Week 4

Intro to Computer Science - CSI 1430

Hello and welcome to the weekly resources for CSI 1430: Introduction to Computer Science! This week's resource will be covering the course material that will be covered in week 4 of the course.

A good piece of advice for these first few weeks is to really make sure that you are getting all of your questions answered! If you let your questions build up over the course of the year, you might be missing key concepts that are crucial to your foundational coding. So whenever in doubt, stop by your professors office hours, not only do they know a great deal about coding, but they make the programs! These documents tend to be super good in going over concepts right before exams, as they are a concise and to the point review of the materials. But remember, this document cannot cover everything, so make sure you get help on any specific concepts you struggle with as well as go over lecture notes, programs, and the textbook.

Reminder: If you have any questions about these study guides, group tutoring sessions, private 30 minutes tutoring appointments, the Baylor Tutoring Youtube channel or any tutoring services we offer, please visit our website https://baylor.edu/tutoring or call our drop in center during open business hours. Monday-Thursday 9am - 8pm on class days (254) 710-4135.

Keywords: Conditional Branching, Logical Operators, Operator Precedence

---

TOPIC OF THE WEEK

Conditional Branching

A lot of the time when you are programming, you will want to determine a certain “course of action” based on the results of another variable or expression. This is where the if statement, if-else statement, and else statement come into play.

Say you want to display the letter grade you received on a test, based on numerical score. To do this, we can employ a couple conditional statements to achieve just that.

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

int main() {

    int testScore;

    cout << "Input your test score." << endl;
    cin >> testScore;

    if(testScore >= 90) {
        cout << "You got an A!" << endl;
    }
    else if(testScore >= 80) {
        cout << "You got an B!" << endl;
    }
    else if(testScore >= 70) {
        cout << "You got an C!" << endl;
    }
    else {
        cout << "You didn't pass :( " << endl;
    }

    return 0;
}
```

In this example, after the test score is input, the first if statement will compare that test score to the integer value 90. If the score is 90 or greater, then the cout statement presenting a letter grade of an “A” will be displayed. Otherwise, it will move to the next statement comparing it to 80. The same will happen for 70, until it reaches the final else statement. If the else statement is reached, then this means that none of the previous conditions have been evaluated to be true, and therefore there is an alternate statement; which in this case is that the test score was not a passing grade.

There are many implications of this that you will start to see as you create more complex programs, but the general basis is that if you want your programs to have a certain behavior based on a certain criteria, you need to use conditional statements to achieve that. It is also worth mentioning, that when you string a bunch of if-else statements together, only one of them will be processed. In terms of our example, A grade of a 95 would fit the criteria for an “A” letter grade, but also a “B” and a “C” since 95 is greater than 90, 80, and 70. However, the behavior of the if-else statement combats against that and only goes with the first encountered true statement.

OTHER IMPORTANT TOPICS

Logical Operators

In most cases, conditional branching is not going to be dependent on one variable’s “state” alone. Rather, it is going to be the result of multiple variables “states”. The combination of the different states and the operators that join them, will aggregate to a total combined Boolean value (true or false). There can be as many of these operators streamed together as needed. These logical operations are performed very similarly to arithmetic on numerical values, however in this case there are only two possible outcomes (being true and false).

Operator	Name	Effect
a && b	*Logical AND*	*Evaluates to true, only if a AND b are true*
a || b	*Logical OR*	*Evaluates to true, only if a OR b (or both) is true*
!a	*Logical NOT*	*Evaluates to true if a is false (or evaluates to false if a is true)*

It is always good to see an example of this. Let's pretend we have variables a, b, and c that are true, true, and false respectively, where the expression we want to evaluate is (a && (b || c)).

First, we would follow basic arithmetic rules that state that parentheses are the most highly-precedented operation. Therefore, we should evaluate (b || c) first. The OR operator states that only one of the two values must be true to result to true, then since b is true, then (b || c) is true. Now, our expression is condensed down to (a && true). The AND operator states that both values on either side of the operator must be true to result to true, and since a is true and that result of (b || c) is true, then the total aggregate expression (a && (b || c)) is true.

Operator Precedence

Another thing to remember is how the precedence of operators is ordered within C++. While it's not a super complicated subject, it is one I would recommend that you attempt to memorize, as there are many issues that can arise from not properly treating mathematical or relational expressions in C++ carefully. In my experience, I like to use a "more than needed" amount of parentheses when I write out my expressions. This tends to make sure that what you intend matches with what is actually being interpreted.

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In `(a * (b + c)) - d`, the + is evaluated first, then *, then -.
!	! (logical NOT) is next	`! x || y` is evaluated as `(!x) || y`
* / % + -	Arithmetic operators (using their precedence rules; see earlier section)	`z - 45 * y < 53` evaluates * first, then -, then <.
< <= > >=	Relational operators	`x < 2 || x >= 10` is evaluated as `(x < 2) || (x >= 10)` because < and >= have precedence over ||.
== !=	Equality and inequality operators	`x == 0 && x >= 10` is evaluated as `(x == 0) && (x >= 10)` because < and >= have precedence over &&. == and != have the same precedence and are evaluated left to right.
&&	Logical AND	`x == 5 || y == 10 && z != 10` is evaluated as `(x == 5) || ((y == 10) && (z != 10))` because && has precedence over ||.
||	Logical OR	|| has the lowest precedence of the listed arithmetic, logical, and relational operators.

Baylor CS 1430 zyBook

CHECK YOUR LEARNING

1. **What would be the outcome of the following logical expression? (a || (b && (!c && d)), where a = false, b = true, c = false, d = true**
2. **Write an if-else statement that displays "Hello World" to stdout if variables a and b are true, otherwise "Oh No!" is displayed to stdout.**
3. **What is the operator that has the highest precedence, no matter what?**

THINGS STUDENTS MAY STRUGGLE WITH

1. Logical operators do not always evaluate the entire expression statement. For example, if we have the expression (a || b), then the "a" variable will be evaluated for true or false first...if the "a" variable is true, then the rest of the expression will not even be looked at.
2. Understand that it can be super important which order you put your if statements in when conjoining them with if-else statements. This is because whichever statement evaluates to true first, will be the only one that gets processed.
3. Arithmetic can be mixed with boolean expressions. Therefore, expressions like ((8 != (52 - (4 * 20)) && true) can exist.
4. An if statement does not always have to have an accompanying else statement, but an else statement cannot exist without an if statement being present first. This same logic applies to if-else statements. They can only exist if a if statement precedes them.

Thanks for checking out these weekly resources! Don't forget to check out our website for group tutoring times, video tutorials and lots of other course resources: https://baylor.edu/tutoring. The answers to the 'Check Your Learning' questions are below.

Answers:

1. True
2.
```
if(a && b) {
    cout << "Hello World" << endl;
}
else {
    cout << "Oh No!" << endl;
}
```
3. Parentheses, ()