Lecture 6: Conditionals

AP Computer Science Principles

Type `boolean`

Type `boolean`

- **`boolean`**: A logical type whose values are `true` and `false`.

```
int age = 18;

boolean minor = age < 21; //true

boolean lovesCS = true;
```

The `if` statement

Executes a block of statements only if the test is true

```
if (test) {
    statement;
    ...
    statement;
}
```

- Example:

```
float gpa = 2.5;
if (gpa >= 2.0) {
    println("Application accepted.");
}
```

The `if/else` statement

Executes one block if a test is true, another if false

```
if (test) {
    statement(s);
} else {    // no test after else
    statement(s);
}
```

- Example:

```
float gpa = 1.9;
if (gpa >= 2.0) {
    println("Welcome to Mars University!");
} else {
    println("Application denied.");
}
```

Relational expressions

- Tests use *relational operators*:

Operator	Meaning	Example	Value
`==`	equals	`1 + 1 == 2`	`true`
`!=`	does not equal	`3.2 != 2.5`	`true`
`<`	less than	`10 < 5`	`false`
`>`	greater than	`10 > 5`	`true`
`<=`	less than or equal to	`126 <= 100`	`false`
`>=`	greater than or equal to	`5.0 >= 5.0`	`true`

Misuse of `if`

- What's wrong with the following code?

```
if (percent >= 90) {
    println("You got an A!");
}
if (percent >= 80) {
    println("You got a B!");
}
if (percent >= 70) {
    println("You got a C!");
}
if (percent >= 60) {
    println("You got a D!");
}
if (percent < 60) {
    println("You got an F!");
}
```

Nested `if/else`

Chooses between outcomes using many tests

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```

- Example:

```
if (x > 0) {
  println("Positive");
} else if (x < 0) {
  println("Negative");
} else {
  println("Zero");
}
```

Nested `if/else/if`

- If it ends with `else`, exactly one path must be taken.
- If it ends with `if`, the code might not execute any path.

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else if (test) {
    statement(s);
}
```

- Example:

```
if (place == 1) {
    println("Gold medal!");
} else if (place == 2) {
    println("Silver medal!");
} else if (place == 3) {
    println("Bronze medal.");
}
```

Nested `if` structures

- exactly 1 path *(mutually exclusive)*

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```

- 0 or 1 path *(mutually exclusive)*

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else if (test) {
    statement(s);
}
```

- 0, 1, or many paths *(independent tests; not exclusive)*

```
if (test) {
    statement(s);
}
if (test) {
    statement(s);
}
if (test) {
    statement(s);
}
```

Which nested `if/else`?

- **(1) if/if/if (2) nested if/else (3) nested if/else/if**
  - Whether a user is lower, middle, or upper-class based on income.
    - **(2)** nested `if / else if / else`
  - Whether you made the dean's list (GPA ≥ 3.8) or honor roll (3.5-3.8).
    - **(3)** nested `if / else if`
  - Whether a number is divisible by 2, 3, and/or 5.
    - **(1)** sequential `if / if / if`
  - Computing a grade of A, B, C, D, or F based on a percentage.
    - **(2)** nested `if / else if / else if / else if / else`

Logical operators

- Tests can be combined using *logical operators*:

Operator	Description	Example	Result
`&&`	and	`(2 == 3) && (-1 < 5)`	`false`
`||`	or	`(2 == 3) || (-1 < 5)`	`true`
`!`	not	`!(2 == 3)`	`true`

- "Truth tables" for each, used with logical values *p* and *q*:

p	q	p && q	p || q
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

p	!p
true	false
false	true

Using `boolean`

```
boolean goodAge    = age >= 18 && age < 29;
boolean goodHeight = height >= 70 && height < 76;
boolean rich       = salary >= 100000.0;

if ((goodAge && goodHeight) || rich) {
    println("Okay, let's go out!");
} else {
    println("It's not you, it's me...");
}
```

Evaluating logic expressions

- Relational operators have lower precedence than math.

```
5 * 7 >= 3 + 5 * (7 - 1)
5 * 7 >= 3 + 5 * 6
35    >= 3 + 30
35    >= 33
True
```

- Relational operators cannot be "chained" as in algebra.

```
2 <= x <= 10
```

  - Instead, combine multiple tests with `&&` or `||`

```
2 <= x && x <= 10
true   && false
false
```

Evaluating logic expressions

- ! is evaluated first, then AND is evaluated before OR.

```
int x = 2;
int y = 4;
int z = 5;

z > 2 || x > 3 && y < 3 ;
// true if evaluate && before ||
// false if evaluate || before &&
// the correct answer is true: &&
// must be evaluated before ||
```

Logical questions

- What is the result of each of the following expressions?

```
int x = 42;
int y = 17;
int z = 25;
```

- `y < x && y <= z`
- `x % 2 == y % 2 || x % 2 == z % 2`
- `x <= y + z && x >= y + z`
- `!(x < y && x < z)`
- `(x + y) % 2 == 0 || !((z - y) % 2 == 0)`

  - Answers: `true`, `false`, `true`, `true`, `false`

Scope

- **scope**: The part of a program where a variable exists.
  - From its declaration to the end of the `{ }` braces
    - A variable declared in a `if` block exists only in that block.
    - A variable declared in a method exists only in that method.

```
void setup() {
    int x = 3;
    if(x == 3){
        println(x); // x exists here
    }
    // x exists here
}
// x ceases to exist here
```

If Block

```
if(x <= 3)
{
      int y = 2;

}

y = 5; // error since y does not exist outside
     // the if block
```

Global vs Local Variables

Any variable declared BEFORE setup() is **global variable** and has a scope that spanned the entire program.

A variable declared inside of a method or a block is a **local variable** that exists only for that method or block.

```
int x = 5; //global variabe
void setup(){
        if(x > 10){
                int y = 2; //local variable
                println(y); //y exists here, as does x.
        }
        println(x); //ok
        println(y); //Error! Outside of y's scope.
}
```

Global vs Local Variables

```
int x = 5; // global variable

void setup(){
  println(x); //x exists here
  int y = 10; // local variable
  printMyNumber(x);
}

void draw() {
  println(x); // x exists here also.
  println(y); // Error! y does not exist here.

}

void printMyNumber(int x){//declaring a local variable x
  println(x); // x exists here
              //  however, this x is a local variable
              // not the same as the global x above
  }
```

Lab 1: Mouse Collision I

Write a program that that display a circle whose fill is red if the mouse is inside the circle and blue if it is outside.

Use the built in dist(x1,y1,x2,y2) to compute the distance between (x1,y1) and (x2,y2).

Lab 2: Mouse Collision II

Modify the previous program to display a circle and a rectangle that is initially filled with the color black. If the mouse is inside of the circle, the circle turns red. If it is inside of the rectangle, the rectangle turns blue. If the mouse is not in either shape, both remains black.

Lab 2: Mouse Collision II

Lab 3: Quadrants

Create a program that has a square window size, for example, size(600,600).

Draw a horizontal line and a vertical line that divides the window into four equal quadrants.

Modify the program so that the quadrant containing the current mouse position will light up red and stays the same background color otherwise.

Lab 4: Quadrants II

This program is similar to the previous program except that the quadrant will light up red if the mouse position has been there before.

Create a copy of the previous program instead of modifying it.

Homework

1) Read and reread these lecture notes.

2) Complete the problem set.

3) Complete the labs.

References

Part of this lecture is taken from the following book.

1) Stuart Reges and Marty Stepp. Building Java Programs: A Back to Basics Approach. Pearson Education. 2008.