**Megan Hacha**

**10/17/2024**

## Midterm Exam

### CS331

1. The placement of the prefix and postfix notations affect the resulting variable being incremented/decremented, but it changes the order of events. If you have a prefix (ex: int j = ++i), then the process’s order is incrementing i, then assignment of its new value to j. In the case of postfix (ex: int k = i++), it’s the other way around – assigning the value of i to k before incrementing i. An actual example might look like this:

int i = 0;

int j = ++i;

int k = i++;

cout << i << “ “ << j << “ “ <<k<<endl;

i starts with a value of zero, and does get incremented for both events, but the values of j and k are what differ. In this scenario, attributing the value of ++i to j means that its value will be 1 along with i’s since i was incremented prior to its value being assigned to j. For k, though, it’s given i’s original value of 0 and i is then incremented. The print statement would show the following numbers: 2 1 0.

1. **Programming question, answer is included in .cpp file**
2. Pointer arithmetic allows a pointer to access elements of an array via incrementation and decrementation. When a pointer is initialized with an array’s address, it points to the beginning, or the item at index 0. By incrementing, the pointer can move forward through the array. Decrementing does the same type of thing, but it instead moves the pointer to the previous array index. An example of this is provided in the .cpp file submitted alongside this doc.
3. **Programming question, answer in .cpp file**
4. Initially, the first things to note are the fact that there are 6 rows and we’ll be iterating through them 6 times. Inside that initial for loop, it looks like it’s adding spaces based on the difference between the current row it’s on and the value of i.
   1. The initial for loop will have an i value of 1, which means the for loop containing j will print “ “ 5 times.
   2. When that loop has finished, the next for loop will start. This one is what prints the asterisks, and when i = 1, it will print out 1 asterisk as (2 \* i – 1) = 1 and it starts with j = 1.
   3. The resulting row will look like the following (underscores used to denote spaces):

\_ \_ \_ \_ \_\*

* + 1. The spaces and \* are on the same line as there is no \n or endl present in this set of loops. Before the outer loop is i = 2, though, there is an endl, which means the next set of spaces and \*s will begin on a new line.
  1. The second row will have 4 spaces and 3 asterisks. So it will look like this:

\_ \_ \_ \_\* \* \*

* 1. This repeats until the outer loop is done, resulting in this pattern:

\_ \_ \_ \_ \_\*

\_ \_ \_ \_\*\*\*

\_ \_ \_\*\*\*\*\*

\_ \_\*\*\*\*\*\*\*

\_\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*

* 1. The spaces are decreased by 1 with each loop via the int j <= rows – i condition, and the asterisks are added so it follows a sequence of odd-only numbers starting with 1.