**Practical 4 (Part B)**

**Repetition Control Structure**

1. Trace the output of the following code:

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

int N;

cout << "Enter number: ";

cin >> N;

for (int i=1; i<=N\*N; i++)

{

cout << setw(5) << i;

if (i%N == 0)

cout << "\n";

}

return 0;

}

1. What will happen to the output if we removed the setw(5) from the code?
2. Modify the code to display the output in opposite way.
3. Given the algorithm below, write a C++ program that uses loop to output all odd numbers between *firstNum* and *secondNum*.

**Algorithm**

Read *firstNum* and *secondNum*

For *count* between *firstNum* and *secondNum*

If count divided by 2 remains 1

Display count

End if

End loop

1. Write a ***for*** loop that prints out the multiples of 3 up through 75, separated by a blank space.

3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75

1. Write a C++ program that read a positive integer value, and compute the following sequence: If the number is even, half it; if it's odd, multiply by 3 and add 1. Repeat this process until the value is 1, printing out each value. Finally print out how many of these operations you performed. If the input value is less than 1, print an error message and end the program. Typical output might be:

Initial value is 6

Next value is 3

Next value is 10

Next value is 5

Next value is 16

Next value is 8

Next value is 4

Next value is 2

Next value is 1

Final value 1, number of steps 8

1. Given the following algorithm, write a program that reads an integer from user, then prints a message stating whether it is a **prime number \*** or not.

**NOTE:** Prime numbers are integers (greater than 1) and give no remainder only when divided by 1 and by itself. E.g. the first 10 prime numbers:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...

|  |
| --- |
| **Algorithm:**  **Divide the input integer by every number smaller than itself to see if there is any remainder.** |
| Prompt user for an integer  Read the integer into *n*  IF *n* is less than 2  Set *prime* to false ( i.e. n is not a prime number)  ELSE  Set *prime* to true (i.e. *n* is a prime number )  FOR values of *i* from 2 to *n* – 1 ( i represents all numbers smaller than n )  IF *n* divided by *i* gives no remainder  Set *prime* to false (i.e. n is not a prime number )  BREAK ( jump out from the loop )  END IF  END FOR  END IF  IF *prime* = true  Print *n* is a prime number  ELSE  Print *n* is not a prime number  END IF |

1. Write a C++ program that asks the user to enter a number that is in the range of 10 to 50, inclusive. If the number is valid, then the program prints the message "Valid". If it is not within this range, the program prints "Invalid". The message "Done" is printed in both cases. The program will run until user asks to stop.
2. Write a C++ program that repeatedly collects positive integers from the user, stop when the user enters a negative number or zero. After that, output the product of all positive entries. A sample run should appear on the screen like the text below.

Enter a number: 3

Enter a number: 10

Enter a number: 2

Enter a number: -213

The product of all your positive numbers is 60.

1. Write a C++ program that requests lower and upper integer limits, calculates the sum of all the integer squares from the square of the lower limit to the square of the upper limit, and displays the answer. The program should then continue to prompt for the limits and display answers until the user enters an upper limit that is equal to or less than the lower limit. A sample run should look like the following:

Enter lower and upper integer limits: **2 8**

The sums of the squares from 4 to 64 is 203

Enter next set of limits: **5 30**

The sums of the squares from 25 to 900 is 9425

Enter next set of limits: **3 3**

Ended

1. Write a full C++ program, starting from #include, that asks the user to enter an integer and reports all divisors in ascending order. That is, list all numbers that evenly divide the given number, sorted from smallest to largest. You may assume the user enters a positive integer. An example is shown below, where the user entered 30.

Enter a number: 30

The divisors are: 1 2 3 5 6 10 15 30

Which loop will you use, **while** or **for** loop? Explain.

1. Write a program that reads an integer (positive, negative or zero) from user. After that, use a **do-while** loop to count the **number of digit(s)** in the integer. Finally display the count as shown in the following sample output:

|  |
| --- |
| Enter an integer > **4131025**  Number 4131025 contains 7 digit(s) |

1. Write a program that asks the user to enter a list of integers and key in sentinel value -999 to stop. The program is to determine the smallest value entered and the number of times it was entered. For example, if the following series is entered:
2. 16 11 15 14 18 20 13 11 12 19 18 11 17 20

it would output: **The smallest value is 11 and it was entered 3 time(s).**

1. Write a C++ program, which will display a neat multiplication table (1 - 12) for any number chosen by the user. Example of output (if the user entered **13**):

|  |
| --- |
| Multiplication table of what number? **13**  13 X 1 = 13  13 X 2 = 26  : : : : :  : : : : :  13 X 12 = 156 |

1. Write a C++ program to perform the sum as below.

                    sum = 1 + (1+2) + (1+2+3) + ... + (1+2+3+4...+n)

**n** can be any integer value.

1. Write a C++ program to process a collection of daily high temperatures. Your program should count and print the number of hot days (high temperature 85 or higher), the number of pleasant days (high temperature 60-84), and the number of cold days (high temperature 1-59). It should also display the category of each temperature. Test your program on the following data:

55 62 68 74 59 45 41 58 60 67

65 78 82 88 91 92 93 87 80 78

79 72 68 61 59

Modify your program to display the average temperature (a real number) at the end of the run.

[Note: the program will stop counting when 0 or less than 0 is entered.]