CHAPTER 4: CONDITIONAL STRUCTURES

Flow of Execution

- Every algorithm has a logic flow
- There is a start, steps that happen in chronological order, and an end
- There is a graphical way to describe program flow
- Understanding control flow is essential to creating and testing an implementation of an algorithm

Figure 4.1: This chart has a one directional flow (each step is performed once before the next, but some algorithms can have multiple possible execution flows)

Conditional Control

- A condition is an expression defined using relational and Boolean operators
- A condition has a Boolean value, either True or False

irb(main):001:0> 5 == 5 => true
irb(main):002:0> 5 == 6 => false
irb(main):003:0> 5 <= 5 => true
irb(main):004:0> 5 != 5 => false
Relational Operators (Table 4.1)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Is equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>

Conditional Control

- The "!" operator is the negation of a condition or Boolean value
- "!" can work on any true or false statement or conditional
- Usually referred to as "not"
- Boolean operators operate on Boolean values, creating expressions that evaluate to True or False
- Operators include: and, or, not
- The results of the operators are described by truth tables

Example: Boolean Expressions

```
irb(main) :001> !false
=> true
irb(main) :002> !(true or false)
=> false
irb(main) :003> first = true
=> true
irb(main) :004> second = false
=> false
irb(main) :005> (first and second or !(first and second))
=> true
```
Conditional Flow: If Statements

- Ruby uses an if statement for basic conditional control flow (Example 4.3)

```ruby
1 if (condition)
  # section 1
end
```

- Input value of 11 (Example 4.4):

```ruby
# if a number is even, print out "Even"
puts "Enter a number"
number = gets.to_i
if (number % 2 == 0) # evaluates to false
  puts "Even" # does not execute
end
```

Section 1 is executed when the condition evaluates to true or is skipped when the condition evaluates to false.

Conditional Flow: If-Then-Else Statements

- Provides a second flow option
- If the original condition is not met, then the second flow option is taken (Example 4.5)

```ruby
1 if (condition)
  # section 1 executes if true
2 else
  # section 2 executes if false
3 end
```

Example of Program that Determines Prices of Movie Tickets (Example 4.6)

```ruby
puts "Enter the customer's age:"
age = gets.to_i
if (age < 12)
  cost = 9
else
  cost = 18
end
puts "Ticket cost:" + cost.to_s
```

If-Else Statement Logic Flow

- To test the program, input one value for each logic flow option
- Test the edge or boundary conditions (most errors occur here)
```
Movie Ticket Example: Input Value of 8
(Example 4.6)

1 puts "Enter the customer's age: 
2 # Get an integer age value from the user
3 age = gets.to_i
4 5 # Determine the cost based on age
6 if (age < 12) # evaluates to true
7 cost = 9 # so the If portion Executes
8 else
9 cost = 18 # This portion DOES NOT
10 end
11 12 # Print out the final cost
13 puts "Ticket cost: " + cost.to_s
```

(c) 2012 Ophir Frieder et al

```
Movie Ticket Example: Input Value of 25
(Example 4.6)

1 puts "Enter the customer's age: 
2 # Get an integer age value from the user
3 age = gets.to_i
4 5 # Determine the cost based on age
6 if (age < 12) # Evaluates to false
7 cost = 9 # This DOES NOT execute
8 else
9 cost = 18 # Executes
10 end
11 12 # Print out the final cost
13 puts "Ticket cost: " + cost.to_s
```

(c) 2012 Ophir Frieder et al

```
Movie Ticket Example: Input Value of 12
(Figure 4.9)

1 puts "Enter the customer's age: 
2 # Get an integer age value from the user
3 age = gets.to_i
4 5 # Determine the cost based on age
6 if (age < 12) # Evaluates to false
7 cost = 9
8 else
9 cost = 18 # Executes
10 end
11 12 # Print out the final cost
13 puts "Ticket cost: " + cost.to_s
```

(c) 2012 Ophir Frieder et al

```
Elsif Statements

- Conditional flow can have more than two flow options
- There are various ways to implement a multi-flow control
- One of them is using an elsi statement
```

(c) 2012 Ophir Frieder et al
Program that Discounts Tickets for Children & Senior Citizens (Example 4.9)

```ruby
1 puts "Enter the customer's age: 
2 # Get an integer age value from the user
3 age = gets.to_i
4 5 # Determine the cost based on age
6 if (age <= 12) 7 cost = 9 8 elsif (age >= 65) 9 cost = 12 10 else 11 cost = 18 12 end 13 14 # Print out the final cost 15 puts "Ticket cost: " + cost.to_s
```

# Print an integer age value from the user
# Determine the cost based on age
# Print out the final cost

SYNTACTIC SUGAR IN ACTION: Alternative syntax designed for ease of programming and readability

```
1 puts "Enter the customer's age:
2 # Get an integer age value from the user
3 age = gets.to_i
4 5 # Determine the cost based on age
6 if (age <= 12) then cost = 9 end
7 # Print out the final cost
8 puts "Ticket cost: " + cost.to_s
```

Alternative Syntax

Review: Original Movie Ticket Program (Example 4.6)

```ruby
1 puts "Enter the customer's age:
2 # Get an integer age value from the user
3 age = gets.to_i
4 5 # Determine the cost based on age
6 if (age <= 12) then cost = 9 end
7 # Print out the final cost
8 puts "Ticket cost: " + cost.to_s
```

SYNTACTIC SUGAR IN ACTION: Alternative syntax designed for ease of programming and readability

```
1 puts "Enter the customer's age:
2 # Get an integer age value from the user
3 age = gets.to_i
4 5 # Determine the cost based on age
6 if (age <= 12) then cost = 9 end
7 # Print out the final cost
8 puts "Ticket cost: " + cost.to_s
```

Alternative Program

Elif Statement Logic Flow (Figure 4.5)

Only the first condition that evaluates to true gets executed.
Case Statements

- The case statement handles multiple options
- Alternative to if-elsif statements
- Useful for a large number of options
- Case statements evaluate in order
- Only the first when clause that evaluates to true gets executed
- If none evaluates to true, then the else clause is executed

Figure 4.11:

```
case
  when (expression1)
    # section 1
  when (expression2)
    # section 2
  else
    # section 3
end
```

Movie Ticket Program: Rewritten Using a Case Statement (Example 4.12)

```
puts "Enter the customer's age: "
age = gets.to_i
# Determine the cost based on age
case
  when (age <= 12)
    cost = 9
  when (age >= 65)
    cost = 12
  else
    cost = 18
end
puts "Ticket cost: " + cost.to_s
```

Debugging:
Incorrect Movie Ticket Program (Example 4.13)

**Example 1:** The cost will always be 9

```
puts "Enter the customer's age: "
age = gets.to_i
when (age = 12) then
  cost = 9
when (age >= 65) then
  cost = 12
else
  cost = 18
end
puts "Ticket cost: " + cost.to_s
```

Debugging:
Incorrect Movie Ticket Program (Example 4.14)

**Example 2:**

```
puts "Enter the customer's age: "
age = gets.to_i
when (age = 12) then
  cost = 9
when (age >= 65) then
  cost = 12
else
  cost = 18
end
puts "Ticket cost: " + cost.to_s
```
Debugging

- Uses `puts` statements to help identify errors.
- Show variable values where they are not changing.

Example 4.14 cont'd:
17 # DEBUG
18 puts age # Shows that age always equals 12
19 20 # Print out the final cost
21 puts "Ticket cost: " + cost.to_s

(c) 2012 Ophir Frieder et al

Debugging: Alternatives

- Programs can also be debugged using constants.
- In each section, there is an `if` statement with a debugging constant as the flag.
- The flag determines whether a put statement is executed.
- When a section is judged to be correct, the constant is set to false.
- There is no need to check variables.
- The debug output should be fully descriptive.

Example 4.15:
1 # Flag for debugging (change the false when finished debugging)
2 DEBUG_MODULE_1 = true # Initialize and define a flag constant as true.
3 4 puts "Enter the customer's age: "
5 # Get an integer age value from the user
6 age = gets.to_i
7 8 # Determine the cost based on age
9 if DEBUG_MODULE_1 # Changed to false if this section is correct
10 puts age # Prints age if the section is still not debugged
11 end
12 case
13 when (age = 12) # Changed to false if this section is correct
14 puts "debug – age" + age.to_s # NOT puts "age"
15 when (age >= 65) then
16 cost = 9
17 else
18 cost = 18
19 end
20 end
21 if DEBUG_MODULE_1 # Changed to false if this section is correct
22 puts age # prints age if the section is still not debugged incorrect
23 end
24 25 # Print out the final cost
26 puts "Ticket cost: " + cost.to_s

(c) 2012 Ophir Frieder et al
Summary

- Every program follows a control flow, which is determined by the logic flow of its algorithms.
- Logic and control flow can often be one directional or conditional.
- The relational operators are the key operators to creating conditional flows.
- Another way to create conditional flow is by employing if, elsif, and case statements.

(c) 2012 Ophir Frieder et al