FINAL EXAM PREP

Intro to Computer Science - CSI 1430

This week is the last week of class, and typically in this week (and the surrounding weeks of the class) you are reviewing for the final exam. Please use the review below (as well as other resources from the semester) as a general overview of some of the key concepts taught in the course. Please take a look at all previous resources listed on our website to help you review for the final exam!

If you have any questions about these study guides, group tutoring sessions, private 30-minute tutoring appointments, the Baylor Tutoring 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. **The last day of tutoring in the drop-in center will be the last day of class.** To learn about additional resources available during Finals Week, please visit CASE in the West Wing basement of Sid Rich! Good luck on your final exam!

Keywords: *Final Review*

MAJOR REVIEW TOPICS

Input and Output (Week 3)

One good thing to start with reviewing for the final exam and practicum is C++ input and output and how we can do different types based on what we want to achieve.

INPUT:

In terms of input, we want to think about the different ways we might want to receive input. The *cin* object can be used for reading in things from stdin or file by using the extraction operator (>>). This is good for when we want to read word by word or anything that is space delimited. An example of when we want to use something like this is if we are counting the number of occurrences of a certain word in a file...then we could iterate through a file until EOF is reached… However, if we want to do things line by line, for something like a csv file

with table values, we might want to use the *getline()* function to read in the entire data set contained in the line and do manipulation based on that line. DON'T FORGET that if we want to ignore lines in a file or any sort of input (based on a specification or something along those lines), then we can use the *ignore()* function! Doing that is the safest way to do this and we can guarantee what we want to ignore based on a specific length or delimiter.

Another thing to note is the existence of the stringstream library. A stringstream is a good way that we can operate on lines that are retrieved and can be used as a temporary buffer. This is super important if we want to do things by line, but also do each line as if we were reaching from *cin*.

OUTPUT:

Output in C++ is pretty straight forward...We can use the *cout* object paired with the insertion operator (<<) to print things to stdout or a file. The insertion operator works on strings, ints, chars, or anything with a valid << operator. There are also so many valid output modifiers to format output to look a certain way or to make sure data can follow a certain pattern. The most common example of an output modifier is setprecision to get the output of floating point numbers to fit a certain number of decimal places.

Conditional Branching & Logical Operators (Week 4)

The concept of branching really dials down to "what needs to happen and when". This can be based on one or more different criteria, and we can have multiple scenarios that this can split off to. Branching can be done with the use of boolean variables and logical operators. The conditions that guard different branches of an if-statement are basically a long boolean expression that will simplify into a single true or false result. Therefore linking them together will be contingent on the type of operator that is present.

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)*

A good example of using logical operators is if we need to have a animal that is a cat and a animal color that is orange...then our conditional statement would look similar to:

```
if (animal == “Cat” && animalColor == “Orange”)
```

We can chain multiple statements together with if-else and else statements appended onto the initial if-statement… So if we have multiple conditions that we would like to be testing then we can put them in succession to each other. However, pay attention as only ONE of these statements will end up executing and the first valid condition that evaluates to true will be the one that is run.

Looping (Week 6)

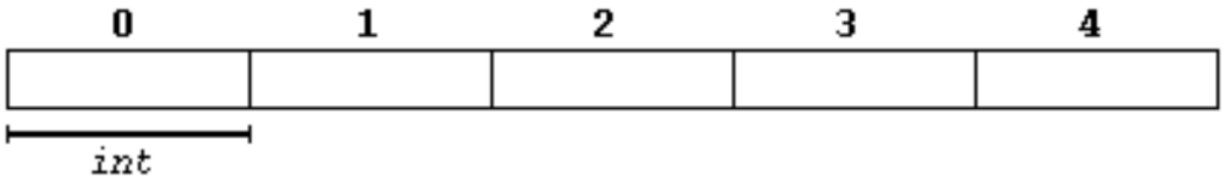
We often need to repeat a task when we are coding for a certain number of iterations OR until a certain condition is met. To achieve this, we can invoke the use of looping. The three main loops we will use in C++ are:

- For-loops
- While-loops
- Do-while loops

Take some time to understand why the different variants exist as there normally is a “best option” that fits every situation. However, most loops will be able to achieve the same results as all of the other loops. In a quick summary, for-loops are used when we know how many iterations of a loop that we would like to go through. While-loops are the most basic type of loop and will loop until the loop condition yields a false result. Finally, do-while loops are essentially the same as while-loops, except that the loop body will iterate once before the loop condition is evaluated. In all these cases we want to make sure that the content within the body of the loop is making some progress to make it so that the loop condition will eventually exist. Otherwise we encounter a situation where we never stop looping and enter what is called an infinite-loop.

Arrays / Vectors (Week 7 & 8)

Variables such as ints, doubles, and chars are ways that we can store a single value that can be referenced by a certain name. Many times we can encounter a situation where we need to store a list of like-variables (such as a list of grades) and want to do it as a collection rather than individually. Using arrays and vectors,we can group variables under one name. We can visualize an example of this with a list of 5 ints.



Each "cell" represents a single int value and each has an index number that identifies them. We can access each of these elements individually and the array or vector will retain the value placed at that specific index. Arrays and vectors, while having a similar purpose, have different implementations and uses. Arrays have a fixed size whereas vectors have a dynamic size and they can resize themselves. Therefore, arrays are length specific and are limited to the size that they are declared to, whereas vectors can grow in length as more and more elements are needed to be added to it.

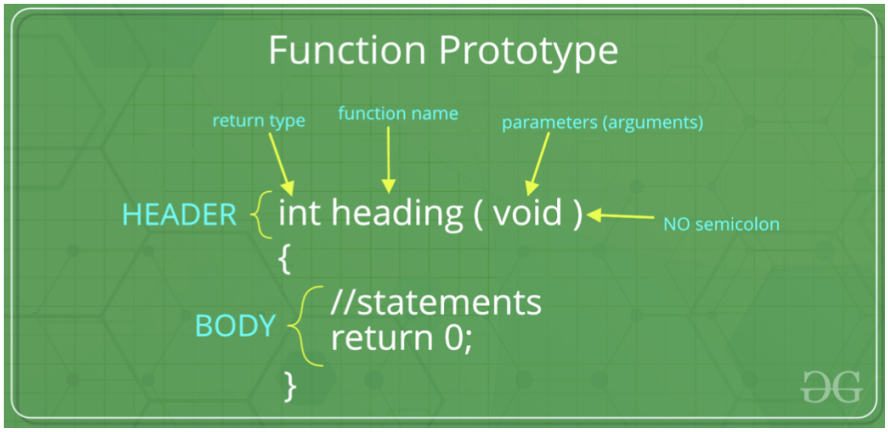
Classes and Structs (Week 11, 12, & 13)

Classes and structs are essentially ways that we can group together "like" data for the purpose of being more organized and packing the information into a single unit. This is part of a paradigm called Object Oriented Programming where there are ways we can define concepts like a Person or a Car that have multiple attributes to them within a new object structure. We not only can define the data that is contained within these object structures, but also some other functionality to access, modify and calculate specific things in regards to the object. Specifically, the most common type of function for every class are setters and getters. These are a way we can make the modification of the variables within a class a little more secure and so there is a little more protection over that data.

One thing to note and remember is the difference between structs and classes. While they both exist for different purposes, the only easily noticeable difference is the default type of access that the specifiers have. For classes, variables and functions within are private whereas for structs that are public.

Functions (Week 9)

Functions are essentially a way that we can group together chunks of code that we want to reuse or just to simply section off and compartmentalize our code. Plenty of function are prebuilt into the common STL C++ libraries such as *sqrt()*, but in order to make some of our own we must define them with the following structure:



In basic terms, within the function header, the return is what the "outcome" of the function will be, the function name is how our defined chunk of code will be called, and the parameters are what is needed to be passed to the function in order for a certain behavior to be achieved. The body of the function is responsible for defining the actual behavior of what the function will do. The entire body of a function is encapsulated in the curly braces of the function and normally ends in a return. A return to a function is not provided if it is a void returning function...which basically means there is no value being yielded from the function call.

CHECK YOUR LEARNING

1. **What type of access specifier do structs have by default? What about for classes?**
2. **Does an array or a vector automatically resize itself?**
3. **What does the || operator stand for? When does it evaluate to be true?**
4. **What are the three main parts of a function header?**
5. **What is the >> operator's name and is it paired with input or output?**
6. **What are the 3 types of loops? Which one is count controlled?**

THINGS TO GO OVER BEFORE THE FINAL

1. Make sure to go over the precedence of operators. Some of the behavior and ordering of how this works is not “natural” so make sure you look over that.
2. Files use the fstream library and allow us to read in from (or write data to) a file. Practicing with this and making sure that you can read from a file just as easily as using input will be super important.
3. Don't forget the difference between pass by reference vs. pass by value. In pass by value, the parameter value copies to another variable. However, in a pass by reference, the actual parameter passes to the function.
4. In terms of classes, I would make sure you understand how to create a class, make setters and getters, have a couple “calculation” functions, and use them. This is a very common practicum topic and will also show every aspect of what you have learned in this course so I would get very comfortable doing this quickly.
5. Make sure you don't just memorize how the different sorting algorithms are coded, but also that you understand what is happening. This makes recalling them and coding them up a lot easier. Also make sure you go over the space and runtime complexities of the sorting algorithms you have learned.
6. Also be sure to look back at previous weeks resources for in depth overviews of any topic you need to practice more. The previous week's resources go more into depth on each topic and have examples with code alongside them.

Answers:

1. Public. Private.
2. Vector
3. Logical OR. Evaluates to true when both sides of the operator are true.
4. Return type, name, and parameters
5. >> is the extraction operator and is paired with the *cin* object for input. Whereas << is the insertion operator which is paired with the *cout* object used for output.
6. For, while, and do-while loops. The for-loop is used when a known number of iterations should occur.

Thank you for using these resources this semester! Best wishes on your final exam!

Note: All tables were taken from the Baylor CS 1430 zyBook