SYDE 121
Lab Number 04

As stated in Lab #3, you are expected to continue using the design methodology used in Lab #2. Although this design will not be submitted nor graded, the onus is on the student to ensure that their designs are prepared prior to trying to code the solution. It is highly recommended that you develop the habit of planning your program before you type it into the computer.

Exercise 1: 2-Dimensional Vector Operations

Learning Objectives: To verify data entry, use **while loops**, and gain experience with **switch statements**.

Read This First

As you are learning in some of your other SYDE courses this term (e.g. 113, 181), vectors can be used to represent key physical phenomenon such as forces, and are often used during engineering design and analysis. To computationally model such real-world phenomenon, we often need to implement vectors and vector operations programmatically. For 2-dimentional space (i.e. a plane), key vector operations for the vectors

$\mathbf{u} = \langle U_1, U_2 \rangle$

$\mathbf{v} = \langle V_1, V_2 \rangle$

are:

$\mathbf{u} + \mathbf{v} = (U_1 + V_1)\mathbf{i} + (U_2 + V_2)\mathbf{j}$ **(vector addition)**

$\mathbf{u} \cdot \mathbf{v} = U_1V_1 + U_2V_2$ **(dot product)**

$\mathbf{u} \times \mathbf{v} = U_1V_2 - U_2V_1$ **(cross product)**

The vector components U_1, U_2, V_1, and V_2 each represent points in the 2D plane $U_1=(x_{u1}, y_{u1})$, $U_2=(x_{u2}, y_{u2})$, $V_1=(x_{v1}, y_{v1})$, $V_2=(x_{v2}, y_{v2})$.

In this assignment, you will obtain these base point values from the user, build the **u** and **v** vectors, and perform selected vector operations on these vectors.

You are expected to a use **while loop** to implement the user menu, and a **switch statement** to determine the appropriate action to take for a given user input. Do NOT use **for loops**.

What To Do

Write a program, *lab0401.cpp*, which reads in the four points, U_1, U_2, V_1, and V_2, as integer values, displays the corresponding **u** and **v** vectors, and then asks the user to select which type of vector operation they would like to perform using the following menu.

```
Select a vector operation to perform:
  1: u + v (addition)
  2: u x v (cross product)
  3: u . v (dot product)
Enter 1-3 from the menu above, or -1 to exit:
```

Use a while loop to implement this menu, so that the user can perform as many vector operations as they wish, until they enter -1 to exit the program. Use a **switch statement** to handle the input and to perform the vector operations.

An example of output from this program would be:

```
2-Dimensional Vector Operations
Vector u has components U1 and U2.
Vector v has components V1 and V2.

Please enter the components for u.
First enter U1 as two integers: 1 1
Now enter U2 as two integers: 2 2

Please enter the components for v.
First enter V1 as two integers: 3 3
Now enter V2 as two integers: 5 5

You have entered:
U1 = (1,1)
U2 = (2,2)
V1 = (3,3)
V2 = (5,5)
```

```
The vectors u and v are:
u = 1*i + 1*j
v = 2*i + 2*j

Select a vector operation to perform:
1: u + v (addition)
2: u x v (cross product)
3: u . v (dot product)
Enter 1-3 from the menu above, or -1 to exit:
1

u + v = 3*i + 3*j

Select a vector operation to perform:
1: u + v (addition)
2: u x v (cross product)
3: u . v (dot product)
Enter 1-3 from the menu above, or -1 to exit:
2

u x v = 0

Select a vector operation to perform:
1: u + v (addition)
2: u x v (cross product)
3: u . v (dot product)
Enter 1-3 from the menu above, or -1 to exit:
3

u . v = 4

Select a vector operation to perform:
1: u + v (addition)
2: u x v (cross product)
3: u . v (dot product)
Enter 1-3 from the menu above, or -1 to exit:
-1
```

You could even provide more explicit instructions to the user to help with correct input of the points, for instance, you could specify a valid type or range of input (e.g., positive integers). It is not required in this exercise, but it is also good programming practice to verify that the user input correct values.

Create a “*README_Lab0401.txt”* file, and create detailed instructions on how to run your *lab0401.cpp* program, including what input should be entered (see Previous Labs for instructions on README files.)

What To Hand In

Submit both your *lab0401.cpp* files and *README_Lab0401.txt* files to the Lab Assignment 4 LEARN dropbox.

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:

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

What to Do

Write a program, *lab0402.cpp*, which 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 i.e., your loop should allow the same operation to be performed over and over until the stopping criterion is met.

Create a “*README_Lab0402.txt”* file, and create detailed instructions on how to run your *lab0402.cpp* program, including what input should be entered (see Previous Labs for instructions on README files.)

What to Hand In

Submit both your *lab0402.cpp* files and *README_Lab0402.txt* files to the Lab Assignment 4 LEARN dropbox.

Exercise 3: Exploring Data Types

Learning Objectives: Understanding the difference between integer data types, unsigned int and int.

What to Do

Create a text file, lab0403.txt, and use it to record the answers to the following questions.

In the previous program (lab0402.cpp):

a) What is the largest value of n that the program can correctly handle?

b) What happens when the program breaks?

Change the **unsigned int** variable used for the factorial to the data type **int,** and rerun the program.

c) What is the largest value of n that the program can correctly handle?

What to Hand In

Submit the *lab0403.txt* file to the Lab Assignment 4 LEARN dropbox.

Exercise 4: Random number generation

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

Read This First

This programming 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, *lab0404.cpp*, which 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.

Create a “*README_Lab0404.txt”* file, and create detailed instructions on how to run your *lab0404.cpp* program, including what input should be entered (see Previous Labs for instructions on README files.)

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 to exit the loop, but instead design an appropriate stopping condition for your loop.

What to Hand In

Submit both your *lab0404.cpp* files and *README_Lab0404.txt* files to the Lab Assignment 4 LEARN dropbox.

All materials for all exercises due Thursday, October 11 by 11:59pm.