Incorporating Decisions (if)

If statement

In many programs, we may not always want to perform the same set of steps. Instead, depending on some decision, we may want to carry out one set of steps or skip them. One simple example is calculating the cost of an item at the grocery store. Some items that are deemed to be essential foods (fresh fruit, for example) are not taxes, but other items that aren't necessary (for example chips), are taxed. Thus, we only want to add tax to the cost of an item if it's a taxed item.

In C++, the if statement is used for running different directions depending on a certain condition. The most simple form of an if statement is:

```
if (boolean expression) {
    stmt1;
    stmt2;
    …
    stmtn;
}
```

The parentheses after the if are required, and inside those parentheses, one can have any Boolean expression, of type bool or type int. (If an integer is in the expression, any non-zero number is treated as true and 0 is treated as false.) In short, if the Boolean expression evaluates to true, then all the statements inside the block (indicated by the curly braces) are executed. If the Boolean expression evaluates to false, then all the statements within the curly braces are skipped. The curly braces are only necessary if there is more than one statement to be placed "inside" the if, but it's good practice to always use them to avoid subtle errors.

Simple Boolean expressions are formed using the conditional operators:

```
>, >=, <, <=, ==, !=
```

To form a valid expression, place expressions to both the left and right of these operators. The meaning of the first four are straightforward. The fifth operator ***checks*** for equality between two expressions and the last is the "not equal to" operator. None of these, on their own, will modify the value of any variable, rather, they will check to see if two expressions satisfy the given operator and if so, return true, otherwise return false.

A single bool variable is also a valid Boolean expression.

Finally, we can create more complicated Boolean expressions with the following Boolean operators:

```
&& (and)
|| (or)
```

The and of two Boolean expressions is true ONLY if both expressions are true. The or of two Boolean expressions is true as long as at least ONE of the two expressions is true.

The if statement comes with optional branches (only the if is required). Here is the full possible syntax:

```
if (bool exp 1) {
    stmts1;
}
else if (bool exp 2) {
    stmts2;
}
…
else if (bool exp k) {
    stmtsk;
}
else {
    stmtslast;
}
```

The way the statement is interpreted is that each Boolean expression, in the order they appear, is evaluated until one evaluates to true. At that point in time, each of the statements inside of that branch are executed, and then command jumps to after the else closes. If none of the Boolean expressions is true, then the statements in the else clause are executed, and continues after it.

In the layout above, it's guaranteed that exactly one set of statements will be executed. If the else is omitted, then either one set of statements OR none of the statements will be executed. This layout of one if statement is DIFFERENT than the following layout:

```
if (bool exp 1) {
    stmts1;
}
if (bool exp 2) {
    stmts2;
}
```

This layout is two separate if statements, one after the other. In this layout, it's possible that BOTH sets of statements, stmts1 and stmts2 get executed, because even if bool exp 1 is true, after that if statement finishes, we simply continue to the next statement, which is the if statement with bool exp 2.

So a simple code snipped using the if might look like this:

```
double cost;
cin >> cost;
bool taxed;
int tax;
cin >> tax;
if (tax == 1)
    cost = cost*1.065
```

In this example, we read in the cost and whether or not the item is taxed (assume 1 means taxed and 0 means not taxed). If the item is taxed, then we reassign cost to be 6.5% higher. (As of this writing, this is the sales tax in Orange County, which is near UCF.)

There are quite a few potential pitfalls with if statements that can occur in C++ due to a few reasons:

1) Inadvertent semicolons
2) Forgetting the use of curly braces when they are necessary
3) Accidental use of an arithmetic expression instead of a Boolean expression (since C++ allows both in the if statement decision)
4) Not using parentheses when they are necessary due to order of operations rules.

Rather than go through a full listing of the rules and pitfalls (this could take 20 pages, no joke), our philosophy here will be to get started quickly and expect students to discover these rules as unintended behavior occurs.

If Statement and Integer Division Example: FYI

Here's a link to the problem:

https://open.kattis.com/problems/fyi

The gist of the problem is that we must read in an integer guaranteed to be 7 digits long and extract the first three digits to see if they equal to 555 or not. As previously discussed, dividing an integer by 10 is the same as chopping off its last digit. Here we just want to chop off the last 4 digits. It follows that dividing by 10000 will do the trick. (Notice that if we mod by 10000, that will reveal those last four digits. And more generally, dividing by 10^k chops off the last k digits and modding by 10^k reveals the last k digits.) Using this idea, and noting that we must output differently depending on whether or not the first 3 digits are 555 or not, here is a solution to fyi:

```
using namespace std;
#include <iostream>

int main() {

    int num;
    cin >> num;

    if (num/10000 == 555) {
        cout << 1 << endl;
    }
    else {
        cout << 0 << endl;
    }

    return 0;
}
```

If Statement Example: Quadrant Selection

As you might imagine, any type of statement can be placed inside of an if statement, including another if statement. In the following problem:

https://open.kattis.com/problems/quadrant

the input is a point on the Cartesian plane that does not lie on either the x or y axis, and you must determine which quadrant the point is in. This can be solved with a single if statement with four branches, but the following solution illustrates the idea of splitting up the work so that inside of an if statement, there is another if statement. The outer level if statement will check if x is positive. Then, inside of both the other level if and else, another if-else statement will check whether y is positive or not. In this example, the curly braces for the inner if have been omitted to show that they are not necessary (It turns out that the curly braces for the outer if are also not necessary, but they are left in for clarity.)

```
using namespace std;
#include <iostream>

int main() {
    int x, y;
    cin >> x >> y;
    int res = -1;

    if (x > 0) {
        if (y > 0)
            res = 1;
        else
            res = 4;
    }
    else {
        if (y > 0)
            res = 2;
        else
            res = 3;
    }

    cout << res << endl;
    return 0;
}
```