Capítulo 2 – Estrutura de Controle

Índice

Capítulo 2 – Estrutura de Controle

Índice

2.1 Introdução

- Antes de escrever um programa:
  - Tenha um completo entendimento do problema
  - Planeje com cuidado sua abordagem para resolvê-lo
- Enquanto estiver escrevendo um programa:
  - Saiba quais "blocos de construção" estão disponíveis
  - Use bons princípios de programação

2.2 Algoritmos

- Todos os problemas computacionais
  - Podem ser solucionados ao executar uma série de ações em uma ordem específica
- Algoritmo
  - Um procedimento determinando:
    - As ações a serem executadas
    - A ordem em que estas ações deverão ser executadas
- Controle do programa
  - Especifica a ordem pela qual os comandos deverão ser executados

2.3 Pseudocódigo

- Pseudocódigo
  - Linguagem artificial e informal usada ara desenvolver algoritmos
  - Similar à linguagem natural
  - Não é realmente executado em computadores
  - Nos permite pensar o programa antes de escrever o seu código
  - Fácil de converter em um programa correspondente em C++
  - Consiste apenas de comandos executáveis

2.4 Estruturas de Controle

- Execução seqüencial
  - Comandos são executados um após o outro na ordem em que foram escritos
- Transferência de controle
  - Quando o próximo comando for executado, não será o próximo na seqüência
- Bohm e Jacopini: todos os programas são escritos em termos de 3 estruturas de controle
  - Estrutura de seqüência
    - Inserido no C++. Programas são executados seqüencialmente por default.
  - Estruturas de seleção
    - C++ possui três tipos – **`if`**, **`if/else`**, e **`switch`**
  - Estruturas de repetição
    - C++ possui três tipos – **`while`**, **`do/while`**, e **`for`**

2.4 Estruturas de Controle

- Palavras chaves do C++
  - Não podem ser usadas como identificadores ou nomes de variáveis.

C++ Keywords

Keywords common to the C and C++ programming languages

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++ only keywords

asm	bool	catch	class	const_cast
delete	dynamic_cast	explicit	false	friend
inline	mutable	namespace	new	operator
private	protected	public	reinterpret_cast	
static_cast	template	this	throw	true
try	typeid	typename	using	virtual
wchar_t				

2.4 Estruturas de Controle

- Fluxograma (Flowcharts)
  - Representação gráfica de um algoritmo
  - Desenhado com certos símbolos de propósito específico conectados por setas chamadas linhas de fluxo (*flowlines*)
  - Símbolo Retangular (símbolo de ação)
    - Indica algum tipo de ação.
  - Símbolo oval
    - Indica começo ou fim de um programa, ou uma seção do código (círculos).
- Estruturas de controle única-entrada/única-saída
  - Conectam o ponto de saída de uma estrutura de controle com o ponto de entrada da próxima (empilhamento de estruturas de controle).
  - Torna os programas fáceis de montar.

2.5 A Estrutura de Seleção if

- Estruturas de Seleção
  - Usadas para escolher um entre cursos de ação alternativos
  - Exemplo em pseudocódigo:

    If student's grade is greater than or equal to 60
    Print "Passed"
  - Se a condição é **`true`**
    - o comando print é executado e o programa segue para o próximo comando
  - Se a condição é **`false`**
    - o comando print é ignorado e o programa segue para o próximo comando
  - Identação torna o programa mais fácil de ler
    - C++ ignora espaços em branco

2.5 A Estrutura de Seleção if

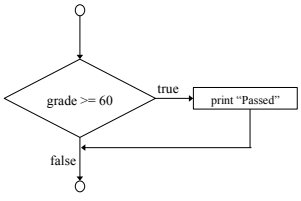
- Tradução do comando em pseudocódigo para C++:

```
if ( grade >= 60 )
   cout << "Passed";
```

- Símbolo diamante (símbolo de decisão)
  - Inidica uma decisão a ser tomada
  - Contém uma expressão que pode ser verdadeira ou falsa.
    - Teste a condição, siga o curso apropriado
- Estrutura **`if`** é uma estrutura única-entrada/única-saída

2.5 A Estrutura de Seleção if

- Fluxograma do comando em pseudocódigo



Uma decisão pode ser tomada em qualquer expressão.

zero - **`false`**

não zero - **`true`**

Exemplo:

`(3-4)` é **`true`**

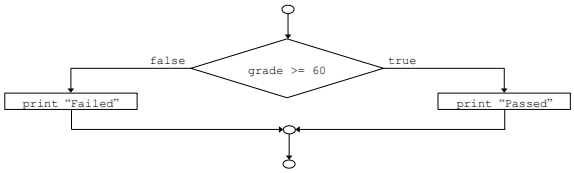
2.6 A Estrutura de Seleção `if/else`

- **`if`**
  - Somente executa uma ação se a condição for verdadeira
- **`if/else`**
  - Uma ação diferente é executada quando a condição é verdadeira e quando a condição é falsa
- Pseudocódigo

  if student's grade is greater than or equal to 60
  print "Passed"
  else
  print "Failed"
- Em C++

```
if ( grade >= 60 )
   cout << "Passed";
else
   cout << "Failed";
```

2.6 A Estrutura de Seleção `if/else`



- Operador condicional ternário (`?:`)
  - Usa três argumentos (condição, valor para **`true`**, valor para **`false`**)
- Nosso pseudocódigo poderia ser escrito como:

```
cout << ( grade >= 60 ? "Passed" : "Failed" );
```


2.6 A Estrutura de Seleção `if/else`

- Estruturas **`if/else`** aninhadas
  - Teste múltiplos casos ao colocar estruturas de seleção **`if/else`** dentro de estruturas de seleção **`if/else`**.

```
if student's grade is greater than or equal to 90
   Print "A"
else  if student's grade is greater than or equal to 80
         Print "B"
else  if student's grade is greater than or equal to 70
         Print "C"
else if student's grade is greater than or equal to 60
         Print "D"
else Print "F"
```

  - Quando uma condição é satisfeita, o resto dos comandos são ignorados

2.6 A Estrutura de Seleção `if/else`

- Estruturas **`if/else`** aninhadas
  - Teste múltiplos casos ao colocar estruturas de seleção **`if/else`** dentro de estruturas de seleção **`if/else`**.

```
if student's grade is greater than or equal to 90
   Print "A"
else
   if student's grade is greater than or equal to 80
         Print "B"
      else
      if student's grade is greater than or equal to 70
            Print "C"
         else
            if student's grade is greater than or equal to 60
               Print "D"
      else
         Print "F"
```

  - Quando uma condição é satisfeita, o resto dos comandos são ignorados

2.6 A Estrutura de Seleção `if/else`

- Grupo de comandos:
  - Conjunto de comandos entre chaves
  - Exemplo:

```
if ( grade >= 60 )
   cout << "Passed.\n";
else
{
   cout << "Failed.\n";
   cout << "You must take this course again.\n";
}
```

  - Sem as chaves, o comando

```
cout << "You must take this course again.\n";
```

  seria automaticamente executado
- Bloco
  - Grupo de comandos com declarações

2.6 A Estrutura de Seleção `if/else`

- Erros de sintaxe
  - Erros capturados pelo compilador
- Erros de lógica
  - Erros que acontecem em tempo de execução
    - Erros de lógica não fatais
      - programa roda, mas tem uma saída incorreta
    - Erros de lógica fatais
      - programa termina prematuramente

2.7 A Estrutura de Repetição while

- Estrutura de Repetição
  - O Programador especifica uma ação a ser repetida enquanto uma condição permanecer verdadeira
  - Pseudocódigo

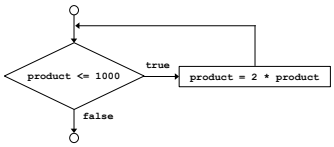
```
while there are more items on my shopping list
   Purchase next item and cross it off my list
```

  - laço **`while`** é repetido até que a condição se torna falsa.
- Exemplo

```
int product = 2;
while ( product <= 1000 )
   product = 2 * product;
```

2.7 A Estrutura de Repetição while

- Fluxograma para laço **`while`**



2.8 Formulando Algoritmos (Repetição Controlada por Contador)

- Repetição controlada por contador
  - Laço é repetido até que um contador alcança um determinado valor.
- Repetição definida
  - Número de repetições é conhecida
- Exemplo

  Uma classe de dez alunos fez um teste. As notas (inteiros entre 0 e 100) deste teste estão disponíveis. Determine a média da classe para este teste.

2.8 Formulando Algoritmos (Repetição Controlada por Contador)

- Pseudocódigo por exemplo:

  Set total to zero
  Set grade counter to one
  While grade counter is less than or equal to ten
  Input the next grade
  Add the grade into the total
  Add one to the grade counter
  Set the class average to the total divided by ten
  Print the class average

- A seguir está o código C++ para este exemplo

Índice

1. **Inicializar as Variáveis**
2. **Executar o Laço**
3. **Imprima o resultado**

```
1  // Fig. 2.7: fig02_07.cpp
2  // Class average program with counter-controlled repetition
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    int total,        // sum of grades
12        gradeCounter, // number of grades entered
13        grade,        // one grade
14        average;      // average of grades
15
16    // initialization phase
17    total = 0;                    // clear total
18    gradeCounter = 1;             // prepar
19
20    // processing phase
21    while ( gradeCounter <= 10 ) {  // loop
22       cout << "Enter grade: ";     // prompt
23       cin >> grade;               // input
24       total = total + grade;      // add grade to total
25       gradeCounter = gradeCounter + 1;  // increment counter
26    }
27
28    // termination phase
29    average = total / 10;               // integer division
30    cout << "Class average is " << average << endl;
31
32    return 0;   // indicate program ended successfully
33 }
```

O contador é incrementado cada vez que o laço é executado. Em alguma altura, o contador faz com que o laço termine.

Índice

Saída do Programa

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```


2.9 Formulando Algoritmos com Refinamento Passo-a-Passo Top-Down (Repetição Controlada por Sentinela)

- Suponha que o problema seja:

  Desenvolva um programa de média da classe que irá processar um número arbitrário de notas a cada execução.
  - Número de alunos desconhecido – como o programa saberá que deve terminar?
- Valor sentinela
  - Indica "fim da entrada dos dados"
  - Laço termina quando o sentinela é lido
  - Valor sentinela deve ser escolhido de forma a não ser confundido com um dado regular (como -1 neste caso)

2.9 Formulando Algoritmos com Refinamento Passo-a-Passo Top-Down (Repetição Controlada por Sentinela)

- Refinamento passo-a-passo top-down
  - Comece com uma representação em pseudocódigo do topo:
    Determine the class average for the quiz
  - Divida o topo em tarefas menores e as liste em ordem:
    Initialize variables
    Input, sum and count the quiz grades
    Calculate and print the class average

2.9 Formulando Algoritmos com Refinamento Passo-a-Passo Top-Down

- Muitos programas podem ser divididos em três fases:
  - Inicialização
    - Inicialização das variáveis do programa
  - Processamento
    - Valor dos dados de entrada são lidos e o programa ajusta as suas variáveis de acordo
  - Finalização
    - Calcula e imprime os resultados finais.
    - Ajuda a quebra de programas para um refinamento top-down.
- Refinando a fase de inicialização de
  Initialize variables
  para
  Initialize total to zero
  Initialize counter to zero

2.9 Formulando Algoritmos com Refinamento Passo-a-Passo Top-Down

- Refinando
  Input, sum and count the quiz grades
  para
  Input the first grade (possibly the sentinel)
  While the user has not as yet entered the sentinel
  Add this grade into the running total
  Add one to the grade counter
  Input the next grade (possibly the sentinel)
- Refinando
  Calculate and print the class average
  para
  If the counter is not equal to zero
  Set the average to the total divided by the counter
  Print the average
  Else
  Print "No grades were entered"

Índice


```
1  // Fig. 2.9: fig02_09.cpp
2  // Class average program with sentinel-controlled repetition.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::ios;
9
10 #include <iomanip>
11
12 using std::setprecision;
13 using std::setiosflags;
14
15 int main()
16 {
17    int total,        // sum of grades
18        gradeCounter, // number of grades entered
19        grade;        // one grade
20    double average;   // number with decimal point for average
21
22    // initialization phase
23    total = 0;
24    gradeCounter = 0;
25
26    // processing phase
27    cout << "Enter grade, -1 to end: ";
28    cin >> grade;
29
30    while ( grade != -1 ) {
```

Tipo de dado **double** usado para representar números decimais.

Índice


```
31       total = total + grade;
32       gradeCounter = gradeCounter + 1;
33       cout << "Enter grade, -1 to end: ";
34       cin >> grade;
35    }
36
37    // termination phase
38    if ( gradeCounter != 0 ) {
39       average = static_cast< double >( total ) / gradeCounter;
40       cout << "Class average is " << setprecision( 2 )
41            << setiosflags( ios::fixed | ios::showpoint )
42            << average << endl;
43    }
44
45
46
47
48
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

static_cast<double>() – trata o **total** como uma variável **double** temporariamente.

Necessário porque a divisão de dois inteiros trunca o resto.

gradeCounter é um **int**, mas é *promovido* para **double**.

ios::showpoint) –

ero de saída com um número fixo de ais.

cision(2) – imprime somente dois dígitos após o ponto decimal.

| – separa múltiplas

Programas que usam estas opções devem incluir **<iomanip>**

2.10 Estruturas de Controle Aninhadas

- Problema:
  Um colégio tem uma lista de resultados de um teste (1 = aprovado, 2 = reprovado) para 10 alunos. Escreva um programa que analise os resultados. Se mais de 8 alunos foram aprovados, imprima "Aumente a Mensalidade".
- Podemos ver que
  - O programa deve processar 10 resultados de testes. Um laço controlado por contador será usado.
  - Dois contadores podem ser usados—um para contar o número de alunos aprovados no exame e um para contar o número de alunos reprovados no exame.
  - Cada resultado do teste é um número—ou 1 ou 2. Se o número não for 1, iremos assumir que é um 2.
- Comando no nível topo:
  Analyze exam results and decide if tuition should be raised

2.10 Estruturas de Controle Aninhadas

- Primeiro Refinamento:
  - *Initialize variables*
  - *Input the ten quiz grades and count passes and failures*
  - *Print a summary of the exam results and decide if tuition should be raised*
- Refinando
  - *Initialize variables*
  - para
  - *Initialize passes to zero*
  - *Initialize failures to zero*
  - *Initialize student counter to one*

2.10 Estruturas de Controle Aninhadas

- Refinando
  - *Input the ten quiz grades and count passes and failures*
  - para
  - *While student counter is less than or equal to ten*
    - *Input the next exam result*
    - *If the student passed*
      - *Add one to passes*
    - *Else*
      - *Add one to failures*
    - *Add one to student counter*
- Refinando
  - *Print a summary of the exam results and decide if tuition should be raised*
  - para
  - *Print the number of passes*
  - *Print the number of failures*
  - *If more than eight students passed*
    - *Print "Raise tuition"*

Índice

1. Inicializar as variáveis

2. Entrada dos dados e contar aprovações / reprovações

```
1  // Fig. 2.11: fig02_11.cpp
2  // Analysis of examination results
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    // initialize variables in declarations
12    int passes = 0,           // number of passes
13        failures = 0,         // number of failures
14        studentCounter = 1,   // student counter
15        result;               // one exam result
16
17    // process 10 students; counter-controlled loop
18    while ( studentCounter <= 10 ) {
19       cout << "Enter result (1=pass,2=fail): ";
20       cin >> result;
21
22       if ( result == 1 )       // if/else nested in while
23          passes = passes + 1;
```


Índice

3. Imprimir os resultados

```
24       else
25          failures = failures + 1;
26
27       studentCounter = studentCounter + 1;
28    }
29
30    // termination phase
31    cout << "Passed " << passes << endl;
32    cout << "Failed " << failures << endl;
33
34    if ( passes > 8 )
35       cout << "Raise tuition " << endl;
36
37    return 0;   // successful termination
38 }
```

Saída do Programa

```
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 2
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Passed 9
Failed 1
Raise tuition
```


2.11 Operadores de Atribuição

- Abreviação de expressões de atribuição
  - `c = c + 3;` pode ser abreviados para `c += 3;` usando o operador de atribuição de adição
- Comandos na forma

```
variable = variable operator expression;
```

  podem ser reescritos como

```
variable operator= expression;
```

- Exemplos de outros operadores de atribuição:

```
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```


2.12 Operadores de Incremento e Decremento

- Operador de incremento (`++`) – pode ser usado ao invés de `c += 1`
- Operador de decremento (`--`) – pode ser usado ao invés de `c -= 1`
  - Pré-incremento
    - Quando o operador é usado antes da variável (`++c` ou `--c`)
    - Variável é alterada, e só então a expressão quer a contém é avaliada.
  - Pós-incremento
    - Quando o operador é usado antes da variável (`c++` ou `c--`)
    - A expressão que contém a variável é executada, e só então a variável é alterada.
- Se `c` é `5`, então
  - `cout << ++c;` imprime **6** (`c` é alterada antes que `cout` seja executado)
  - `cout << c++;` imprime **5** (`cout` é executado antes do incremento. `c` agora vale **6**)

2.12 Operadores de Incremento e Decremento

- Quando a variável não está em uma expressão
  - Pré-incremento e pós-incremento têm o mesmo efeito.

```
++c;
cout << c;
```

e

```
c++;
cout << c;
```

têm o mesmo efeito.

2.13 O Essencial da Repetição Controlada por Contador

- Repetição controlada por contador necessita:
  - O nome da variável de controle (ou contador do laço).
  - O valor inicial da variável de controle.
  - A condição que testa o valor final da variável de controle (ou seja, se o laço deve continuar).
  - O incremento (ou decremento) que modifica a variável de controle a cada passagem pelo laço.
- Exemplo:

```
int counter =1;           //initialization
while (counter <= 10){    //repetition
                          // condition
   cout << counter << endl;
   ++counter;             //increment
 }
```


2.13 O Essencial da Repetição Controlada por Contador

- A declaração

```
int counter = 1;
```

  - Nomeia **counter**
  - Declara **counter** como um inteiro
  - Reserva espaço em memória para **counter**
  - Inicializa **counter** com o valor **1**

2.14 A Estrutura de Repetição for

- O formato geral dos laços **for** é

```
for ( inicialização; TesteDeContinuidade;
  incremento )
   comando
```

- Exemplo:

```
for( int counter = 1; counter <= 10; counter++ )
   cout << counter << endl;
```

  - Imprime os inteiros de 1 a 10

Sem ponto-e-vírgula após o último termo

2.14 A Estrutura de Repetição for

- Laços **for** normalmente podem ser escritos como laços **while**:

```
inicialização;
while ( TesteDeContinuidade ){
   comando
   incremento;
}
```

- Inicialização e incremento como listas separadas por vírgula

```
for (int i = 0, j = 0;  j + i <= 10; j++, i++)
   cout << j + i << endl;
```


2.15 Exemplos Usando a Estrutura for

- Programa para somar os número pares de 2 a 100

```
1  // Fig. 2.20: fig02_20.cpp
2  // Summation with for
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int sum = 0;
11
12    for ( int number = 2; number <= 100; number += 2 )
13       sum += number;
14
15    cout << "Sum is " << sum << endl;
16
17    return 0;
18 }
```

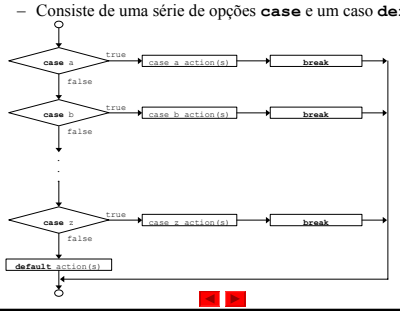
```
Sum is 2550
```

2.15 Exemplos Usando a Estrutura for

- Comando *for* não precisa apresentar expressões de:
  - **inicialização**
  - **TesteDeContinuidade**
  - **incremento**
- Exercício: refazer o programa anterior 2 vezes
  - Uma sem inicialização na estrutura do comando *for*
  - Uma sem incremento na estrutura do comando *for*
- O que acontece se o teste na estrutura do comando *for* for omitido

2.16 A Estrutura de Seleção Múltipla switch

- **switch**
  - Útil quando a variável ou expressão devem ser testadas para múltiplos valores
  - Consiste de uma série de opções **case** e um caso **default** opcional



Índice

1. Inicializar as variáveis
2. Entrada dos daos

2.1 Use o laço switch para atualizar os contadores

```
1  // Fig. 2.22: fig02_22.cpp
2  // Counting letter grades
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    int grade,       // one grade
12        aCount = 0,  // number of A's
13        bCount = 0,  // number of B's
14        cCount = 0,  // number of C's
15        dCount = 0,  // number of D's
16        fCount = 0;  // number of F's
17
18    cout << "Enter the letter grades." << endl
19         << "Enter the EOF character to end input." << endl;
20
21    while ( ( grade = cin.get() ) != EOF ) {
22
23       switch ( grade ) {
24
25          case 'A':  // grade was uppercase A
26          case 'a':  // or lowercase a
27             ++aCount;
28             break;  // necessary to exit switch
29
30          case 'B':  // grade was uppercase B
31          case 'b':  // or lowercase b
32             ++bCount;
33             break;
34
```

Note como o comando **case** é usado

Índice

2.1 Use o laço switch para atualizar os contadores

3. Imprima os resultados

```
35          case 'C':  // grade was uppercase C
36          case 'c':  // or lowercase c
37             ++cCount;
38             break;
39
40          case 'D':  // grade was uppercase D
41          case 'd':  // or lowercase d
42             ++dCount;
43             break;
44
45          case 'F':  // grade was uppercase F
46          case 'f':  // or lowercase f
47             ++fCount;
48             break;
49
50          case '\n': // ignore newlines,
51          case '\t': // tabs,
52          case ' ':  // and spaces in
53             break;
54
55          default:   // catch all other characters
56             cout << "Incorrect letter grade entered."
57                  << " Enter a new grade." << endl;
58             break;  // optional
59       }
60    }
61
62    cout << "\n\nTotals for each letter grade are:"
63         << "\nA: " << aCount
64         << "\nB: " << bCount
65         << "\nC: " << cCount
66         << "\nD: " << dCount
67         << "\nF: " << fCount << endl;
68
69    return 0;
70 }
```

break faz o **switch** terminar e o programa continua do primeiro comando após a estrutura **switch**.

Note o comando **default**.

Índice

Saída do Programa

```
Enter the letter grades.
Enter the EOF character to end input.
a
B
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```


2.17 A Estrutura de Repetição do/while

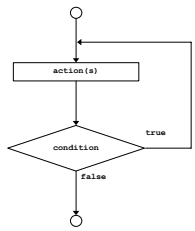
- A estrutura de repetição **do/while** é similar à estrutura **while**,
  - A condição de repetição é testada após o corpo do laço ser executado
- Formato:

```
do {
   comando
} while ( condição );
```

- Exemplo (com `counter` = 1):

```
do {
  cout << counter << " ";
} while (++counter <= 10);
```

  - Imprime inteiros de **1** a **10**



- Todas as ações são executadas pelo menos uma vez

2.18 Os comandos break e continue

- **`Break`**
  - Causa a saída imediata de estruturas **`while`**, **`for`**, **`do/while`** ou **`switch`**
  - A execução do programa continua do primeiro comando após a estrutura
  - Usos comuns do comando **`break`**:
    - Saída prematura de um laço
    - Ignorar o restante de uma estrutura **`switch`**

2.18 Os comandos break e continue

- **`Continue`**
  - Ignora os comandos restantes do corpo de uma estrutura **`while`**, **`for`** ou **`do/while`** e continua a próxima iteração do laço
  - Em um **`while`** e **`do/while`**, o teste de continuação do laço é avaliado imediatamente após a execução do comando **`continue`**
  - Na estrutura **`for`**, a expressão de incremento é executada, e só então o teste de continuação do laço é avaliada

2.19 Operadores Lógicos

- **`&&`** (**`AND`** lógico)
  - Retorna **`true`** se ambas as condições forem **`true`**
- **`||`** (**`OR`** lógico)
  - Retorna **`true`** se pelo menos uma das condições for **`true`**
- **`!`** (**`NOT`** lógico, negação lógica)
  - Reverte a condição de verdade/falsidade de sua condição
  - Retorna **`true`** quando a condição é **`false`**
  - É um operador unário, tomando apenas uma condição
- Operadores lógicos usados como condições em laços

Expressão	Resultado
`true && false`	`false`
`true || false`	`true`
`!false`	`true`

2.20 Confundindo Operadores de Igualdade (==) e Atribuição (=)

- Estes erros são danosos por não gerarem erros de sintaxe comuns.
  - Lembre-se de que qualque expressão que produza um valor pode ser usada em uma estrutura de controle. Valores diferentes de zero são **`true`**, e valores zero são **`false`**
- Exemplo:

```
if ( payCode == 4 )
   cout << "You get a bonus!" << endl;
```

  - Checa o código de pagamento, e se for **`4`** então um bônus é premiado
- Se **`==`** fosse substituído por **`=`**

```
if ( payCode = 4 )
   cout << "You get a bonus!" << endl;
```

  - Faz **`payCode`** receber o valor **`4`**
  - **`4`** é diferente de zero, então a expressão é **`true`** e um bônus é premiado, independente do valor de **`payCode`**.

2.20 Confundindo Operadores de Igualdade (==) e Atribuição (=)

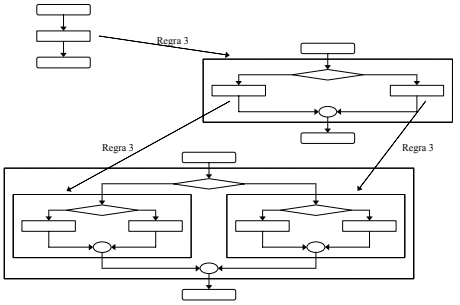
- Lvalues
  - Expressões quer podem aparecer no lado esquerdo (left) de uma equação
  - Seus valores podem ser alterados
  - Nomes de variáveis são exemplos comuns
    - como em **`x = 4;`**
- Rvalues
  - Expressões que só podem aparecer no lado direito (right) de uma equação
  - Constantes, como números
    - ou seja, você não pode escrever **`4 = x;`**
- Lvalues podem ser usados como Rvalues, mas não o contrário

2.21 Sumário de Programação Estruturada

- Programação Estruturada
  - Programas são fáceis de entender, testar, depurar e modificar.
- Regras para programação estruturada
  - Somente estruturas de controle com única-entrada/única-saída são usadas
  - Regras:
    1) Começe com o "fluxograma mais simples".
    2) Qualquer retângulo (ação) pode ser substituído por dois retângulos (ações) em seqüência.
    3) Qualquer retângulo (ação) pode ser substituído por qualquer estrutura de controle (seqüência, if, if/else, switch, while, do/while ou for).
    4) Regras 2 e 3 podem ser aplicadas em qualquer ordem múltiplas vezes.

2.21 Sumário de Programação Estruturada

Representação da Regra 3 (substituindo qualquer retângulo por uma estrutura de controle)



2.21 Sumário de Programação Estruturada

- Todos os programas podem ser quebrados em
  - Seqüência
  - Seleção
    - **`if`**, **`if/else`**, ou **`switch`**
    - Qualquer seleção pode ser reescrita como um comando **`if`**
  - Repetição
    - **`while`**, **`do/while`** ou **`for`**
    - Qualquer estrutura de repetição pode ser reescrita como um comando **`while`**