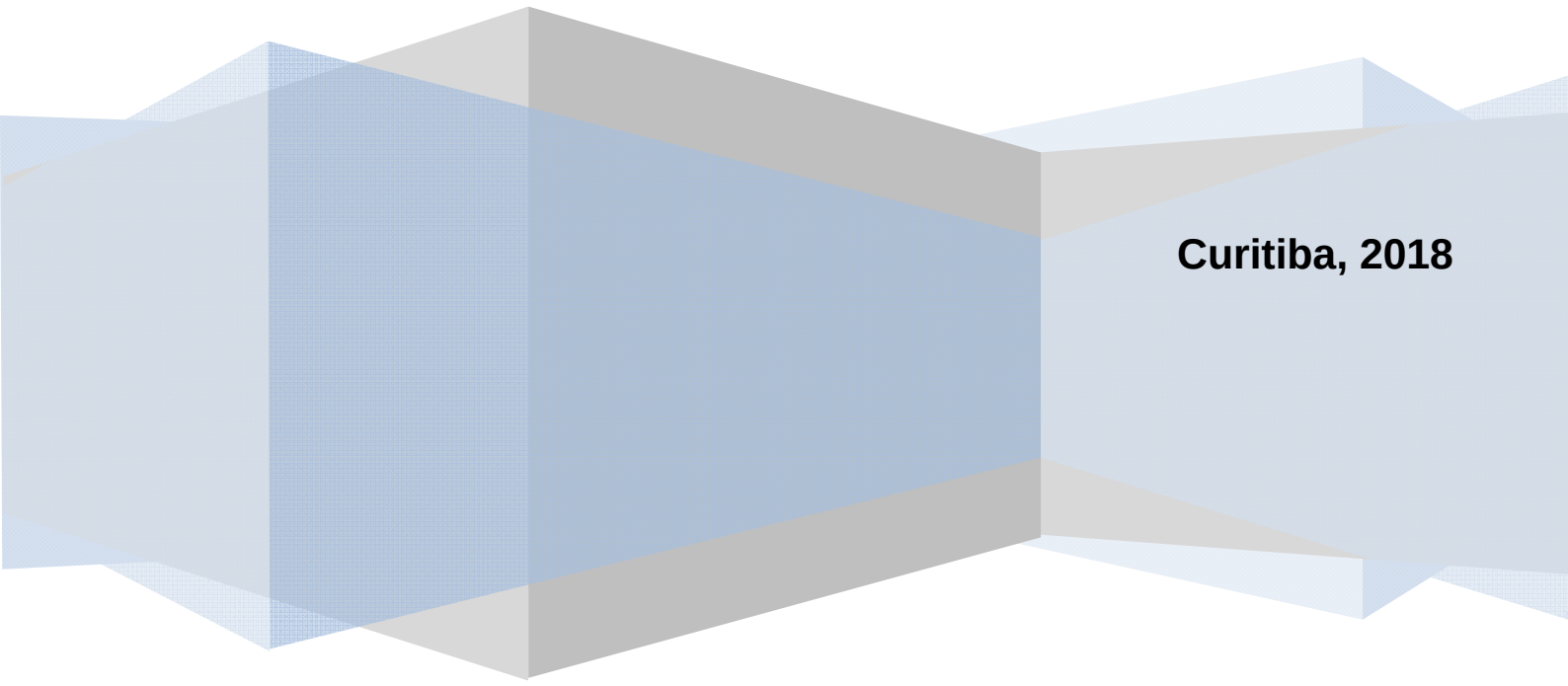
Introdução à programação C++

Cicero Aparecido Bezerra



Curitiba, 2018

SUMÁRIO

1 SOBRE O AUTOR

Possui graduação em Informática pela Universidade do Vale do Rio dos Sinos (1992), mestrado em Engenharia de Produção pela Universidade Federal de Santa Catarina (2001), doutorado em Engenharia de Produção pela Universidade Federal de Santa Catarina (2007) e estágio pós-doutoral em Gestão Estratégica da Informação e do Conhecimento pela Pontifícia Universidade Católica do Paraná (2012). Atualmente é professor Associado nível II da Universidade Federal do Paraná. Tem experiência em profissional em desenvolvimento, implantação e gestão de Sistemas de Informação e, Processos de Produção. Enquanto docente, leciona disciplinas alinhadas ao desenvolvimento de Sistemas de Informação e Análise de Dados. Como pesquisador, tem voltado sua atenção aos Métodos e Técnicas de Análise de Dados, aplicados à Gestão do Conhecimento e Inovação.

2 APRESENTAÇÃO

O material ao qual vocês estão tendo acesso apresenta, passo a passo, uma maneira para o desenvolvimento de programas computacionais na linguagem C++. Foi desenvolvido com a expectativa que sua leitura sequencial conduza, naturalmente, à construção de programas de computador, a partir de uma linguagem coloquial, próxima ao leitor.

A primeira parte consiste em uma visão geral das instruções e operações básicas requeridas na linguagem C++. A seguir, são apresentadas as estruturas de seleção simples, composta e de múltipla escolha. Logo na sequência, é a vez das estruturas de repetição, com teste de início e no fim e, com variável de controle. A próxima seção traz as estruturas que permitem o armazenamento e manipulação de dados. Finalmente, a próxima seção demonstra o emprego de funções, cujo intuito é modularizar os programas, tornando-os mais econômicos em termos de linhas de código.

Espero que este material possa ser útil na compreensão básica dos conceitos e comandos necessários à programação de computadores em C++.

Bons estudos...

3 CARACTERÍSTICAS DAS LINGUAGENS DE PROGRAMAÇÃO

As linguagens de programação imperativas são, em graus variados, abstrações da arquitetura do computador de von Neumann. Dois dos componentes principais desta arquitetura são a memória (responsável por armazenar tanto instruções como dados) e o processador (responsável por fornecer operações para modificar o conteúdo da memória).

Constantes

É um dado que não tem seu valor alterado ao longo do tempo, durante a execução de um programa. Pode ser numérica, literal ou lógica, conforme os exemplos:

- **Numérica**: 15; +8; -3; 0,3258; +8,65; -25,387; 5,7 * 10 ** 2; 8 / 10 ** -10...
- **Literal**: representada por qualquer sequência de caracteres, desde que colocada entre aspas: “Programacao”; “Sistemas_de_Informacao”; “124,67”; “31/03/1999”.

Palavras especiais

As palavras especiais em linguagens de programação são utilizadas para tornar os programas mais legíveis e dar nome às ações a serem executadas. Também são utilizadas para separar as entidades sintáticas dos programas. Estas palavras não podem ser utilizadas em contextos diferentes aos quais foram criadas.

Variáveis

Posição de memória que pode variar ao longo de um tempo, durante a execução de um programa. Apesar de existir a possibilidade desta variável assumir diferentes valores, ela só pode armazenar um valor a cada instante. Uma variável é identificada por 1 ou mais caracteres, sendo que a primeira posição **deve ser obrigatoriamente** uma letra e as demais **não podem ser símbolos especiais**: ()*&%$#@!"',.:+-*/][{, etc... Assim como as constantes, as variáveis serão subdivididas em tipos, a saber:

- **Real**: são aquelas que só podem assumir os valores formados pelo conjunto dos números reais;
- **Inteiro**: só podem assumir os valores do conjunto dos números inteiros.
- **Lógica**: só assumem 2 valores: **V**erdadeiro ou **F**also.
- **Caracteres**: pode assumir quaisquer valores compostos por letras, números ou símbolos especiais.

As variáveis só podem assumir valores de um mesmo tipo e para que isso fique explícito ao programa, nossa primeira providência é declara-las. Ao declararmos as variáveis, é feita a associação do nome escolhido com a respectiva posição de memória.

Todas as variáveis possuem um **tipo** que determina a faixa de valores e o conjunto de operações definidas para aquele tipo.

As variáveis podem assumir valores diferentes em determinado momento quando da execução do programa. A esta mudança de valores chamamos **vinculação**, que nada mais é do que uma associação entre um atributo e uma entrada ou entre uma operação e um símbolo. A vinculação pode ser **estática** (quando ocorre antes do tempo de execução e permanece inalterada ao longo da execução do programa) ou **dinâmica** (quando modificada no decorrer da execução do programa).

Uma variável pode ser **declarada** (quando é atribuído um tipo específico à variável) ou **definida** (quando especificam-se atributos e causam alocação de memória).

4 IMPLEMENTAÇÃO EM C++

Para representarmos a declaração de variáveis em C++, é necessário conhecermos a estrutura do programa. Um programa C++ é, basicamente, uma função chamada `main( )`. Uma função consiste em um bloco de código que executa uma ou mais ações comuns. Antes de trabalhar com a função é necessário informar ao pré-compilador quais bibliotecas de funções serão utilizadas pelo seu programa.

Exemplo:

```
# include <iostream.h> /* biblioteca que controla entrada e saída */
void main() /* indica que a função principal (seu programa) não irá
               retornar nenhum parâmetro (void) e que  não receberá nenhum
               parâmetro para sua execução ( ) */
{
const valor = 15; /* declaração de constante */
enum cargo {operacional, tatico, estrategico}; /* constante */
float slarioBruto, salarioLiquido; /* o formato de nomes diferencia letras
                                      maiúsculas e minúsculas - declaração */
int idade = 0; /* permite a atribuição de um valor para a variável
                  no momento de sua declaração - definição */
char estagio; /* não existe variável lógica em C++ */
}
```

No exemplo anterior, as palavras em azul são palavras especiais na linguagem C++, as palavras em verde são comentários e as demais são nomes criados para identificar variáveis do programa.

Inicialmente iremos utilizar apenas os seguintes tipos de variáveis:

Tipo	Tamanho	Valores
`unsigned short int`	2 bytes	0 a 65.535
`short int`	2 bytes	-32.768 a 32.767
`unsigned long int`	4 bytes	0 a 4.294.967.295
`long int`	4 bytes	-2.147.483.648 a 2.147.483.647
`char`	1 byte	256 caracteres

`float`	4 bytes	1.2e-38 a 3.4e38
`double`	8 bytes	2.2e-308 a 1.8e308

4.1 Instruções primitivas

Para praticarmos os conceitos de variáveis e constantes é necessária a utilização de algumas instruções de programação.

Comandos de atribuição

Estes comandos permitem que armazenemos um valor em uma determinada variável. È imprescindível que a variável seja do mesmo tipo do valor a ser armazenado.

```
# include <iostream.h>
void main()
{/*---------- Declaração de variáveis ----------*/
const valor = 15;
enum cargo {operacional, tatico, estrategico};
float salarioBruto, salarioLiquido;
int idade = 0;
char estagio;

/*---------- Início de programa ----------*/
salarioBruto = 1000; /* comando de atribuição */
estagio = 's'; /* comando de atribuição */
}
```

Comandos de saída de dados

São comandos com a finalidade de permitir que uma informação seja transmitida ao usuário.

Comandos de entrada de dados

São comandos com a finalidade de permitir que o usuário dê entrada nos dados que farão parte do processamento. Deve ser especificado de tal maneira que não dê margens para a entrada de dados incorretos.

```
# include <iostream.h>
void main()
{/*---------- Declaração de variáveis ----------*/
const valor = 15;
enum cargo {operacional, tatico, estrategico};
float salarioBruto, salarioLiquido;
int idade = 0;
char estagio;

/*---------- Início de programa ----------*/
cout << ("Informe salário bruto"); /* comando de saída */
cin >> (salarioBruto); /* comando de entrada */
}
```

4.2 Operadores e expressões

Operadores são símbolos aplicados em variáveis e/ou constantes. Esta combinação é chamada de expressão. Os símbolos devem ser utilizados em expressões que contém variáveis, constantes e funções **do mesmo tipo**, agindo sobre os mesmos e produzindo um resultado.

Operadores aritméticos

São aqueles que atuam sobre variáveis, constantes e funções numéricas, gerando portanto, um resultado numérico. São os seguintes:

Operador	Tipo	Operação	Prioridade
+	Unário	Manutenção de sinal	1
-	Unário	Inversão de sinal	1
*	Binário	Multiplicação	3
/	Binário	Divisão	3
+	Binário	Adição	4
-	Binário	Subtração	4

Exemplo:

2 + 3 é a expressão, onde:

2 e 3 são constantes (operandos)

+ é o operador aritmético

x * y é a expressão, onde:

x e y são as variáveis (operandos)

* é o operando aritmético

Os seguintes exemplos, mostram a representação de expressões matemáticas:

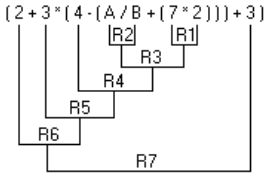
Expressão matemática	Expressão computacional
2 . a + b	2 * a + b
b² + 4 . a . c	b ** 2 + 4 * a * c
¾ + c	3 / 4 + c

Assim como na matemática, os operadores agem sobre os operandos seguindo uma determinada prioridade, que pode ser alterada através do uso de parênteses, conforme exemplo:

- 10 * 3 – 8 = 22, ou seja, como a multiplicação tem prioridade sobre a subtração, resolve-se primeiro a multiplicação e o resultado será o operando da subtração. Caso desejar executar primeiro a subtração, usa-se o parênteses para alterar a prioridade
  10 * (3 – 8) = - 50

- caso os operadores possuírem a mesma prioridade, executa-se a partir da parcela mais a esquerda para a parcela mais a direita:

Também como na matemática, quando existirem mais de um grupo de parênteses, a prioridade de execução deve ser do mais interno para o mais externo, conforme exemplo:



Segue um exemplo sobre operadores:

```
# include <iostream.h>
void main()
{/*---------- Declaração de variáveis ----------*/
int minhaIdade = 34;
int suaIdade = 34;

/*---------- Início de programa ----------*/
cout << "Eu tenho " << minhaIdade << " anos de idade\n";
cout << "Voce tem " << suaIdade << " anos de idade\n";
minhaIdade++; /* incremento sufixo em uma unidade */
++suaIdade; /* incremento prefixo em uma unidade */
cout << "Um ano se passou...\n";
cout << "Eu tenho " << minhaIdade << " anos de idade\n";
cout << "Voce tem " << suaIdade << " anos de idade\n";
cout << "Outro ano se passou...\n";
cout << "Eu tenho " << minhaIdade++ << " anos de idade\n";
cout << "Voce tem " << ++suaIdade << " anos de idade\n";
cout << "Verificando a idade novamente...\n";
cout << "Eu tenho " << minhaIdade << " anos de idade\n";
cout << "Voce tem " << suaIdade << " anos de idade\n";
}
```

4.2.1 Operadores relacionais

São aqueles que resultam em resultados lógicos através da comparação entre 2 operandos **do mesmo tipo**. Todos são operadores do tipo binário, ou seja, sempre precisam ter um operando à esquerda e à direita do operador.

Operador	Operação	Prioridade
==	Igualdade	1
!=	Diferença	1
<	Menor	1
>	Maior	1
<=	Menor ou igual	1
>=	Maior ou igual	1

O resultado de uma operação relacional só pode assumir 2 valores: **Verdadeiro** ou **Falso** (**Sim** ou **Não**). Da mesma forma que os operadores aritméticos, a operação lógica **é executada a partir do resultado da expressão mais à esquerda comparado com o resultado da expressão mais à direita do operador**.

4.2.2 Operadores lógicos

São aqueles que fornecem **resultados lógicos** através da **comparação** entre 2 **expressões lógicas**, realizando a conjunção, a disjunção ou a negação dos resultados.

Operador	Tipo	Operação	Prioridade
!	Unário	Negação ou inversão (não)	1
&&	Binário	Conjunção ou união (e)	2
||	Binário	Disjunção (ou)	3

5 ESTRUTURAS DE SELEÇÃO

Permitem a execução de determinado conjunto de instruções conforme o valor de uma expressão.

5.1 Estrutura de seleção simples

É quando um conjunto de ações são executadas se a condição sob as quais estão subordinadas for verdadeira.

```
# include <iostream.h>
void main()
{/*---------- Declaração de variáveis ----------*/
const valor = 15;
enum cargo {operacional, tatico, estrategico};
float salarioBruto, salarioLiquido;
int idade = 0;
char estagio;

/*---------- Início de programa ----------*/
cout << ("Informe o salario bruto "); cin >> (salarioBruto);
cout << ("\nInforme o salarioLiquido "); cin >> (salarioLiquido);
cout << ("\nE estagiario? "); cin >> (estagio);
if (salarioLiquido == salarioBruto)
    cout << ("Salarios iguais\n");
}
```

5.2 Estrutura de seleção composta

É quando existirem ações a serem executadas para as condições verdadeira e falsa.

```
# include <iostream.h>
void main()
{/*---------- Declaração de variáveis ----------*/
const valor = 15;
enum cargo {operacional, tatico, estrategico};
float salarioBruto, salarioLiquido;
int idade = 0;
char estagio;

/*---------- Início de programa ----------*/
cout << ("Informe o salario bruto "); cin >> (salarioBruto);
cout << ("\nInforme o salarioLiquido "); cin >> (salarioLiquido);
cout << ("\nE estagiario? "); cin >> (estagio);
if (salarioLiquido == salarioBruto)
    cout << ("Salarios iguais\n");
```

```
else
    cout << ("Salarios diferentes\n");
}
```

5.3 Seleção de múltipla escolha

É a seleção de várias opções para a variável que faz parte da condição, ou seja, a condição pode ter mais que 2 opções que a satisfaçam.

```
# include <iostream.h>
void main()
{/*---------- Declaração de variáveis ----------*/
const valor = 15;
int cargo;
float salarioBruto, salarioLiquido;
int idade = 0;
char estagio;

/*---------- Início de programa ----------*/
cout << ("\nInforme o salario bruto: "); cin >> (salarioBruto);
cout << ("\nInforme o cargo.....: "); cin >> (cargo);
switch (cargo)
    {
    case 1: salarioLiquido = salarioBruto + salarioBruto*valor/100;
            cout << ("\nSalario liquido e' ", salarioLiquido);
            break;
    case 2: salarioLiquido = salarioBruto + salarioBruto*valor/80;
            cout << ("\nSalario liquido e' ", salarioLiquido);
            break;
    case 3: salarioLiquido = salarioBruto + salarioBruto*valor/50;
            cout << ("\nSalario liquido e' ", salarioLiquido);
            break;
    default: cout << ("\nCargo invalido!!!");
             break;
    }
}
```

6 ESTRUTURAS DE REPETIÇÃO

Permitem a execução de determinado conjunto de instruções tantas vezes quanto forem necessárias, conforme o valor de uma expressão.

6.1 Estrutura de repetição com teste no início

Consiste numa estrutura de controle do fluxo lógico que permite executar diversas vezes um mesmo trecho do algoritmo, porém, sempre verificando antes de cada execução se é "permitido" repetir o mesmo trecho.

```
# include <iostream.h>
void main()
{/*---------- Declaração de variáveis ----------*/
int a = 0;
int b = 0;
int resultado = 0;
char tecla = 's';

/*---------- Início de programa ----------*/
while (tecla == 's')
    {
    cout << ("\nDigite o valor de a: "); cin >> (a);
    cout << ("\nDigite o valor de b: "); cin >> (b);
    resultado = a + b;
    cout << ("\nO resultado e......: ", resultado);
    cout << ("\nDigite <s> para continuar "); cin >> (tecla);
    }
}
```

Ou seja, enquanto a condição <tecla = 's'> for Verdadeira, os comandos constantes entre { } serão executados. Quando a condição <tecla = 's'> for Falsa, os comandos não serão mais executados e finaliza-se a instrução *While*. Esta situação (Verdadeiro ou Falso) indica que um dos comandos que estão sendo repetidos, em algum momento, deve ter seu valor alterado para que a repetição tenha fim, caso contrário o algoritmo executará os comandos infinitamente.

Nesta estrutura utilizaremos com bastante freqüência, os contadores e acumuladores.

Contador: é uma instrução criada por nós quando necessitamos contar quantas vezes estamos repetindo algum comando. Geralmente, é a instrução que irá satisfazer a condição de repetição do comando W*hile* e, também, irá determinar seu fim.

```
# include <iostream.h>
```

```
void main()
{/*---------- Declaração de variáveis ----------*/
float nota1 = 0;
float nota2 = 0;
float media = 0;
int aluno = 0;

/*---------- Início de programa ----------*/
while (aluno < 5)
    {
    aluno = aluno + 1;
    cout << "\nAluno numero " << aluno << "\n";
    cout << ("Digite a nota1: "); cin >> (nota1);
    cout << ("Digite a nota2: "); cin >> (nota2);
    media = (nota1+nota2)/2;
    cout << "A media e......: " << media << "\n";
    }
}
```

Ou seja, iremos contabilizar cada execução (aluno = aluno + 1) dos comandos dentro das { } e quando estes comandos tiverem sido executados 5 vezes, teremos o término da repetição.

Acumulador: é uma instrução que nos permite acumular o valor de determinada variável, a cada repetição, para que no final tenhamos o valor total da variável que se repetiu.

```
# include <iostream.h>
void main()
{/*---------- Declaração de variáveis ----------*/
float nota1 = 0;
float nota2 = 0;
float mediaIndividual = 0;
float mediaGeral = 0;
int aluno = 0;

/*---------- Início de programa ----------*/
while (aluno < 5)
    {
    aluno = aluno + 1;
    cout << "\nAluno numero " << aluno << "\n";
    cout << ("Digite a nota1: "); cin >> (nota1);
    cout << ("Digite a nota2: "); cin >> (nota2);
    mediaIndividual = (nota1+nota2)/2;
    mediaGeral = mediaGeral + mediaIndividual;
    cout << "A media e......: " << mediaIndividual << "\n";
    }
    cout << "A media geral e " << mediaGeral << "\n";
}
```

A execução deste conjunto de instruções significa que a cada repetição dos comandos internos do *While*, estes serão contados (aluno = aluno + 1) e cada repetição será somada à anterior e acumulada (mediaGeral = mediaGeral + mediaIndividual).

6.2 Estrutura de repetição com teste no final

Consiste numa estrutura de controle do fluxo lógico que permite executar diversas vezes um mesmo trecho do algoritmo, porém, sempre verificando após cada execução se a condição permanece Verdadeira (o que indicará o final das repetições) ou se é Falsa (o que indicará nova repetição).

```
# include <iostream.h>
void main()
{/*---------- Declaração de variáveis ----------*/
char sexo = ' ';
char tecla = ' ';
float idade = 0;
float acumuladorIdade = 0;
float contador = 0;
float media = 0;
/*---------- Início de programa ----------*/
do
     {
     cout << ("\nInforme o sexo.: "); cin >> (sexo);
     cout << ("Informe a idade: "); cin >> (idade);
     if (sexo == 'f')
          {
          acumuladorIdade = acumuladorIdade + idade;
          contador = contador + 1;
          }
     cout << ("Digite <s> para continuar "); cin >> (tecla);
     }
while (tecla == 's');
media = acumuladorIdade / contador;
cout << "\nA media de idade das mulheres e " << media << "\n";
}
```

O programa anterior executa os comandos dentro do laço *do-while* pelo menos uma vez, independente do valor da variável *tecla* (que controla a repetição do bloco).

6.3 Estrutura de repetição com variável de controle

Os comandos anteriores mostram que teremos uma série de repetições enquanto a condição (a que está subordinado) for Verdadeira. Temos ainda um outro comando de repetição que, de antemão, saberemos quantas vezes ele será executado.

```
# include <iostream.h>
void main()
{/*---------- Declaração de variáveis ----------*/
int contador = 0;
/*---------- Início de programa ----------*/
for (contador = 0; contador < 5; contador = contador + 1)
    if (contador > 1)
        cout << "Repetiu " << contador << " vezes\n";
    else
        cout << "Repetiu " << contador << " vez\n";
}
```

7 ESTRUTURAS FUNDAMENTAIS DE DADOS

Conforme visto anteriormente, uma variável corresponde a uma (1) posição de memória. Esta variável passa a existir a partir do momento em que a declaramos, associando um nome ou identificador. Qualquer referência a esta posição de memória significa ter acesso ao conteúdo desta única posição de memória.

Fazendo uma comparação com um armário, a variável corresponde a uma (1) gaveta que pode ter seu conteúdo ocupado apenas por um (1) objeto. A partir de agora iremos armazenar mais de um (1) objeto na gaveta. Variáveis implementadas desta forma recebem o nome de estrutura de dados. O principal uso de uma estrutura de dados é a possibilidade de armazenarmos e resgatarmos todos os valores das variáveis.

Qualquer estrutura de dados deve possuir um índice, que possibilita o acesso a um dado individual na estrutura. Podemos ilustrar este ponto através de um livro, onde o livro é a estrutura de dados e os capítulos são os dados armazenados na estrutura. Para que possamos encontrar um determinado capítulo, basta acessarmos o índice e este nos dirá em qual a página está o capítulo desejado.

Existem, basicamente, dois tipos de estruturas fundamentais de dados: aquelas que armazenam dados da mesma natureza (conhecidas por vetores e/ou matrizes e/ou variáveis compostas homogêneas) e aquelas que armazenam dados de naturezas diferentes (conhecidos por registros e/ou variáveis compostas heterogêneas).

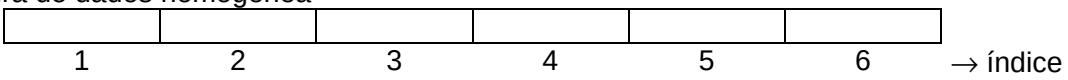
7.1 Variáveis compostas homogêneas

São estruturas de dados onde são armazenados dados da mesma natureza. É um conjunto homogêneo de dados. Podem ser divididas em:

7.2 Variáveis compostas unidimensionais

Ou vetores. São aquelas que são armazenadas em uma única linha, contendo para esta linha, várias posições (colunas) onde os dados são efetivamente armazenados.

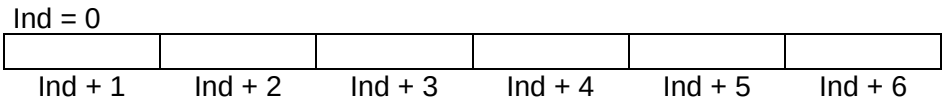
Estrutura de dados homogênea



Cada posição representada pelo índice está preparada para ter uma (e apenas uma) variável armazenada.

Como existem 7 posições que irão armazenar as variáveis, ao necessitar inicializarmos esta estrutura teremos que usar um comando de repetição, onde atribuiremos um valor nulo para cada posição:

Desta forma, cada posição representada por um apontador (índice) que irá variar de 1 em 1, receberá o valor 0 (zero) a cada variação. Este exemplo também mostra como "caminharmos" dentro do vetor:



É importante frisar que um vetor composto de 5 elementos, não terá qualquer posição menor que 1 ou maior que 5. Portanto esta abrangência de valores terá que ser bem controlada.

Todas as demais operações com variáveis são permitidas às estruturas de dados. A seguir um exemplo em C++:

```
# include <iostream.h>
void main()
{/*----- Inicialização de variáveis -----*/
int vetor[5];
int contador = 0;
/*----- Corpo do programa -----*/
for (contador = 0; contador < 5; contador = contador + 1)
{
    cout << "Digite vetor[" << contador << "]: ";
    cin >> vetor[contador];
}
cout << "\nOs valores digitados foram:";
for (contador = 0; contador < 5; contador = contador + 1)
    cout << "\nVetor na posicao " << contador << " = " << vetor[contador];
}
```

Em C++ é importante frisar que a primeira posição do vetor é referenciada pelo índice de número 0 (zero) e assim sucessivamente, ou seja, se um vetor possui 4 (quatro) posições, seus índices serão 0, 1, 2 e 3.

7.2.1 Variáveis compostas multidimensionais

Ou matriz. É o conjunto de dados referenciado pelo mesmo nome que necessita de mais de um índice para ter seus elementos individualizados. Enquanto no vetor tínhamos apenas 1 linha e várias colunas, na matriz teremos várias linhas e várias colunas (posições de memória) onde os dados estarão armazenados.

Vetor

1	2	3	4	5	6

Matriz

	1	2	3	4	5
1					
2					
3					

Todas as particularidades de um vetor valem para uma matriz, bem como as operações sobre seus elementos. A seguir um exemplo em C++:

```
# include <iostream.h>
void main()
{/*---- Inicialização de variáveis -----*/
int matriz[3][4] = {{1,2,3,4},
{5,6,7,8},
{9,10,11,12}};
int coluna = 0;
int linha = 0;
/*----- Corpo do programa -----*/
for (linha = 0; linha < 3; linha = linha + 1)
    for (coluna = 0; coluna < 4; coluna = coluna + 1)
    {
        cout << "\nMatriz[" << linha << "][" << coluna << "]: ";
        cout << matriz[linha][coluna];
    }
}
```

Da mesma forma que um vetor é controlado a partir do índice 0 (zero), uma matriz possui suas linhas e colunas controladas a partir do valor 0 (zero). Note que em C++ podemos inicializar um vetor ou matriz a partir de sua declaração.

7.2.2 Variáveis compostas heterogêneas

Ou registros. São conjuntos de dados logicamente relacionados, porém de tipos diferentes (numérico, caracter). O conceito de registro visa facilitar o agrupamento de variáveis que não possuem o mesmo tipo, correspondendo a conjuntos de posições de memória conhecidos por um mesmo nome e individualizados por identificadores associados a cada conjunto de posições.

Registro

Nome	caracter			
Nota	real	real	real	real

Na variável composta homogênea, a individualização de um elemento é feita através de índices, já no registro cada componente é individualizado pela explicitação de seu identificador.

O exemplo a seguir demonstra todas estas situações e vincula um registro a um vetor, ou seja, cada posição de um vetor irá armazenar um registro contendo o campo nome (caracter) e um campo nota (outro vetor com 4 posições reais):

```
# include <iostream.h>
void main()
{/*----- Declaração de variáveis -----*/
struct cadAluno
{
    char nome[20];
    float nota[04];
};
struct cadAluno aluno[05];
int contAluno = 0;
int contNota = 0;
float media = 0;
/*----- Corpo do programa -----*/
for (contAluno = 0; contAluno < 5; contAluno = contAluno + 1)
{
    cout << "\nInforme o nome do aluno: ";
    cin >> aluno[contAluno].nome;
    media = 0;
    for (contNota = 0; contNota < 4; contNota = contNota + 1)
    {
        cout << "Informe a nota " << contNota + 1 << " do aluno: ";
        cin >> aluno[contAluno].nota[contNota];
        media = (media + aluno[contAluno].nota[contNota]);
    }
    cout << "A media é: " << media/4;
}
}
```

8 FUNÇÕES

Uma função (ou subprograma) é um conjunto de código-fonte utilizado por determinadas partes do programa principal. É como se dividíssemos um programa maior em programas menores que seriam executados à medida em que fossem chamados. A vantagem de se programar utilizando funções é a obtenção de economia em espaço de memória e codificação.

As funções apresentam as seguintes características:

- Cada função tem um único ponto de entrada;
- O programa que chamou a função é interrompido durante a execução da função, ou seja, somente existe 1 (um) programa sendo executado na memória em um dado momento;
- A função sempre retorna ao programa que a chamou quando do término de sua execução.

Em C++, as funções devem ser declaradas antes da função principal (main).

```
# include <iostream.h>
/*----- Função calcula área -----*/
float calcularArea(float auxBase, float auxAltura)
{
    return auxBase*auxAltura/2;
}

void main()
{/*----- Declaração de variáveis -----*/
    float base = 0;
    float altura = 0;
/*----- Programa principal -----*/
    cout << "\nInforme o valor da base..: "; cin >> base;
    cout << "Informe o valor da altura: "; cin >> altura;
    cout << "O valor da base é.......: " << calcularArea(base, altura);
}
```

As funções também podem trabalhar com variáveis locais (só existem na memória durante a execução da função). **Em C++ é altamente recomendável a utilização de variáveis locais**.

```
# include <iostream.h>
/*----- Função que converte temperatura Fahrenheit em Celsius -----*/
float conversao(float auxFahrenheit)
{
    float celsius = 0;
    celsius = ((auxFahrenheit-32)*5)/9;
    return celsius;
```

```
}

void main()
{/*----- Declaração de variáveis -----*/
    float fahrenheit = 0;
    float celsius = 0;
/*----- Programa principal -----*/
    cout << "\nEntre com a temperatura Fahrenheit: ";
    cin >> fahrenheit;
    celsius = conversao(fahrenheit);
    cout << "O valor da temperatura Celsius é..: " << celsius;
}
```

9 BIBLIOGRAFIA

DEITEL, H.M. **C++ Como Programar com Projeto Orientado a Objeto com UML**. Porto Alegre: Bookman, 2000.

LIBERTY, Jesse. **Aprenda em 24 horas C++**. Rio de Janeiro: Campus, 2000.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação**. Porto Alegre: Bookman, 2002.