Aula Extra de Estruturas de Dados

Programa Mínimo em C/C++

```
#include <iostream>
using namespace std;
int main(void) {

}
```

Vetores

"tabelas" de números de tipos iguais declaração:

```
tipo nome[colunas];
```

Uso:

```
nome[coluna] = valor;
variavel = nome[coluna];
cout << nome[coluna];
cin >> nome[coluna];
```

Exemplos:

```
int v[4];
int x = 20;
cin >> v[0];
v[1] = 10;
v[2] = v[0]*v[1];
v[3] = x;
v[4] = x;  // Erro!
cout << v[0] << v[1] << v[2] << v[3] << endl;
```

Vetores - Exercícios

1) Crie um vetor de inteiros com 50 posições e, sem preenche-las, imprima cada uma delas.
2) Modifique o programa anterior para que ele preencha o vetor com os valores de 0 a 49, antes de imprimi-lo.
3) Modifique o programa anterior para que ele preencha o vetor com os valores de 1 a 50
4) Modifique o programa anterior para que ele crie um segundo vetor de inteiros, com 50 posições, copiando o primeiro vetor para o segundo
5) Modifique o programa anterior para que o segundo vetor tenha apenas 25 posições e apenas as valores pares sejam copiados do primeiro vetor para o segundo.
6) Modifique o programa anterior para que todos os valores do segundo vetor sejam impressos.
7) Modifique o programa anterior para que apenas os valores divisíveis por 10 sejam impressos.
8) Modifique o programa para que ele imprima a soma de todos os valores do primeiro vetor e também a soma de todos os valores do segundo vetor.

Funções

"Algoritmo que pode ser executado por outro algoritmo."

Declaração / Protótipo:

```
tipo_de_retorno nome([tipo1 parm1][, tipo2 parm2][, ...][, tipon parmn]);
```

Definição da Função

```
tipo_de_retorno nome([tipo1 parm1][, tipo2 parm2][, ...][, tipon parmn]) {

	// código da função

	return valor;// valor deve ser do mesmo tipo que tipo_de_retorno
}
```

Uso:

```
nome();		// Chamada de função sem parâmetro
nome(x);		// Chamada de função com um parâmetro
nome(1,7,x);	// Chamada de função com 3 parâmetros
x = nome();	// Chamada com armazenamento do retorno
```

Exemplos:

```
// Retorna 1 se o VALOR for ímpar; 0 caso contrário
int ehImpar(int valor) {
	int i;
	i = valor%2; // valor%2 resulta em 1 para números ímpares
	return i;
}

int main(void) {
	int res;
	res = ehImpar(3);		// O "3" será copiado para o parâmetro VALOR
	if (res == 1) cout << "Valor é ímpar!";
	else cout << "Valor é par!";
}
```

Escopo das Variáveis

Há três lugares possíveis em que uma variável pode ser declarada:
a) Fora de qualquer função, abaixo dos "includes"
b) Dentro de uma função
c) Como parâmetro de uma função

No caso A, aquela variável é válida em qualquer parte do programa.
Nos casos B e C, a variável só é válida NA FUNÇÃO EM QUE FOI DECLARADA.
Mesmo no caso C (parâmetro)? SIM!
Então se uma função modificar o valor de um parâmetro essa mudança não vai valer no resto do programa? NÃO!

E se eu precisar que a mudança em um parâmetro valha no resto do programa? Por exemplo:

```
// Multiplica o valor por 2
void atualiza(int valor) {
	valor = valor * 2;
}

int main(void) {
	int x;
	cin >> x;
	atualiza(x);
	cout << x;
}
```

Basta modificar a declaração da função, acrescentando & na frente do nome da variável que muda:

```
// Multiplica o valor por 2
void atualiza(int &valor) {
	valor = valor * 2;
}
```

Isso se chama "passagem por referência". Sem o "&" chama-se "passagem por valor".
Quando um parâmetro for passado por referência, NÃO podemos indicar um número no lugar deste parâmetro:

```
// Multiplica o valor por 2
void atualiza(int &valor) {
	valor = valor * 2;
}

int main(void) {
	atualiza(3); // ERRO!
}
```

Vetores são SEMPRE passados por referência, não se deve indicar o &.
Além disso, não se indica, usualmente, o tamanho do vetor que é recebido como parâmetro:

```
void atualiza(int vetor[]);	// Usual.
void atualiza(int vetor[5]);	// Incomum e, normalmente, "incorreto".
```

1) Crie uma função **imprime(int v[], int n)** que deve imprimir os **n** valores de um vetor **v**. Use essa função no programa que já estava pronto.
2) Crie uma função chamada **preenche(int v[], int n, int vi)**, que preencha o vetor **v**, com **n** posições, com valores inteiros sequenciais, começando pelo valor **vi**. Use a função no último programa.
3) Crie uma função **copiaPares(int o[], int d[], int n)** que copia os valores pares (até **n** valores) de**o** para **d**. Use esta função no programa que já estava pronto.
4) Crie uma função **somaResultados(int v1[], int v2[], int &s1, int &s2)** que some todos os elementos de **v1** em **s1** e todos os valores de **v2** em **s2**.