Week 6

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 6 of the course.

The semester might start getting to a point where you feel like there is a lot on "your plate". But just remember to reach out for help when you need it! Don't be afraid to attend group tutoring or reach out to your professors, as they are super willing to help out with things you are having a difficult time with. **Just as a reminder:** 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: *Looping, Files Streams*

TOPIC OF THE WEEK

Looping

There is often the need to execute the same block of code multiple times with some possible slight modifications. This is where we invoke the use of a loop. A loop is able to take a chunk of code, and execute it until some exit condition is met. There are different types of loops that can be used in multiple situations. They are all essentially doing the same things and can represent each other in some form or fashion, but there is often a "better choice" based on what your code is trying to achieve.

To show these different types, I will use the example of displaying the numbers 0-10 to stdout. This obviously can be achieved through hard coding the values 0-10 and printing them all out individually, but that is a lot more code than necessary and it is not concise. As programmers, we

should always be looking to make things clear to look at; as well as easier for ourselves. The code below shows this example 3 different times using the 3 different loop types. As stated above, all three of these loops do the exact same thing, although they are formatted differently.

Different loop types:

[While-loops]

```
int main () {

    int number = 0;

    while (number <= 10) {
        cout << number << endl;
        number++;
    }

    number = 0;

    do {
        cout << number << endl;
        number++;
    } while (number <= 10);

    for (number = 0; number <= 10; number++) {
        cout << number << endl;
    }

    return 0;
}
```

- The most basic loop type is the while loop. This can be pretty intuitive to understand. The body of the loop (everything in between the curly braces) will execute while the condition in the parentheses is true. This operates very similarly to how if-statements work, as it uses logical operators and expressions to evaluate if the condition is true or false, where true means that the loop will continue running. In this example the while loop runs until the variable *number* is greater than 10. Since the variable starts as 0 and increments every pass of the loop, then the numbers 0-10 are printed to stdout.

[Do-While-Loops]

- Do-while loops operate almost exactly the same as while loops, with one difference. The body of a do-while is executed once before the condition is evaluated. This means even if the condition is false, the body will still be executed once. In this example, the variable *number* starts at 0 just like before and runs through the loop once. Once hitting the condition it will be evaluated. Since the number 1 (one because the variable was incremented) is still less than or equal to 10, then the looping will continue until 0-10 is printed.

[For-Loops]

- For-loops might be the least intuitive. This type of loop is used when you know exactly how many iterations of the loop you want to go through. In our case, we know we want to loop starting at 0 and ending at 10. The **format for a for-loop** is as follows:
    - Initialization of the loop variable(s)
    - The loop condition
    - The loop variable increment/change condition

- Once a for-loop starts the initialization of the loop variables will take place. Immediately following, the loop condition will be checked. If the condition evaluates to true, then the loop body will execute. After the body is finished executing, then the loop increment is triggered before going straight back to evaluating the loop condition.
- *For-Loops would be the best choice for this example* since we know the number of loops that we want to perform, and it is the most clean and concise way of presenting the "code idea".

One thing to think about is in order for loops to work, there MUST be some progress happening that moves toward the exit condition. In our example, the progress was that the number was getting incremented each pass of the loop. If there is no progress then the loop will never exit.

HIGHLIGHTS

File Streams

Streaming in C++ is a way that we can read or write output to files. For example, if we wanted to read in a file that contained phone numbers and print them to the screen, we would need to invoke a stream to achieve this. In order to be able to use these streams, you must '#include <fstream>'. This is similar to how we have to include the string or cmath libraries to use their functionality in C++.

Stream types:

- *fstream*
  - This represents a general file stream. This can be used to read AND write from files.
- *ifstream*
  - This represents an input file stream. This can be used to read in data from files.
- *ofstream*
  - This represents an output file stream. This can be used to write data to files.

In order to use streams, there are a few things that you must know how to do.

1. To open and use a fstream, declare the stream like you would any other variable or object, and then perform *streamName.open("FileName");*

2. Once the stream is created and the open() is performed, we can check that the file has opened properly by doing *streamName.is_open()*. This returns a boolean value that can be used in a conditional statement to express if the file opened correctly or not.
3. To close a stream, just do *streamName.close()*. Always do this, as it is not only good programming practice, but also makes it so the compiler is not forced to close it for you.

Make sure that you use good programming practice when using streams. When you open a file, always check that it opened properly by doing a conditional statement. If the file opened correctly, continue as normal, otherwise display a statement that provides information that the file did not open correctly.

CHECK YOUR LEARNING

1. **If you wanted to write the string "Hello World" to a file, which of the streaming methods would be most appropriate?**
2. **What is the best loop type to use if you are certain that you want to run the body of the loop at least once before the conditional statement?**
3. **In a for-loop, what is the first thing that executes?**
4. **What will cause a loop to never stop looping?**

THINGS STUDENTS MAY STRUGGLE WITH

1. Most people don't really know the difference between fstream and the other two streams (ofstream and ifstream). In most cases, fstream will work as both an input and output stream. The use of ofstream and ifstream really boils down to what you are using the stream for. If you are just inputting from a file, then ifstream will suffice, and the same goes for outputting to a file with ofstream. However, fstream will work as well in both of these cases. fstream opens a file in user-specified mode, which will not be explored in this class. But be aware of its existence as it offers additional streaming functionality including a way to read from binary files and such.
2. ostream will create a file under the specified name if the file does not already exist. This will be placed in wherever your current Working-Directory is.

3. Make sure you are careful when writing loops. Think carefully about what the exit condition from the loop will be. If you write an exit condition that is not able to be achieved, or if the exit condition is incorrect, you can end up in an infinite loop. This means that the code will continue forever, with no exit, until the program is manually terminated. Make sure you think carefully about the code you are writing in these situations to avoid annoyingly buggy code.

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. ofstream
2. Do-While loop
3. Variable initialization, which will only happen once.
4. If no progress is being made, aka, no actions are being made to move towards the exit condition.