ITM 312, Fall 2013

Chapter 4 Lecture Notes

4.1. Relational Operators

a. Use operators to make comparisons
   i. > greater than
   ii. < less than
   iii. >= greater than or equal to
   iv. <= less than or equal to
   v. == equal to
   vi. != not equal to

b. Boolean expressions are `true` or `false`
   i. `12 > 5` → `true`
   ii. If `x=10`, then `x==10` is `true`

c. Expressions can be assigned to a variable
   i. `true` typecasted to an int is `1`, `false` typecasted to an int is `0`
   ii. = denotes assignment, == denotes comparison

4.2. The `if` Statement

a. Used to make decisions on what code to execute
   i. if the condition is false, skip the block of code
   ii. Multiple statements can be contained within the `{}` of the if statement

b. Syntax:

```
if (expression)
    statement;
```

   i. Do not place a semicolon after (expression)
   ii. Place `statement;` on a separate line after expression, indented

c. To evaluate:
   i. If `expression` returns `true`, execute the statement
      1. Be careful testing `floats` and `doubles` for equality
      2. 0 is false, any other value is true
   ii. If `expression` returns `false`, skip the statement

4.3. Expanding the if statement

a. Curly braces { } create a block of code

b. Use these curly braces to execute more than one statement as part of an if statement

c. Syntax:

```
if (score > 90)
{
    grade = 'A';
    cout << "Good Job!\n";
}
```

4.4. The if/else statement

a. Provides two possible paths of execution

b. Performs the code inside the if statement if the expression is true, otherwise performs the code inside the else block

c. Syntax:

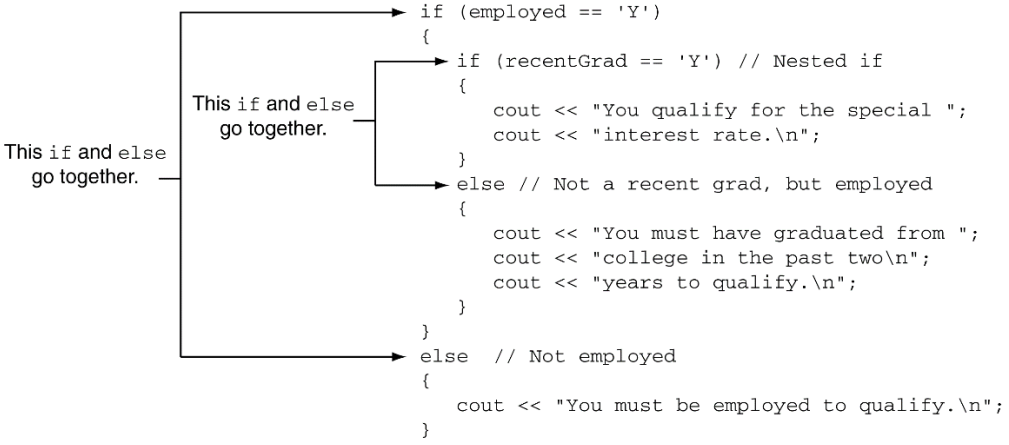
```
if (expression)
    statement1; // or block; execute this if expression is true
else
    statement2; // or block; execute this if expression is false
```

4.5. Nested `if` Statements

a. An `if` statement that is nested inside another `if` statement

b. Can be used to test more than one condition

c. Syntax:



4.6. The `if/else if` Statement

a. Tests a series of conditions until one is found to be true

b. Often simpler than using nested if/else statements

c. Can be used to model thought processes

   i. If it is winter, wear a coat

   ii. Else, if it is fall, wear a jacket

   iii. Else, wear sunglasses

d. Syntax:

```
if (expression1)
    statement1;  // or block
else if (expression2)
    statement2;  // or block
                 // other else ifs
else if (expressionn)
    statementn;  // or block
```

4.7. Flags

a. Variables that signal a condition

b. Usually implemented as a `bool` variable

c. Can also be an integer

   i. Like if statements, 0 is considered false and any other nonzero value is considered true

d. Must be assigned an initial value before it can be used

4.8. Logical Operators

a. Used to create relational expressions from other relational expressions

b. Operators:

   i. `&&` / AND – new relational expression is true if both expressions are true

   ii. `||` / OR – new relational expression is true if either expression is true

   iii. `!` / NOT – reverses the value of an expression – true becomes false, and false becomes true

c. Syntax (`int x = 12, y = 5, z = -4`):

   i. `(x > y) && (y > z)` → true

   ii. `(x <= z) || (y != z)` → true

   iii. `!(x >= z)` → false

d. ! has highest precedence, followed by &&, then || (order of operations)

e. If the value of an expression can be determined by evaluating just the sub-expression on left side of a logical operator, then the sub-expression on the right side will not be evaluated (*short circuit evaluation*)
    i. E.g. `int x, y; x = 0;`
    `(x && y)` → false
    Will not give an error because it did not check the value of `y`, `x` is already false

4.9. Checking Numeric Ranges with Logical Operators

a. Use to test if value falls inside a range:

```
if (grade >= 0 && grade <= 100)
    cout << "Valid grade";
```

b. Or outside a range:

```
if (grade <= 0 || grade >= 100)
    cout << "Invalid grade";
```

c. Cannot use mathematical notation:

```
if (0 <= grade <= 100) //doesn't work!
```

4.10. Menus

a. A menu-driven program is a program whose execution is controlled by the user selecting from a list of actions
    i. Menu – list of choices on a screen
b. Menu-driven programs display a list of numbered or lettered choices for actions
c. Prompt user to make a selection
d. Test the user selection in an expression
    i. If a match, then execute the code for that action
    ii. If not, then go onto the next expression

4.11. Validating User Input

a. Input validation – inspecting input data to determine whether it is acceptable
b. Bad output will be produced from bad input
c. Validation tests:
    i. Within range?
    ii. Is it reasonable?
    iii. Is the menu choice valid?

iv. Check for divide by zero error

4.12. Comparing Characters and Strings

a. Characters compared using their corresponding ASCII values
   i. `'A' < 'B'` is true because the ASCII value of A (65) is less than the ASCII value of B (66)
   ii. Lowercase letters have higher ASCII codes than uppercase letters
   `'a' > 'Z'` is true
b. Strings are also compared using their ASCII values
   i. The characters in each string must match before they are equal (*case-sensitive*)
   ii. ``Mary'' < ``Mary Jane'' is true
   ``Mary'' <= ``Mark'' is false

4.13. The Conditional Operator

a. Shorthand version of if/else statement
b. Syntax: `expr1 ? expr2 : expr3;`
   If expr1 is true, execute expr2, else execute expr3

4.14. The `switch` Statement

a. Shorthand version of if/else if statements with the same condition
b. Syntax:

```
switch (expression) //integer
{
    case exp1: statement1;
               break;
    case exp2: statement2;
               break;
    case exp3: statement3;
               break;
    ...
    case expn: statementn;
               break;
    default: statementn+1;
}
```

c. Requirements
   i. `expression` must be an integer variable, or an expression that evaluates to an integer value

ii. *exp1* through *expn* must be constant integer expressions or literals, and must be unique within the switch statement (no duplicate expressions)
iii. `default` is optional but recommended

d. How it works
i. Expression is evaluated
ii. The value of expression is compared against *exp1* through *expn*
iii. If expression matches value *expn*, the program branches to the statement following *expn* and continues to the end of the switch
iv. If no matching value is found, the program branches to the statement after default