feitas com cada elemento do vetor. O acesso individual a cada elemento é feito especificando-se a

sua posição, através de um índice. O índice pode variar de 0 até o tamanho do vetor menos 1.

Exemplo: **int** pares[6];

0	1	2	3	4	5
2	**4**	**6**	**8**	**10**	**12**

← *posições dos elementos no vetor*

Suponha que o vetor pares tenha sido inicializado com os elementos colocados nas "caixas" acima, cada uma representando uma das seis posições declaradas do vetor. Em um programa, para ser ter acesso ao terceiro elemento do vetor, deve-se fazer:

pares[**2**], *onde 2 é o índice do terceiro elemento.*

Neste caso, o valor retornado seria 6. Outros exemplos de acesso aos elementos de vetores por meio do seu índice (qualquer expressão inteira positiva) podem ser vistos abaixo:

a) a[1] : representa o acesso ao 2º elemento do vetor **a*;***
b) nome[p] : representa o acesso ao *p-ésimo* elemento do vetor *nome;*
c) x[2 * 1 + 3 * j – 4 * k] : A avaliação da expressão entre colchetes, que deverá resultar em um inteiro positivo, fornecerá a posição do elemento a ser acessado no vetor **x**;
d) pares[6] : ERRADO, pois o vetor pares possui 6 elementos, acessíveis da posição 0 à posição 5.

Atribuindo valores a um vetor

Quando se declara um vetor, da mesma forma que na declaração de uma variável simples, os seus elementos são indeterminados ou vazios. Dessa forma, deve-se inicializar cada elemento do vetor antes que se possa usá-lo, o que pode ser feito por meio de uma atribuição (=) ou de uma instrução de leitura (*cin*, por exemplo).

Na atribuição, como vimos anteriormente, não é possível utilizar o nome do vetor diretamente. Nesse caso, deve-se referenciar cada posição do vetor que se deseja inicializar por meio do seu índice.

Exemplos:

a. x[1] = 0;

b. num[3] = 3 * num[1] + 5 * num[2];

c. **for** (**int** i = 0 ; i < 10 ; i++) {

    p[i] = 3 * i – 1;

   }

Leitura e escrita de vetores

A leitura dos elementos de um vetor deve ser feita elemento por elemento, normalmente utilizando-se uma estrutura de repetição. Abaixo, vê-se alguns exemplos:

```
/* Inicialização de 100 elementos para o vetor x. A variação da variável de controle "i", a cada repetição, faz com que cada elemento do vetor seja acessado (x[i]). */
a) int x[100];
   for (int i = 0; i < 100; i++) {
        cin >> x[i];
   }

/* Inicialização de "n", sendo "n" um valor fornecido anteriormente e que esteja dentro dos limites para o tamanho do vetor "num". Adicionalmente, neste exemplo, acrescentou-se como condição adicional que o vetor receberia apenas números inteiros positivos. */
 b) for (int k = 0; k <= n; k++) {
         do {
             cout << "Digite um número inteiro positivo";
             cin >> num[k];
         } while (num[k] > 0);
   }

/* Semelhante ao exemplo anterior, mas neste caso, é lido um número, e este número é
   atribuído à posição do vetor. */
 c)  int i = 0;
   while (i < n) {
        do {
             cout << "Digite um número positivo";
             cin >> p;
        } while (p > 0);
        x[i] =  p;
        i = i + 1;
   }
```

A escrita de vetores também é feita elemento por elemento, utilizando-se a instrução escreva:

a.
```
for (int i = 0; i < 10; i++)  {
    cout << x[i] << endl;
}
```

b.
```
cout << “Vetor solução: ” << endl;
for (int j = 0; j < n; j++) {
    cout << “x[” <<  j <<  “]=” << x[j] << endl;
}
```

EXEMPLOS DE ALGORITMOS COM VETORES

1. Soma dos elementos de dois vetores, com atribuição em um terceiro vetor

```
#include <iostream>
using namespace std;

int main () {
    float vet1[5];
    float vet2[5];
    float vet3[5];

    //Leitura dos elementos do primeiro vetor
    for (int i = 0; i < 5; i++) { //i vai de 0 a 4
        cout << "vet2[" << i << "] = ";
        cin >> vet1[i];
    }

    cout << endl;
    /*Leitura dos elementos o segundo vetor, soma dos elementos e atribuição ao terceiro vetor */
    for (int i = 0; i < 5; i++) {
        //leitura do segundo vetor
        cout << "vet2[" << i << "] = ";
        cin >> vet2[i];
        //soma
        vet3[i] = vet1[i] + vet2[i];
    }

    cout << endl;
    //impressão do vetor resultado
    for (int i = 0; i < 5; i++) {
        cout << "soma[" << i << "] = " << vet3[i] << endl;
    }

    return 0;
}
```

2. Geração e impressão da série de Fibonacci

```
#include <iostream>
using namespace std;

int main () {
	int fib[100];
	int n = 0; //número de elementos da série

	do {
		cout << "Número de elementos (entre 1 e 100): ";
		cin >> n;
	} while (n < 1 && n > 100);

	fib[0] = 1;
	fib[1] = 1;

	//geração dos elementos da série. Salvos no vetor "fib"
	for (int i = 2; i < n; i++) {
		fib[i] = fib[i-2] + fib[i-1];
	}

	//Impressão da série de Fibonacci
	for (int i = 0; i < n; i++) {
		cout << fib[i] << endl;
	}

	return 0;
}
```