Algoritmo

NOTAS DE AULA

Rafael Dias Ribeiro
rdribeiro@processware.com.br

PROCESSWARE
Consultoria e Treinamento

Conteúdo retirado do site:

http://www.cplusplus.com

em 20/08/2006.

Estruturas

Um programa geralmente não é limitado a uma sequência linear de instruções.

Durante o processo, pode bifurcar, repetir código, ou tomar decisões. Por esse motivo, C++ provê estruturas de controle que servem para especificar o que precisa ser feito para executar nosso programa.

Com a introdução das sequências de controle, teremos que introduzir um novo conceito: o *bloco de instruções*.

Um bloco de instruções é um grupo de instruções separados por pontos-e-vírgulas (;), mas agrupados em um bloco delimitado por sinais de chaves: { e }.

A maioria das estruturas de controle permitem um *conteúdo* genérico como parâmetro, isso se refere a uma única instrução ou a um bloco de instruções, como quisermos.

Se quisermos que o conteúdo seja uma única instrução, não precisamos colocá-la entre chaves ({}). Se quisermos que o conteúdo seja mais que uma única instrução, precisamos colocá-la entre chaves ({}) formando um bloco de instruções

Estrutura condicional: *if* e *else*

É usada para executar uma instrução ou bloco de instruções somente se uma condição for satisfeita. Sua forma é:

if (*condição*) *conteúdo*

Onde *condição* é a expressão que está sendo avaliada. Se a condição for true, o *conteúdo* é executado. Se for false, o *conteúdo* é ignorado (não é executado) e o programa continua na próxima instrução depois da estrutura condicional.

Por exemplo, o seguinte fragmento de código imprime x e 100 somente se o valor guardado na variável x for igual a 100:

```
if (x == 100)
cout << "x e 100";
```

Se quisermos que mais de uma única instrução seja executada no caso da *condição* ser true, precisamos especificar um *bloco de instruções* usando chaves { }:

```
if (x == 100)
 {
 cout << "x e ";
 cout << x;
 }
```

Estrutura condicional: *if* e *else*

Podemos especificar também o que queremos que aconteça caso a condição não seja satisfeita usando a palavra-chave *else*. Sua forma usada em conjunto com if é:

```
if (condição)
conteúdo1
else
conteúdo2
```

Por exemplo:

```
if (x == 100)
cout << "x e 100";
else
cout << "x nao e 100";
```

imprime na tela x e 100 se x for igual a 100, mas se não for –e somente se não for- imprime x não e 100.

Estrutura condicional: *if* e *else*

Podemos especificar também o que queremos que aconteça caso a condição não seja satisfeita usando a palavra-chave *else*. Sua forma usada em conjunto com if é:

```
if (condição)
conteúdo1
else
conteúdo2
```

Por exemplo:

```
if (x == 100)
cout << "x e 100";
else
cout << "x nao e 100";
```

imprime na tela x e 100 se x for igual a 100, mas se não for –e somente se não for- imprime x não e 100.

Estrutura condicional: *if* e *else*

As estruturas *if* + *else* podem ser concatenadas com a intenção de verificar um intervalo de valores.

O seguinte exemplo mostra seu uso dizendo se o valor presente guardado em x é positivo, negativo ou nenhum dos anteriores, ou seja, igual a zero.

```
if (x > 0)
cout << "x e positivo";
else if (x < 0)
cout << "x e negativo";
else
cout << "x e 0";
```

Lembre-se que caso queiramos que mais de uma única instrução seja executada, precisamos agrupá-las em um *bloco de instruções* usando chaves { }.

Estruturas Repetitivas ("loops")

O loop *while*.

Seu formato é:

while (*expressão***)**
conteúdo

E sua função é simplesmente repetir o *conteúdo* enquanto a *expressão* for verdadeira.

Lembre-se que caso queiramos que mais de uma única instrução seja executada, precisamos agrupá-las em um bloco de instruções usando chaves **{ }**.

```
while (expressão)
{

Bloco de conteúdo

}
```

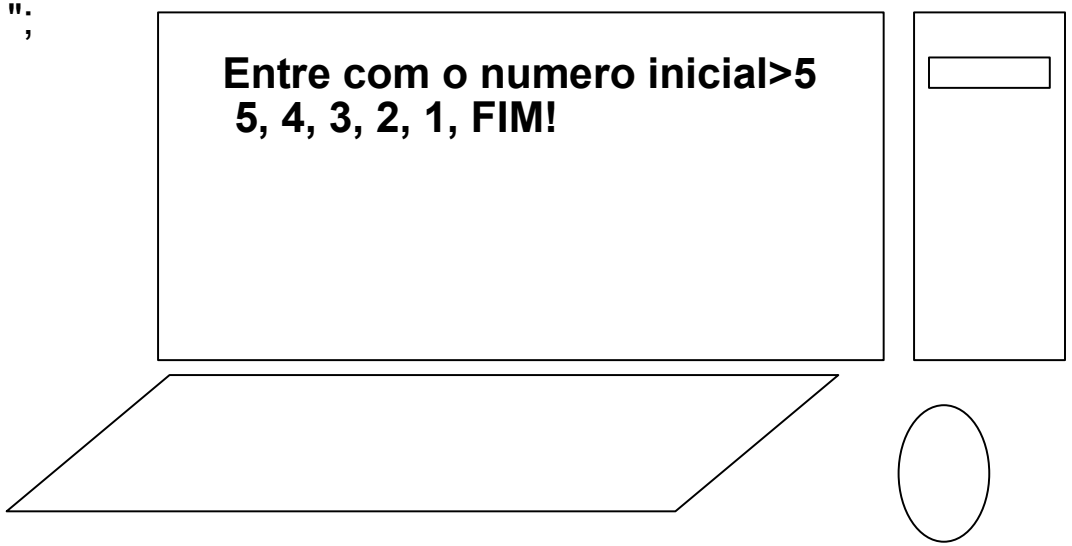
Estruturas Repetitivas ("loops")

O loop *while*.

// contagem regressiva customizada usando while

```
#include<iostream.h>
main ( )
{
int n;
cout << "Entre com o numero inicial > ";
cin >> n;
        while (n>0)
        {
        cout << n << ", ";
        - -n;
        }
cout << "FIM!";

}
```



Estruturas Repetitivas (“loops”)

O loop *while*.

// contagem regressiva customizada usando while

```
#include<iostream.h>
main ( )
{
int n;
cout << "Entre com o numero inicial > ";
cin >> n;
        while (n>0)
        {
        cout << n << ", ";
        - -n;
        }
cout << “FIM!";

}
```

Estruturas Repetitivas ("loops")

O loop *while*.

*Quando o programa inicia, é pedido ao usuário que insira um número inicial para a contagem regressiva. Então quando o loop while inicia, se o valor entrado pelo usuário satisfizer a condição **n>0** (de que **n** é maior que **0**), o bloco de instruções seguinte será executado um número indefinido de vezes enquanto a condição **(n>0)** permanecer verdadeira.*

*Todo o processo no programa acima pode ser interpretado de acordo com o seguinte script: iniciando em **main**:*

***1.** O usuário atribui um valor para **n**.*

***2.** A instrução while checa se **(n>0)**. Nesse ponto há duas possibilidades:*

- ***true:** executa o conteúdo (passo **3,**)*
- ***false:** pula o conteúdo. O programa segue para o passo **5.***

***3.** Executa o conteúdo:*
cout << n << ", ";
--n;
*(imprime **n** na tela e decrementa 1 de **n**).*

4.** Fim do bloco. Retorna automaticamente ao passo **2.

***5.** Continua o programa depois do bloco: imprime **FIM!** e termina o programa.*

Estruturas Repetitivas (“loops”)

O loop *while*.

Precisamos considerar que o loop precisa terminar alguma hora, sendo assim, dentro do bloco de instruções (o conteúdo do loop) precisamos prover algum método que força a condição a se tornar falsa em algum momento, ou em caso contrário, o loop continuará se repetindo para sempre.

Nesse caso nós incluímos ***--n;*** *que faz com que a condição se torne* ***false*** *depois de algumas repetições de loop: quando* ***n*** *se torna* ***0****, que é quando nossa contagem regressiva termina.*

É claro que essa é uma ação bastante simples para nosso computador que toda a contagem regressiva é feita de maneira instantânea sem um atraso prático entre os números.

Estruturas Repetitivas ("loops")

O loop *do-while*.

Formato:

do
conteúdo
while (*condição***);**

Sua funcionalidade é exatamente a mesma que a do loop *while*, com exceção que a *condição* do *do-while* é avaliada depois da execução da *conteúdo* ao invés de antes, garantindo pelo menos uma execução do *conteúdo* mesmo se a *condição* nunca for satisfeita.

Lembre-se que caso queiramos que mais de uma única instrução seja executada, precisamos agrupá-las em um bloco de instruções usando chaves **{ }**.

do {
Bloco de conteúdo
} **while (**expressão**)**

Estruturas Repetitivas ("loops")

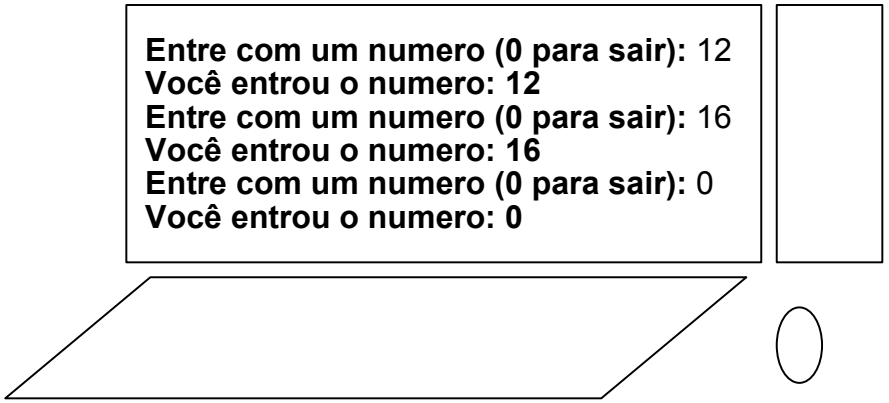
O loop *do-while*.

```
// repetidor de número

#include <iostream.h>
main ( )
{
int n;

      do
      {
      cout << "Entre com um numero (0 para sair): ";
      cin >> n;
      cout << "Você entrou o numero: " << n << "\n";
      } while (n != 0);

}
```



Estruturas Repetitivas (“loops”)

O loop *do-while*.

O loop *do-while* geralmente é usado quando a condição que determina se chegou ao fim está dentro do conteúdo do loop, como no caso anterior, no qual a entrada do usuário dentro do bloco de instruções é o que determinar o fim do loop.

Se você nunca entrar com o valor 0 no exemplo anterior, o loop não terminará nunca.

Estruturas Repetitivas ("loops")

O loop *for:*

Seu formato é:

```
for (inicialização; condição; incremento)
 conteúdo;
```

Lembre-se que caso queiramos que mais de uma única instrução seja executada, precisamos agrupá-las em um bloco de instruções usando chaves **{ }**.

```
for (inicialização; condição; incremento)
{
Bloco de conteúdo;
}
```

E sua principal função é repetir o *conteúdo* enquanto a *condição* permanecer verdadeira, como no loop *while*.

Mas, além disso, o for provê lugares para especificar a instrução de *inicialização* e a instrução de *incremento*. Então esse loop é projetado especialmente para realizar uma ação repetitiva com um contador.

Estruturas Repetitivas ("loops")

O loop *for:*

Funciona da seguinte maneira:

1, a *inicialização* é executada. Geralmente é a configuração de um valor inicial para um contador. É executado somente uma vez.

2, a *condição* é checada, se for true o loop continua, caso contrário o loop termina e o *conteúdo* é pulado.

3, o *conteúdo* é executado. Como sempre, pode ser tanto uma única instrução ou um bloco de instruções colocadas entre chaves { }.

4, finalmente, o que quer que seja que estiver especificado no campo de *incremento*, é executado e o loop volta ao passo 2.

Aqui está um exemplo de contagem regressiva usando o loop *for*.

Estrutura Seletiva

Switch :

A sintaxe da instrução *switch* é um pouco peculiar. Seu objetivo é checar diversos valores constantes possíveis para uma expressão, algo parecido com o que fizemos no início dessa seção com a junção de diversos comandos *if* e *else if*. Sua forma é a seguinte:

```
switch (expressão) {
  case constante1:
    bloco de instruções 1
    break;
  case constante2:
    bloco de instruções 2
    break;
  .
  .
  .
  default:
    bloco de instruções padrão
  }
```

Estrutura Seletiva

Switch :

Funciona da seguinte maneira:

o switch avalia a expressão e checa se é equivalente à constante1, e se for, executa o bloco de instruções 1 até achar a palavra-chave break, então o programa irá pular para o fim da estrutura seletiva switch.

Se a expressão não for igual à constante1, irá checar se a expressão é equivalente à constante2. E se for, irá executar o bloco de instruções 2 até achar a palavra-chave break.

Finalmente, de o valor da expressão não foi equivalente à nenhuma constante especificada anteriormente (você pode especificar quantos comandos case como valores você quiser), o programa irá executar as instruções incluídas na seção default: se existir, já que é opcional.

Estrutura Seletiva

Switch :

Ambos os fragmentos de código seguintes são equivalentes:

```
switch (x) {

case 1:
cout << "x e 1";
break;

case 2:
cout << "x e 2";
break;

default:
cout << "valor de x desconhecido";
}
```

```
if (x == 1) {
cout << "x e 1";
}
else if (x == 2)
{
cout << "x e 2";
}
else
{
cout << "valor de x desconhecido";
}
```

Estrutura Seletiva

Switch :

Note a inclusão da instrução break no final de cada bloco. Isso é necessário porque se, por exemplo, não a incluíssemos no bloco de instruções 1, o programa não iria pular para o final do bloco seletivo switch (}) e continuaria executando o resto dos blocos de instruções até que a instrução break aparecesse pela primeira vez ou chegasse ao fim do bloco seletivo switch.

Isso faz com que seja desnecessário incluir chaves { } em cada um dos casos, e também pode ser útil para executar o mesmo bloco de instruções para valores possivelmente diferentes da expressão avaliada.

Estrutura Seletiva

Switch

Por Exemplo:

```
switch (x) {
case 1:
case 2:
case 3:
cout << "x e 1, 2 ou 3";
break;
default:
cout << "x nao e 1, 2 nem 3";
}
```

Note que o switch só pode ser usado para comparar uma expressão com constantes diferentes. Sendo assim, não podemos colocar variáveis (case (n*2):) ou intervalos (case (1..3):) porque não são constantes válidas.