Linguagem de Programação C++

Universidade Federal de Santa Catarina

Departamento de Engenharia Elétrica, CTC

Prof. Eduardo Augusto Bezerra

"Esse material foi adaptado a partir de um website não identificado."

Estruturas de controle

Comando condicional: *if* e *else*

Utilizado para executar uma instrução ou bloco de instruções, **se** uma determinada condição for satisfeita. Formato da instrução:

```
if (condição) comandos
```

onde *condição* é a expressão a ser avaliada. Se a condição for verdadeira (true), os ***comandos*** são executados. Se for falsa (false), os *comandos* são ignorados (não são executados) e o programa continua seu fluxo logo após o término do bloco do comando condicional.

Por exemplo:

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

apresenta na tela **`x eh 100`** se x for ighal a 100, caso contrário, apresenta na tela**`x nao eh 100`**.

Comandos de repetição (laços)

O uso de *laços* objetiva repetir a execução de *comandos* um determinado número de vezes.

O laço *while*.

Formato:

```
while (expressão) comandos
```

Por exemplo, programa que realiza uma contagem regressiva:

```
// contagem regressiva com while
#include <iostream>
using namespace std;

int main ()
{
```

```
Entre com um valor
inicial para a contagem >
8
8, 7, 6, 5, 4, 3, 2, 1,
FIM!
```

```
  int n;
  cout << "Entre com um valor inicial para a contagem > ";
  cin >> n;
  while (n>0) {
    cout << n << ", ";
    --n;
  }
  cout << "FIM!";
  return 0;
}
```

O laço *do-while*.

Formato:

```
do comandos while (condição);
```

A diferença para o laço *while* é que no *do-while* a condição é avaliada após a execução dos comandos, possibilitando assim que os comandos possam ser executados, pelo menos, uma vez antes da verificação se a condição é verdadeira ou falsa. Por exemplo, o programa a seguir apresenta os valores fornecidos pelo usuário até que seja fornecido o valor zero.

```
// apresenta valores
#include <iostream>
using namespace std;

int main ()
{
  unsigned long n;
  do {
    cout << "Entre com um numero (0 para encerrar): ";
    cin >> n;
    cout << "Entrou com: " << n << "\n";
  } while (n != 0);
  return 0;
}
```

```
Entre com um numero (0 para encerrar): 12345
Entrou com: 12345
Entre com um numero (0 para encerrar): 160277
Entrou com: 160277
Entre com um numero (0 para encerrar): 0
Entrou com: 0
```

O laço *for*.

Formato:

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

Essa instrução repete os *comandos* enquanto a *condição* for verdadeira, de forma semelhante ao *while*. Porém, adicionalmente, a instrução **for** possibilita definir uma *inicialização* e um *incremento*. Dessa forma, esse tipo de laço é adequado para realizar repetições relacionadas a contagens.

Exemplo:

```
// contagem regressiva
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    cout << n << ", ";
  }
  cout << "FIRE!";
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

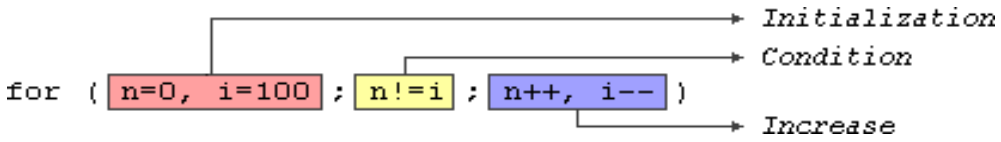
Os campos *inicialização* e *incremento* são opcionais. Porém, os caracteres "ponto-e-vírgula" entre eles são obrigatórios. Por exemplo, pode ser utilizado: **for (;n<10;)** caso não seja necessária a

inicialização e *incremento*; ou **`for (;n<10;n++)`** se for necessário incluir o campo de *incremento* mas não a *inicialização*.

Opcionalmente, utilizando a vírgula (**`,`**) pode ser especificado mais de uma instrução em qualquer um dos campos de um laço **`for`**. Por exemplo, supor a necessidade de inicializar mais de uma variável:

```
for ( n=0, i=100 ; n!=i ; n++, i-- )
{
  // instruções a serem executadas...
}
```

Esse laço vai executar 50 vezes se as variáveis **`n`** ou **`i`** não forem modificadas pelas instruções dentro do laço:



`n` inicia com **`0`** e **`i`** com **`100`**, e a condição é **`(n!=i)`** (ou seja, **`n`** diferente de **`i`**). Como a variável **`n`** é incrementada de uma unidade, e **`i`** é decrementada de uma unidade, a condição do laço vai se tornar `falso` após a repetição de número 50 do laço, ou seja, quando **`n`** e **`i`** forem iguais a 50.

Bifurcação e controle de desvios.

A instrução *break* .

Com o uso de *break* é possível sair de um laço, mesmo que a condição não tenha sido satisfeita. Pode ser utilizado, por exemplo, para sair de um laço infinito. Por exemplo, o código a seguir interrompe uma contagem regressiva antes do término previsto (um dos motores do foguete falhou!):

```
// exemplo de uso de break
#include <iostream>
using namespace std;

int main ()
{
  int n;
  for (n=10; n>0; n--) {
    cout << n << ", ";
    if (n==3)
    {
      cout << "lançamento abortado!";
      break;
    }
  }
  return 0;
}
```

`10, 9, 8, 7, 6, 5, 4, 3, lançamento abortado!`

A instrução *continue*.

A instrução *continue* faz com que o programa ignore o restante das instruções de um laço, como se o final do bloco de instruções tivesse sido atingido, forçando assim o processo a passar para a próxima comparação. O exemplo a seguir salta o número 5 da contagem regressiva:

```
// exemplo de continue
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    if (n==5) continue;
```

`10, 9, 8, 7, 6, 4, 3, 2, 1, FIRE!`

```
    cout << n << ", ";
  }
  cout << "FIRE!";
  return 0;
}
```

A instrução *goto* .

Essa instrução é utilizada para realizar um desvio incondicional para uma determinada instrução do programa. Trata-se de uma instrução que deve ser evitada, pois ignora qualquer tipo de encadeamento de instruções (ex.laços dentro de laços).

O destino do desvio é identificado por um label, que é usado como um argumento pela instrução goto.

Essa instrução não possui nenhuma utilidade em programação estruturada ou orientada a objetos, e é utilizada, basicamente, por desenvolvedores de programas em linguagem assembly. Exemplo de uso de **goto** em um laço:

```
// exemplo de laço com goto
#include <iostream>
using namespace std;

int main ()
{
  int n=10;
  repeticao:
  cout << n << ", ";
  n--;
  if (n>0) goto repeticao;
  cout << "FIRE!";
  return 0;
}
```

10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!

A função *exit*.

A função *exit* é utilizada para terminar a execução de um programa, informando um código para quem chamou esse programa (normalmente o sistema operacional):

```
void exit (int código de saída);
```

O *código de saída* é utilizado por alguns sistemas operacionais. Por convenção, o *código de saída* 0 significa que o programa terminou normalmente. Qualquer outro valor significa que um erro ocorreu.

A estrutura de seleção: *switch*.

O objetivo da instrução *switch* é verificar qual o valor constante foi o resultado do processamento de uma determinada expressão. O funcionamento do *switch* é similiar ao de diversas instruções *if* e *else if* encadeadas. O formato da instrução é o seguinte:

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

Ambos fragmentos de código são equivalentes:

Exemplo de *switch*

```
switch (x) {
  case 1:
    cout << "x eh 1";
    break;
  case 2:
    cout << "x eh 2";
    break;
  default:
    cout << "valor de x eh desconhecido";
  }
```

***if-else* equivalente**

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

A instrução **`switch`** não aceita como argumento uma espressão que necessite ser processada para descobrir o valor resultante. Assim, a instrução **`switch`** não aceita variáveis (**`case (n*2):`**) ou intervalos (**`case (1..3):`**) como argumentos, uma vez que não são constantes válidas.

Para esses casos, usar **`if`** e **`else if`** encadeados.