CS 121 SI Week 4 Worksheet - Nested For Loops and Text File IO

**Nested For Loops:**

By now we're worked with the three types of loops: **while, do-while,** and **for**. Although we can make nested while and do-while loops (and you will in 131), nested for loops are much more common in 121 than the other two.

Consider the following code (which is like the one in lecture but with tweaks):

const int ROW = 4;

const int COL = 4;

for(int r = 0; r < ROW; ++r)

{

for(int c = 0; c < COL; ++c)

{

cout << "\* ";

}

cout << endl;

}

In console, we would get the following output:

\* \* \* \*

\* \* \* \*

\* \* \* \*

\* \* \* \*

Taking it step-by-step, the logic isn't too difficult. In short: The outer loop follows by the number of rows (**ROW**), and the inner loop goes with the number of columns (**COL**).

For a longer explanation: The outside loop will run ROW times (4) and the inside will run COL times each time ROW loops (in our case, that is **ROW** \* **COL,** or 4 \* 4 = 16 times), with the endl at the end of each row.

If both explanations are still confusing, the two best ways to learn the concept are by either tracing (i.e. looking at what happens each step in the program) or compiling the code yourself. Preferably both.

Note also that, in each for loop we have initialized integers named **r** and **c** respectively. This helps us note what was mentioned before: Outer loop goes with row, inner loop goes with column.

Try changing **COL** to another integer (anything that isn't 4). What will happen to the console output? What would happen if we instead tried changing **ROW** to another value?

The cool thing about working with nested for loops is that you can come up with many different printable shapes. For instance, the below will print out a right triangle (with its corner in the bottom left):

int r, c; //another way of writing this

int BASE = 3;

for(r = 0; r < BASE; ++r)

{

for(c = 0; c <= r; ++c)

{

cout << "\* ";

}

cout << "

}

Edit the above code so the corner is in **a)** the top left and **b)** The top right. You should only need to change two parts of the original code for **a)**, however **b)** requires a bit more editing. For **b)**, think about the number of spaces required before printing stars.

**HINT:** When working with printing shapes, you're main thought should be "How does everything relate to one another?". Finding patterns (e.g. how do row and column relate) are good starts.

Another cool shape we can work with is an equilateral triangle. If you recall way back from geometry class: An equilateral triangle is a triangle with all sides equal (and each corner is 60 degrees). There are two ways to go about it: One with two inner for loops, and another with absolute value. Here's the first one:

int r, c; //row and column

int side = 3; //equal sides

for(r = 0; r < side; ++r)

{

for(c = r; c < side; ++c) //spaces; inner loop #1

{

cout << " ";

}

for (c = 0; c < ((2 \* r) + 1); ++c) //asterisks; inner loop #2

{

cout << "\*";

}

cout << endl;

}

Try to solve the absolute value version. It should only require two loops (an outer loop and an inner loop):

Although the above is mainly for fun and games, nested for loops will mainly be used for accessing cells within a two-dimensional array. For instance, assume the two-dimensional integer array below, **grid,** has already been initialized. If we were to access position row one, column three, the code would be:

grid[0][2];

Using the above information, we can print out the entire contents of the variable **grid**.

Write a code segment to print out the contents of **grid**. Assume you already have variables **ROW** and **COL** as your constraints, and that the 2d array is filled completely (i.e. not a partially-filled two-dimensional array).

const int ROW; //assume initialized

const int COL; //assume initialized

int grid[ROW][COL]; //assume initialized

**Text File Input/Output:**

At first, text file input/output might seem like a completely new and scary topic to deal with. In reality, it's not far off from using the simple cin/cout statements we're already used to. For grabbing input from files, use the object **ifstream**, whereas for outputting to a file use **ofstream**.

Both inputting files and outputting files follow the same flow structure, only differing in step **2i.**:

1. Attempt to open the file
2. If the file was opened
3. Read from the file (**ifstream**) or write to the file (**ofstream**)
4. Close the file (**IMPORTANT**)
5. If the file could not open, let the user know that an issue occurred.

Translating this to C++, we have the following:

ifstream infile; //object for grabbing input from files

infile.open("example.txt"); //some .txt located in the same folder

if(infile.is\_open()) //if the input file was successfully opened

{

//grab whatever we want from the file by "infile >> ..." or getline

infile.close();

}

else

{

cout << "Error: Could not open input file!\n";

}

Grabbing Input:

Just like **cin**, use either the extraction operator (**>>**) or **getline** to grab input from a file.

Example:

infile >> ch; //works just like cin, except now we grab from file

getline(infile, str); //same as getline(cin, str);

As an exercise, try to make sense of what the following code segments are doing. Write comments:

if(infile.is\_open()) //

{

while(infile >> ch) //

{

//do whatever

}

}

----------------------------------------------------------

if(infile.is\_open())

{

while(getline(infile, str) //

{

//do whatever

}

}

----------------------------------------------------------

if(infile.is\_open()) //

{

while(infile.good()) //

{

getline(infile, str[i]); //

i++; //

}

}

----------------------------------------------------------

if(infile.is\_open()) //

{

while(infile.good()) //

{

infile >> list[i]; //

++i; //

}

}

----------------------------------------------------------

Writing/Outputting to a Text File:

Almost exactly like cout. Same steps as before, just replacing a few bits from the input file portion.

C++:

ofstream outfile("example.txt"); //a way to combine declaration and open()

if(outfile.is\_open())

{

outfile << "This is an example output line\n";

}

else

{

cout << "Error when writing to output file!\n";

}

There isn't really too much to go on from here about each type of file. Just know that "is\_open()" for input files will fail when the file cannot be found, whereas for output files "is\_open()" fails when the location it's writing to is inaccessible, when the disk does not have enough space, or the path length exceeds the operating system's limit (failure when working with output files are rare though).

Noting that **ifstream** and **ofstream** work pretty much like **cin** and **cout**, we can use this and the previous concept to efficiently read and write to text files.

Given the following variables and contents of a text file, write the code to read from a file and input the values into **num\_table**, and then output it afterwards. Also, the input file name is "input.txt", and the output is "output.txt".

Text File:

0 1 2 3 4

5 6 7 8 9

10 11 12 13 14

15 16 17 18 19

20 21 22 23 24

Given code:

const int ROW; //initialize this based off of above

const int COL; //initialize this based off of above

int row\_count; //count for number of rows

int column\_count; //count for number of columns

int num\_table[ROW][COL]; //2d array to be written into and outputted to

HINT: For the input, you will use counters to keep track of the row and column you are working on. For the output, use **ROW** and **COL** in a nested for loop to control the output (the output file should be the exact same as the input).

NOTE: Normally you would ask the use for the values ROW and COL (since, when grabbing input from a text file, we don't really know how many values to expect). To simplify things in the above example, we defined ROW and COL ourselves.