

Conditionals

Lecture Outline & Notes

Lecture Outline

1. Introduction

- We need a way to make decisions based on various conditions or to handle different cases in different ways
- Default control flow is sequential: statements are executed in a linear manner, unconditionally; *order matters*
- Default control flow can be interrupted with *conditional* (aka “selection”) control structures
- Conditionals may execute single statements or segments (blocks) of code
- Major types of conditional statements:
 - i. If-statement
 - ii. If-then-else
 - iii. If-else-if
 - iv. Switch statement

2. Logical Statements & Operators

A logical statement or condition is a statement (variable or expression) that is either true or false

1. Numeric comparison operators

- i. <
- ii. >
- iii. <=
- iv. >=
- v. ==
- vi. !=

2. Statements

- i. Binary operators have two operands
- ii. Operands can be:
 1. Constants
 2. Variables
 3. Expressions

3. Compound statements

- i. Unary negation operator
 1. Syntax: !
 2. Expression or variable
 3. C: no Boolean variables, can apply ! to any numeric operand

- 4. Java: may only apply ! to a Boolean operand (variable or expression)
 - ii. Logical conjunction (And)
 - 1. Syntax: &&
 - 2. Expression or variable
 - 3. True only if *both* of its operands are true
 - iii. Logical disjunction (Or)
 - 1. Syntax: ||
 - 2. Expression or variable
 - 3. True only if at least one of its operands is true
 - iv. DeMorgan's law
 - 1. $!(a \ || \ b) = (!a \ \&\& \ !b)$
 - 2. $!(a \ \&\& \ b) = (!a \ || \ !b)$
 - v. Operator precedence: !, {+, -} (unary), {*, /, %}, {+, -} (binary), {<, >, <=, >=}, {==, !=}, {&&}, {||}
 - vi. Short Circuiting
 - 1. Left-to-right evaluation
 - 2. For &&, if first operand is false, second does not get evaluated
 - 3. For ||, if first operand is true, second does not get evaluated
 - 4. Non numeric comparisons
 - i. Character comparison
 - 1. ASCII Text Table
 - ii. String comparison (more later)
 - iii. User defined types (more later): the Comparator pattern
3. If Statement
- 1. Syntax:


```
if(<expression>)
```
 - 2. Behavior: code block immediately following the if statement is executed if and only if the expression evaluates to true
 - 3. Best practice: always use curly brackets even if it only applies to one statement (readability, consistency, easy to add additional code)
4. If-Else Statement
- 1. Syntax:


```
if(<expression>) { ..
} else { ..
}
```
 - 2. Behavior: exactly one of the code blocks is executed depending on whether the expression is true or false
 - 3. Shorthand syntax:


```
<condition> ? <>true result> : <>false result>
int min = (a < b) ? a : b;
```
5. If-Else-If statement

1. Syntax:


```
if(<expression>) { ..
  } else if(<expression2>) { ..
  } else if(<expression3>) { ..
  } else { ..
  }
```
 2. Any number of else-if statements are allowed; last else statement is optional
 3. Best for range checking
6. Switch Statement
1. Syntax:


```
switch(variable) {
  case A:
    codeblock A;
    break;
  case B: ...
  default:
    defaultcodeblock;
    break;
}
```
 2. Switch variable may only be an integer or character (switch for `String` types supported in Java 1.7+)
 3. Behavior: code blocks start executing on the first matched case, end at the next `break` statement.
 4. Omitting the `break` statements can lead to *fall through*: code blocks in other cases will be executed until a `break` statement is encountered
 5. Best to include a default block
7. Nested Statements
1. Style: always use the same level of indentation for code blocks at the same level
8. Common errors
1. `if(a = 5)`
 - i. C: always evaluates to true, `if(a = 0)` always evaluates to false
 - ii. Java: not possible (compile error) since it is not possible to implicitly convert from an integer to a Boolean (since Java has a built-in `boolean` type)
 2. `if(0 <= x <= 4)`
 - i. C: left-to-right evaluation: `0 <= x` is 0 or 1 which is always less than 4
 - ii. Java: syntax error, same reason
 3. Both: `if(a == 5);` is syntactically correct, but will not produce expected behavior. Semicolons used for statements, not control structures