Computational Expression

Conditionals

Janyl Jumadinova

10 October, 2018
Computational Thinking: a problem solving process

- Decomposition
- Pattern Recognition
- Abstraction
- Algorithm Design
Algorithms

- **Algorithm** is a procedure for solving a problem in terms of the *actions* to be executed and the *order* in which those actions are to be executed.
Algorithms

- **Algorithm** is a procedure for solving a problem in terms of the *actions* to be executed and the *order* in which those actions are to be executed.
Control Structures

- We may need to be able to **make decisions** (selection) and **repeat actions** (looping) in our programs to allow for more complex programs.
- Selection and looping are common to all programming languages. The way they implement these concepts, however, may differ from language to language.
Control Structures

Three Groups of Control Structures

1. **Sequential Structure**
   - It is just built into the language itself.
Control Structures

Three Groups of Control Structures

1. Sequential Structure

2. Selection Structures
   - **if** : single selection
   - **if/else** : double or multiple selection
   - **switch** : multiple selection
Control Structures

Three Groups of Control Structures

1. Sequential Structure
2. Selection Structures
3. **Repetition Structure**
   - `while`
   - `do/while`
   - `for`

[Diagram: Loop (Iteration)]
Java programs are built from only these seven control structures:

- three *selection* (if, if/else, switch)
- three *repetition* (while, do/while, for)

You implement computer algorithms by stringing sequences of these seven control structures together.
if/else

- **if** only has a “do it or don’t do it” mentality – if the assertion is true, you do the associated action, if it’s false, you skip it.
if/else

- **if** only has a “do it or don’t do it” mentality – if the assertion is true, you do the associated action, if it’s false, you skip it.

- The **if/else** structure gives more flexibility by providing something to do if the assertion is false – the “else” portion of the structure.

- **Nested if/else** structure strings together multiple if/else statements to handle a range of values.
Which of these code segments will determine a letter grade correctly based on a variable ‘grade’?

```java
if (grade < 60)
    System.out.println("F");
else if (grade >= 60)
    System.out.println("D");
else if (grade >= 70)
    System.out.println("C");
else if (grade >= 80)
    System.out.println("D");
else
    System.out.println("A");
```

```java
if (grade >= 90)
    System.out.println("A");
else if (grade >= 80)
    System.out.println("B");
else if (grade >= 70)
    System.out.println("C");
else if (grade >= 60)
    System.out.println("D");
else
    System.out.println("F");
```
What if you wanted to do more than one thing in an `if` or an `if/else` action?

Need to use braces (`{` and `}`) to form a compound statement.
if and if/else tips to remember:

- They can be used to test ranges of values.
- In a nested if/else structure, an else always attempts to match up with the closest and most immediately unmatched preceding if statement.
- Always use compound statements with if/else structures to prevent problems down the road.
Logical Operators

- Using logical operators, we have a way to string multiple simple conditions together to help avoid/simplify nesting statements.
- These logical operators are based on the concept of Boolean logic or Boolean algebra.
Using logical operators, we have a way to string multiple simple conditions together to help avoid/simplify nesting statements.

These logical operators are based on the concept of Boolean logic or Boolean algebra.

These are the three logical operators in Java:

1. `&&` (logical AND)
2. `||` (logical OR)
3. `!` (logical NOT, or negation)
## Logical `and` Truth Table

<table>
<thead>
<tr>
<th><code>expr1</code></th>
<th><code>expr2</code></th>
<th><code>expr1 &amp;&amp; expr2</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Logical **or** Truth Table

<p>| expr1 | expr2 | expr1 || expr2 |
|-------|-------|--------|--------|
| false | false | false  |        |
| false | true  | true   |        |
| true  | false | true   |        |
| true  | true  | true   |        |</p>
<table>
<thead>
<tr>
<th>expr1</th>
<th>!expr1</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>