Universidade Federal de Itajubá – UNIFEI
Curso de Engenharia de Computação
ECO 002 – Técnicas de Programação – Prof. Edmilson Marmo Moreira

Capítulo 4 – Introdução à Linguagem de Programação C++

> *"Não é que nos falte valor para empreender as coisas por elas serem difíceis; mas elas são difíceis precisamente porque nos falta valor para as empreender."*
>
> **Séneca**

4.1 Introdução

Um dos poderes de C++ é que ela deriva diretamente da linguagem C. Devido ao fato de C++ ser usualmente implementada como um tradutor que produz o código regular de C a partir dos comandos-fonte C++, C++ não é uma linguagem de programação completamente nova que requer um longo aprendizado.

Um programa C++ consiste em uma série de uma ou mais funções. Estas funções podem ser combinadas em um único arquivo, ou podem estar espalhadas através de vários arquivos em disco.

Deve haver uma – e somente uma – função principal, denominada `main()`.

C++ permite também grande modularidade na programação através de várias suplementações interessantes na programação C. A principal mudança é a introdução dos conceitos da **programação orientada a objetos**, como, por exemplo, classes, herança, polimorfismo, sobrecarga de funções e operadores, etc.

4.2 A estrutura básica de um programa C++

A unidade fundamental de programas C/C++ são as funções. Um programa C++ consiste em uma ou várias funções, além, é claro, das classes que tratam da programação orientada a objetos.

Forma geral das funções C++

Os elementos básicos de toda função C++ são os seguintes:

```
tipo nome ()
{
```

```
  instrução_1;
  instrução_2;
     .
     .
  instrução_n;
}
```

O primeiro exemplo é muito simples e ilustra o formato geral. Este programa deve ser armazenado com extensão `.cpp`, que indica um programa-fonte em C++ (*plus plus*).

Exemplo C++:

```
#include <iostream.h> // para entrada e saída

void main()
{
  // Apresentar uma mensagem na tela
  cout << "Primeiro exemplo";
}
```

Este programa compõe-se de uma única função. Com exceção da função *main*, o programador pode dar qualquer nome, dentro das regras de construção de identificadores da linguagem.

Toda função deve começar com uma chave de abertura de bloco **{** de deve terminar com uma chave de fechamento de bloco **}**.

Neste exemplo o tipo da função foi `void`, significando que a função não retornará valor para o sistema operacional. Em sistemas como o UNIX, a função principal deve retornar um valor inteiro (`int`).

Este programa contém uma única instrução:

```
cout << "Primeiro exemplo";
```

Essa instrução imprime a frase entre aspas duplas na tela. Toda instrução C++ termina em um ponto-e-vírgula. Uma função pode ter qualquer número de instruções. As instruções devem ser escritas entre as chaves que delimitam o corpo da função.

`cout` é um objeto da classe de I/O (input / output) predefinida em C++. A descrição completa deste objeto requer um entendimento maior sobre classes, objetos e sobrecarga de operadores, que serão discutidos no curso sobre orientação a objetos.

O operador <<, chamado "operador de inserção", conecta a mensagem a ser impressa a **cout**. A especificação de **cout** está contida no arquivo "iostream.h", por isso a necessidade da primeira linha do programa. Esta linha não é uma instrução. Ela é uma diretiva do pré-processador C++.

Toda diretiva é iniciada pelo símbolo # e seu texto deve ser escrito em uma única linha. As diretivas servem para auxiliar no desenvolvimento do programa-fonte.

A diretiva #include provoca a inclusão de outro arquivo no programa. O compilador substitui a diretiva pelo conteúdo do arquivo indicado antes de o programa ser compilado.

4.3 Mapeando os algoritmos estudados para programas em C++

Inicialmente utilizaremos apenas a função *main* para mapear os algoritmos desenvolvidos para a linguagem C++. O objetivo principal é apresentar as instruções básicas da linguagem.

Vejamos o primeiro exemplo da nota de aula 01.

Exemplo de Algoritmo:

```
Algoritmo
     declare A, B, C numérico
     leia A, B
     C ← 0
     C ← (A + B) × C
     escreva A, B, C
fim Algoritmo
```

Este algoritmo poderia ser traduzido para C++ conforme o exemplo a seguir:

Exemplo C++:

```
#include <iostream.h> // para entrada e saída

void main()
{
  int A, B, C;    // declaração das variáveis numéricas
  cin >> A >> B; // leitura das variáveis A e B
  C = 0;
  C = (A + B) * C;
  cout << A << B << C;
}
```

A primeira diferença entre a notação utilizada nos algoritmos e a linguagem C++ diz respeito à forma de declaração das variáveis. As três variáveis numéricas do exemplo foram declaradas como tipos inteiros (`int`).

Os tipos numéricos se dividem em vários outros tipos conforme mostra a tabela a seguir. Além disso, na linguagem C++ o tipo vem antes do nome da variável.

Classe	***Tipos e características***
Inteiros	`char` (8-bits: -128 a 127); `short` (16-bits: -32768 a 32767); `int` e `long` (32-bits: -2.147.483.648 a 2.147.483.647) *O intervalo para o tipo* `int` *depende do compilador e da arquitetura utilizada.* *Há na linguagem C++ o especificador* `unsigned` *que é combinado com os tipos anteriores para aproveitar o bit de sinal. Neste caso a variável representa somente números positivos.*
Reais	`float` (32-bits: 3.4E-38 a 3.4E+38) e `double` (64-bits: 1.7E-308 a 1.7E+308)

Além do nome e do tipo da variável, através da declaração a variável ganha um **escopo**. O escopo determina onde aquele identificador pode ser usado, pois, uma variável em C++ pode ser declarada em qualquer lugar do programa.

Outra característica importante no mapeamento realizado é a forma de entrada de dados que foi feita com a utilização do objeto **`cin`** do pacote `iostream.h`.

Existem diversas formas de realizar as operações de entrada e saída de dados, como, por exemplo, as funções **`printf`** e **`scanf`**, provenientes da linguagem C, e que se encontram na biblioteca `stdio.h`.

Destaca-se ainda, a leitura em série realizada com auxílio do operador **>>**. A mesma operação poderia ter sido realizado com a utilização de várias chamadas a **cin**.

Finalmente, as linguagens C/C++ utilizam o sinal de igual (=) como operador de atribuição. Durante uma comparação dois sinais de igual juntos são utilizados (==).

Observação

As linguagens C/C++ São ***case-sensitive***, ou seja, há distinção entre letras maiúsculas e letras minúsculas.

`PrimeiroExemplo` ≠ `primeiroexemplo`

Estrutura condicional

Como foi visto na introdução dos algoritmos estruturados, a estrutura condicional pode ser apresentada através de uma estrutura **simples** ou de uma estrutura **composta**, como apresentado nos exemplos abaixo:

Exemplo de estrutura condicional simples:

```
se condição
     então seqüência de comandos
fim se
```

Exemplo de estrutura condicional composta:

```
se condição
     então seqüência 1 de comandos
     senão seqüência 2 de comandos
fim se
```

Na linguagem C++ existem 2 estruturas condicionais: `if/else` e `switch/case`. Estudaremos, inicialmente, a estrutura `if/else`, devido à associação direta com os algoritmos vistos.

Formato geral da estrutura condicional `if/else` em C++:

```
if (condição)
  comando;
else
  comando;
```

Quando mais de um comando deve ser realizado em uma instrução condicional, eles devem estar delimitados por chaves (**{ }**).

Exemplo de Algoritmo:

```
Algoritmo
     declare A, B, C numérico
     leia A, B, C
     se A + B < C
          então escreva "MENSAGEM"
     fim se
fim Algoritmo
```

Exemplo C++:

```
#include <iostream.h>

void main()
{
  int A, B, C;
  cin >> A >> B >> C;
  // estrutura condicional simples
  if (A + B < C)
    cout << "MENSAGEM";
}
```

Operadores

Um operador executa uma função sobre um, dois ou três operandos. Um operador que exige um operando é denominado **operador unário**. Um operador que requer dois operadores é denominado **operador binário**.

Nas linguagens C/C++ existe apenas um operador que trabalha com três operandos, denominado **operador ternário**. (`?:`). Este operador é uma versão simplificada da instrução `if-else`.

Por exemplo, na expressão:

```
max = (a > b) ? a : b;
```

a variável que contém o maior valor numérico entre `a` e `b` será atribuída a `max`.

Operadores Aritméticos

As linguagens C/C++ suportam vários operadores aritméticos para todos os tipos inteiros e de ponto flutuante.

A tabela seguinte resume estes operadores:

Operador	Uso	Descrição
`++`	`op++`	*Incrementa* `op` *em 1 unidade; avalia o valor de* `op` *antes de incrementá-lo.*
`++`	`++op`	*Incrementa* `op` *em 1 unidade; avalia o valor de* `op` *depois de incrementá-lo.*
`--`	`op--`	*Decrementa* `op` *em 1 unidade; avalia o valor de* `op` *antes de incrementá-lo.*
`--`	`--op`	*Decrementa* `op` *em 1 unidade; avalia o valor de* `op` *depois de incrementá-lo.*
`+`	`op1 + op2`	*Adiciona* `op1` *e* `op2`*.*
`-`	`op1 - op2`	*Subtrai* `op2` *de* `op1`*.*
`*`	`op1 * op2`	*Multiplica* `op1` *por* `op2`*.*
`/`	`op1 / op2`	*Divisão de* `op1` *por* `op2`*.*
`%`	`op1 % op2`	*Resto inteiro da divisão de* `op1` *por* `op2`*.*

Operadores Relacionais e Condicionais

Um operador relacional compara dois valores e determina o relacionamento entre eles.

A próxima tabela resume os operadores relacionais:

Operador	Uso	Descrição
`>`	`op1 > op2`	*Retorna* **`true`** *(***1***) se operador* `op1` *for maior que* `op2`*.*
`>=`	`op1 >= op2`	*Retorna* **`true`** *(***1***) se operador* `op1` *for maior ou igual ao* `op2`*.*

`<`	`op1 < op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *for menor que* `op2`*.*
`<=`	`op1 <= op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *for menor ou igual ao* `op2`*.*
`==`	`op1 == op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *for igual ao* `op2`*.*
`!=`	`op1 != op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *for diferente do* `op2`*.*

Operadores Relacionais e Condicionais

Um operador relacional compara dois valores e determina o relacionamento entre eles.

A próxima tabela resume os operadores relacionais:

Operador	*Uso*	*Descrição*
`>`	`op1 > op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *for maior que* `op2`*.*
`>=`	`op1 >= op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *for maior ou igual ao* `op2`*.*
`<`	`op1 < op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *for menor que* `op2`*.*
`<=`	`op1 <= op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *for menor ou igual ao* `op2`*.*
`==`	`op1 == op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *for igual ao* `op2`*.*
`!=`	`op1 != op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *for diferente do* `op2`*.*

Operadores relacionais são freqüentemente usados com operadores condicionais para construir expressões complexas.

A próxima tabela resume os operadores condicionais:

Operador	Uso	Descrição
`&&`	`op1 && op2`	*Retorna* **`true`** (**1**) *se operador* `op1` *e* `op2` *forem ambos* **`true`**; *condicionalmente avalia* `op2`.
`||`	`op1 || op2`	*Retorna* **`true`** (**1**) *se operador* `op1`, `op2`, *ou ambos forem* **`true`**; *condicionalmente avalia* `op2`.
`!`	`!op`	*Retorna* **`true`** *se operador* `op`, *for* **`false`**.
`&`	`op1 & op2`	*Se os operadores forem números inteiros realiza operação* **AND** *bit a bit.*
`|`	`op1 | op2`	*Se os operadores forem números inteiros realiza operação* **OR** *bit a bit.*
`^`	`op1 ^ op2`	*Retorna* **`true`** *se operador* `op1` *ou* `op2` *for* **`true`**, *mas não ambos.*
`~`	`~op`	*Retorna o complemento de um do número representado por* `op`.

Operadores de Deslocamento de bits

Um operador de deslocamento manipula os dados por descolamento nos bits da variável.

Estes operadores estão resumidos na tabela a seguir:

Operador	Uso	Descrição
`<<`	`op1 << op2`	*Desloca a esquerda os bits de* `op1` *pela distância definida por* `op2`; *são acrescentados bits* **`0`** *à direita de* `op1`.

`>>`	`op1 >> op2`	*Desloca a direita os bits de `op1` pela distância definida por `op2`; são acrescentados bits **0** à esquerda de `op1`.*
`>>>`	`op1 >>> op2`	*Desloca a direita os bits de `op1` pela distância definida por `op2`; são acrescentados bits **0** à esquerda de `op1`. Não considera o bit de sinal.*

Para exemplificar a utilização dos operadores de deslocamento de bits, considere a expressão:

```
cout << (13 >> 1);
```

Esta instrução está apresentando na saída padrão o valor da operação `13 >> 1`, ou seja, o valor `13` com o deslocamento de `1` bit para a direita.

Como `13` em binário corresponde ao valor `1101`, o resultado desta operação será o binário `110`, que corresponde ao valor `6` decimal.

Estrutura de repetição

Nos algoritmos vistos até aqui, a estrutura de repetição é delimitada pelo comando `repita` e pela expressão `fim repita` e a interrupção é feita através do comando `interrompa`.

A condição de interrupção que deve ser satisfeita é representada por uma expressão lógica e a estrutura de interrupção pode apresentar três formas de interrupção: no Início da repetição, no meio da repetição ou no final da repetição.

Esta forma de estruturação pode ser diretamente mapeada para as estruturas de repetição da linguagem C++.

Entretanto, a linguagem C++ apresenta diversas variantes das estruturas de repetição que permitem otimizar o código.

Inicialmente, será apresentado um mapeamento direto para os algoritmos estudados. Em seguida serão apresentadas as variantes possíveis.

Formato em algoritmo para a interrupção de início da estrutura:

```
repita
     se condição
          então interrompa
     fim se
     seqüência de comandos
fim repita
```

Mapeamento possível em C++:

```
while(true)
{
  if (condição)
    break;
  // seqüência de comandos
}
```

Exemplo de Algoritmo:

```
Algoritmo
     declare PAR, SOMA numérico
     SOMA ← 0
     PAR ← 100
     repita
          se PAR > 200
               então interrompa
          fim se
          SOMA ← SOMA + PAR
          PAR ← PAR + 2
     fim repita
     escreva SOMA
fim Algoritmo
```

Exemplo C++:

```
#include <iostream.h>

void main()
{
  int PAR, SOMA;
  SOMA = 0;
  PAR = 100;
  while(true)
  {
    if (PAR > 200)
      break;
    SOMA = SOMA + PAR;
    PAR = PAR + 2;
  }
  cout << "A Soma dos números pares de 100 a 200 e: "
       << SOMA;
}
```

Como foi explicado, a estrutura anterior fornece uma forma direta de mapeamento entre os algoritmos estudados e a linguagem C++.

Entretanto, as estruturas de repetição da linguagem podem (e devem) ser utilizadas de forma mais eficientes.

A instrução `while` consiste na palavra-chave `while` seguida de uma expressão de teste entre parênteses. Se a expressão for verdadeira, o laço é executado uma vez e a expressão de teste é avaliada novamente. Este ciclo de teste e execução é repetido até que a expressão de teste se torne falsa, então o laço termina e o controle do programa passa para alinha seguinte ao laço.

É importante lembrar que um laço em C++ também pode ser interrompido com a utilização da instrução `break`, como foi visto no primeiro exemplo.

Outra instrução de repetição importante é `for`, que consiste da palavra-chave for seguida de parênteses que contêm três expressões separadas por pontos-e-vírgulas.

A primeira das expressões é denominada **inicialização**, a segunda é chamada de **teste** e a terceira parte é conhecida como **incremento**.

Qualquer uma das três expressões pode existir instruções válidas da linguagem.

Em sua forma mais simples, a inicialização é uma instrução de atribuição e é sempre executada uma única vez antes de o laço ser iniciado.

O teste é uma condição avaliada como verdadeira ou falsa e controla o laço. Esta expressão é avaliada toda vez que o laço é iniciado ou reiniciado.

O incremento geralmente define a maneira pela qual a variável de controle será alterada cada vez que o laço for repetido. Essa expressão é executada sempre, imediatamente após a execução do corpo do laço.

Exemplo C++:

```
// Imprime a tabuada de 5
#include <iostream.h>
#include <iomanip.h>
```

```
void main()
{
  int i;
  for(i=1; i<=10; i++)
    cout << "\n" << setw(3) << i << setw(5) << (i*5);
}
```

Qualquer uma das expressões de um laço for pode conter várias instruções separadas por vírgulas. A vírgula, neste caso, é um operador C++ que ordena a ordem de execução das instruções. Um par de expressões separadas por vírgula é avaliado da esquerda para a direita.

Exemplo C++:

```
// Imprime os números de 0 a 98 de 2 em 2
#include <iostream.h>

void main()
{
  for(int i=0, int j=0; (i+j)<100; i++, j++)
    cout << "\n" << (i+j);
}
```

A terceira e última estrutura de laço em C++ é o laço `do-while`. Este é similar ao laço `while` e representa uma repetição com condição de parada no fim da estrutura.

Exemplo de Algoritmo:

```
Algoritmo
     declare PAR, SOMA numérico
     SOMA ← 0
     PAR ← 100
     repita
          SOMA ← SOMA + PAR
          PAR ← PAR + 2
          se PAR > 200
               então interrompa
          fim se
     fim repita
     escreva SOMA
fim Algoritmo
```

Exemplo C++:

```
#include <iostream.h>

void main()
{
  int PAR, SOMA;
```

```
  SOMA = 0;
  PAR = 100;
  do
  {
    SOMA = SOMA + PAR;
    PAR = PAR + 2;
  } while(PAR <= 200);
  cout << "A Soma dos números pares de 100 a 200 e: "
       << SOMA;
}
```

Variáveis Compostas Unidimensionais

Conjuntos de dados referenciados por um mesmo nome e que necessitam de somente um índice para que seus elementos sejam endereçados são ditos compostos unidmensionais.

Novamente, a declaração de variáveis compostas unidimensionais em algoritmo é feita através da seguinte declaração:

```
declare identificador [li:ls] nome-do-tipo
```

Nas linguagens C/C++ o limite inferior de uma variável composta é sempre 0. Desta forma, a declaração exige apenas o total de elementos. Portanto, a faixa de intervalo do índice varia de 0 até total de elementos menos 1.

```
int identificador[ls+1];
```

Exemplo de Algoritmo:

Preencher uma variável do tipo numérico contendo 6 posições com o valor 2:

```
Algoritmo
     declare N[1:6], Contador numérico
     Contador ← 1
     repita
          se Contador > 6
               então interrompa
          fim se
          N[Contador] ← 2
          Contador ← Contador + 1
     fim repita
fim Algoritmo
```

Exemplo C++:

```
#include <iostream.h>

void main()
{
  int N[6], Contador;
  Contador = 0;
  while(true)
  {
    if (Contador > 5)
      break;
    N[Contador] = 2;
    Contador = Contador + 1;
  }
}
```

Este tipo de código é mais bem resolvido com a instrução `for`, conforme o exemplo a seguir:

Exemplo C++:

```
#include <iostream.h>

void main()
{
  int N[6], Contador;
  for(Contador=0; Contador < 6; Contador++)
    N[Contador] = 2;
}
```

É possível fornecer valores a cada elemento da matriz na mesma instrução de sua declaração.

O próximo exemplo calcula o número de dias transcorridos a partir do início do ano até a data especificada pelo usuário. O programa verifica se o ano é ou não bissexto.

Exemplo C++:

```
#include <iostream.h>

void main()
{
  int dmes[12]={31,28,31,30,31,30,31,31,30,31,30,31};
  int dia, mes, ano;
  cout << "Digite a data no formato DD/MM/AAAA: ";
  {
    char ch;
    cin >> dia >> ch >> mes >> ch >> ano;
  }
```

```
  if (ano%4==0 && ano%100 || ano%400==0)
    dmes[1]=29;
  int total=dia;
  for(int i=0; i<mes-1; i++)
    total += dmes[i];
  cout << "Total de dias transcorridos: " << total;
}
```

A instrução de definição de uma matriz inicializada pode suprimir a dimensão da matriz, restando apenas um par de colchetes vazios:

```
int dmes[]={31,28,31,30,31,30,31,31,30,31,30,31};
```

Se nenhum número for fornecido para dimensionar a matriz, o compilador contará o número de valores inicializadores e o fixará como dimensão da matriz.

Variáveis Compostas Multidimensionais

Os elementos de uma matriz podem ser de qualquer tipo, incluindo outras matrizes.

De certa forma, em C++, o termo duas dimensões não faz sentido, pois todas as matrizes são armazenadas na memória de forma linear. Assim, esta expressão representará matrizes em que os elementos são outras matrizes.

Com dois pares de colchetes obtém-se uma matriz de duas dimensões e com cada par de colchetes adicionais obtém-se matrizes com uma dimensão a mais.

Exemplo de Algoritmo:

Ler duas matrizes A e B de ordem 10x10 e apresentar o resultado da operação A+B:

```
Algoritmo
     declare A[1:10,1:10],B[1:10,1:10] numérico
     declare i, j numérico
     i ← 1
     repita
          se i > 10
               então interrompa
          fim se
          j ← 1
          repita
               se j > 10
                    então interrompa
```

```
                    fim se
                    leia A[i, j], B[i, j]
                    j ← j + 1
               fim repita
               i ← i + 1
          fim repita
          i ← 1
          repita
               se i > 10
                    então interrompa
               fim se
               j ← 1
               repita
                    se j > 10
                         então interrompa
                    fim se
                    escreva A[i, j] + B[i, j]
                    j ← j + 1
               fim repita
               i ← i + 1
          fim repita
fim Algoritmo
```

Exemplo C++:

```
#include <iostream.h>

void main()
{
  int A[10][10], B[10][10];
  int i, j;
  for(i=0; i<10; i++)
    for(j=0; j<10; j++)
    {
      cout << "\n\nA[" << i << "][" << j << "]: ";
      cin >> A[i][j];
      cout << "\nB[" << i << "][" << j << "]: ";
      cin >> B[i][j];
    }
  for(i=0; i<10; i++)
    for(j=0; j<10; j++)
      cout << "\nA[" << i << "][" << j << "]+"
           << "B[" << i << "][" << j << "]: "
           << A[i][j] + B[i][j];
}
```

Variáveis Compostas Heterogêneas

As variáveis compostas heterogêneas são conjuntos de dados logicamente relacionados, mas de tipos diferentes (numérico, literal, lógico) e, nas linguagens C/C++ são conhecidas como **estruturas**.

A linguagem C++ também oferece outros dois mecanismos para a criação de variáveis compostas heterogêneas: **uniões** e **classes**.

A sintaxe para a criação de estruturas é a mesma da utilizada para criar classes. Assim, aprender a lidar com estruturas é o caminho para entender classes e objetos.

Por meio da palavra-chave `struct` define-se um novo tipo de dado. Definir um tipo de dado significa informar ao compilador o seu nome, o seu tamanho em bytes e o formato em que ele deve ser armazenado e recuperado da memória.

Como foi visto, a declaração de variáveis compostas heterogêneas em algoritmo é feita através da seguinte declaração:

```
declare identificadores registro (componentes)
```

Em C/C++ esta declaração é realizada da seguinte forma:

```
struct Nome_da_estrutura {
  tipo identificador1;
  tipo identificador2;
  ...
  tipo identificadorN;
} variáveis;
```

Exemplo de Algoritmo:

```
Algoritmo
     declare Tabela[1:5,1:3] numérico
     declare Imóvel registro (Identific numérico,
                              Imposto numérico,
                              MesesAt numérico)
     declare i, Multa numérico
     {leitura da Tabela}
     repita
          leia Imóvel
          se Imóvel.Identific = 0
               então interrompa
          fim se
          i ← 6
          repita
               i ← i - 1
               se Imóvel.Imposto ≥ Tabela[i,1] ou I=1
                    então interrompa
```

```
                fim se
            fim repita
            se Imóvel.Imposto ≥ Tabela[i,1]
                então Multa ← Tabela[i,3] x
                      Imóvel.Imposto / 100 x
                      Imóvel.MesesAt
            fim se
            escreva Imóvel, Multa
        fim repita
fim Algoritmo
```

Exemplo C++:

```
#include <iostream.h>

void main()
{
  int Tabela[5][3];
  struct {int Identific;
          float Imposto;
          int MesesAt;} Imovel;
  int i;
  float Multa;
  // Leitura da Tabela
  while(true)
  {
     cin >> Imovel.Identifc
         >> Imovel.Imposto
         >> Imovel.MesesAt;
     if (Imovel.Identifc == 0)
       break;
     i = 5;
     while(true)
     {
        i = i - 1;
        if ((Imovel.Imposto >= Tabela[i][0]) || (i==1))
          break;
     }
     if (Imovel.Imposto >= Tabela[i][0])
       Multa = Tabela[i][3] * Imovel.Imposto / 100
               * Imovel.MesesAt;
     cout << "\nImovel: " << Imovel
          << " == Multa: " << Multa;
  }
}
```

Bibliografia

DEITEL, H. M. & DEITEL, P. J. *C++ Como Programar.* 3ª. edição. Porto Alegre : Bookman, 2001.

FARRER, H. et. alli. *Algoritmos Estruturados.* 3ª. edição. Rio de Janeiro : LTC – Livros Técnicos e Científicos Editora S/A, 1999.

MIZRAHI, V. V. *Treinamento em Linguagem C++ - Modulo 1.* São Paulo : Makron Books do Brasil Editora Ltda, 1994.

SCHILDT, H. *C Completo e Total.* São Paulo : Makron Books do Brasil Editora Ltda, 1991.