SYDE 121
Lab Number 04

New Requirement for All Lab Exercises: To make it easier for your TAs to grade your assignments, from now on you will have to indicate the degree of success of your implementation in the header above the main function. If the program runs successfully, state that the "Program runs successfully." If the program is not working then provide a brief point-form explanation.

Note: you do not require a TA to sign your plan, programs, or outputs unless explicitly stated in the lab.

Exercise 1: Display a Square

Learning Objectives: To verify data entry, use while loops, and use if statements.

Read This First

You are expected to use 'while' loops to complete this exercise. You are also expected to use only a **single** 'if-else' command when displaying the box (this will require a bit of thought). Do NOT use 'for' loops.

What To Do

Write a program (square.cpp) that reads in the size of the square and prints a hollow square of that size using asterisks and blanks. Your program must make a decision of whether to print a '*' or ' ' at a particular location. Keep in mind that, using the DOS window, the output can only be displayed from left to right and (when you reach the end of a line) from top to bottom. Your program should only accept and work only for even side sizes between 2 and 20. For example, if your program reads in a size of 6, the following output would be displayed:

```
* * * * * *
*         *
*         *
*         *
*         *
* * * * * *
```

What Not To Do

If you want full marks, do **not** write multiple and separate while loops for each component of the square i.e. one loop for the top, another for the sides, and yet another for the bottom. You only require two while loops, one nested in the other. Ask your TA for some direction if you are lost on how to do this properly.

If you are only able to design a solution using multiple and separate while loops, part marks will be provided.

What To Hand In

Email your program to the course account. Submit a hard copy to the submission box with the rest of the material for this lab.

Exercise 2: A simple loop

Learning Objectives: Practice writing a simple loop.

Read This First

The factorial function denoted $n!$ (and read "n factorial") is defined for non-negative integer numbers by the formula:

$$n! = \begin{cases} 1 & n = 0 \\ 1*2* \ldots *n & n >= 1 \end{cases}$$

What to Do

Write a program that asks the user for a number, ensures it is greater than or equal to zero, and computes the factorial. Use a variable of type **unsigned int** to store the computed factorial. Use a 'for' loop to implement a solution.

The methodology used is called an **iterative** solution ie. your loop should allow the same operation to be performed over and over until the stopping criterion is met.

What is the largest value of n that the program can correctly handle? What happens when the program breaks?

What to Hand In

Email the code to the course account and submit a hardcopy of the code to the course submission box.

Answer the indicated two questions and submit these to the course submission box with your code.

Exercise 3: Random number generation

Learning Objectives: How to create random numbers using the **rand** function. Use these random numbers within a program with indicated design specifications.

Read This First

This exercise is more challenging than the first two. Random numbers are generated by using a particular function in the C++ standard library. The **rand( )** function randomly and with equal probability generates an integer between zero and **RAND_MAX.** **RAND_MAX** varies from system to system (you do not need to know the value of **RAND_MAX** to use this function). A random number is assigned to the variable **i** by the following function call:

```
int i = rand( );
```

What to Do

Write a program that will help an elementary school student learn multiplication. Use **rand** to randomly produce two positive single-digit integers e.g., 0, 1, 2, … 9 (you need to put some thought into how to do this). The program should then pose a question such as:

How much is 6 times 7?

If the student's response is correct, congratulate them and then ask another question. If the answer is incorrect, indicate this and then repeatedly ask the same question until the correct answer is given. Allow the user to exit at any time by entering –1. After deciding to quit, indicate the number of questions the student answered correctly *on the first try* for each different question asked eg. "5 questions were answered correctly on the first try out of 7 questions total".

You should plan out the algorithm for this carefully ahead of time and only then go ahead and try to code it. Design and implementation of this algorithm can be a bit tricky.

Note

You should notice that **rand** produces the same numbers in the same order each time the program is run. This is intentional and allows for consistent code debugging. The **srand** command is used to prevent this from happening, but you are not expected to use this here.

Do not use the 'break' command (if you happen to know what I mean about this) to exit the loop (reasons for this will be discussed during lecture time).

What to Hand In

Email the coded solution to the course account and submit a hardcopy of the code to the course submission box.

All materials for all exercises due Friday, October 7 by 6:00pm.