SEHH2042 Computer Programming

Tutorial 4 – Repetition statements (2)

Q1. Write a program **Factor.cpp** that accepts user input of a positive integer and prints all factors of that input number. The program allows user to further input more numbers until a non-positive value is entered. The sample result below shows the program output.

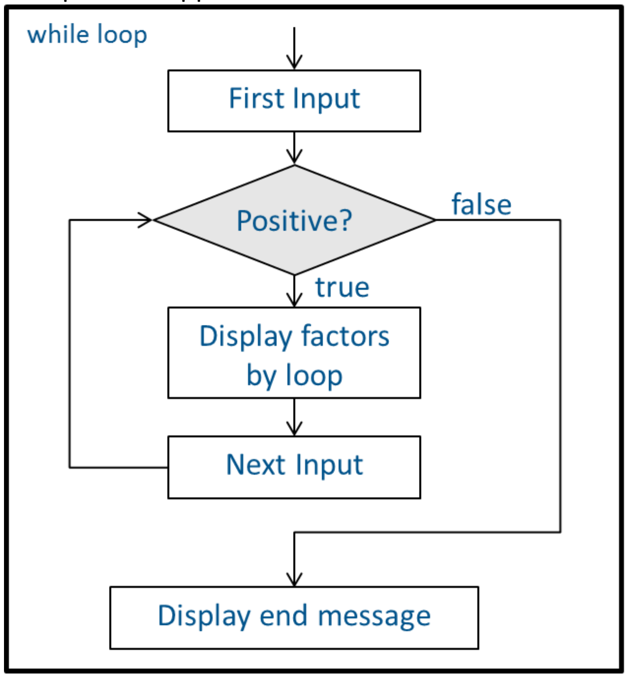
Sample result:

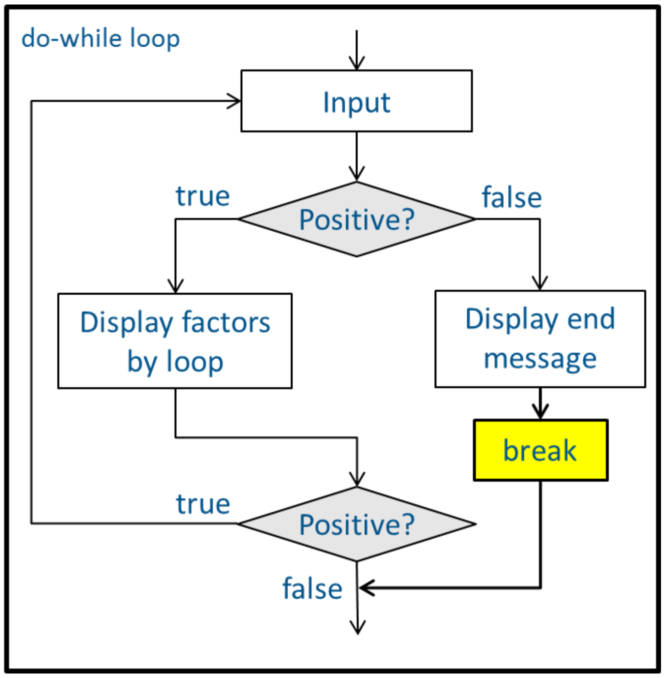
```
Enter a positive integer: 7
Factors of 7: 1 7
Enter a positive integer: 20
Factors of 20: 1 2 4 5 10 20
Enter a positive integer: 56
Factors of 56: 1 2 4 7 8 14 28 56
Enter a positive integer: -2
Only positive integer is accepted. Program ends.
```

Think about the following first:

(1) What header files/library are required?
(2) How many variables are needed?
(3) What is/are the data type(s) of variable(s)?
(4) What are the steps and calculations involved?

Two possible approaches:





Let’s use **while loop**. Follow the steps below to write the program:

1. Include necessary header files/library and write the main block.

```
#include <iostream>
using namespace std;

int main()
{
     // code of subsequent steps
     return 0;
}
```

2. Declare necessary variables.

```
    int input;
```

Factors are not variables here. They are loop counter in the later step.

3. Ask user to input the first number.

```
    cout << "Enter a positive integer: ";
    cin >> input;
```

4. Start the **while loop** with condition of “positive input”.

```
    while (input > 0) {

         // display factors by loop,
         // then ask for next input.

    }
```

Continuation condition controls the loop body should repeat again or not. If input is positive, repeat the steps in body.

5. In the while loop body, use **for loop** to check every number in the range from 1 to input.

```
    cout << "Factors of " << input << ": ";

    for (int n = 1; n <= input; n++) {

         // check if n is a factor and display.

    }
    cout << endl;
```

For loop is within the while loop. This structure is a nested loop.

After showing all factors by for loop, end the line and ready for further input.

6. In the for loop body, check whether n is a factor of input. If yes, display it.

```
    if (input % n == 0)
         cout << n << " ";
```

Need to have space before showing next factor.

7. After the for loop, i.e. after displaying all factors, do step 3 again to ask for next user input.

8. After the while loop, i.e. the input value is not positive, display the end message.

```
    cout << "Only positive integer is accepted. Program ends.";
```

Try to implement this program using **do-while loop** by re-arranging the above steps.

Q2. By referring to T3 Q5, modify the program **Prime.cpp** to print all prime numbers within the range 1 to 200. The result is printed with 10 numbers per row and with 5-character width for each number. No user input is required in this program.

Hint: You need a counter variable to count how many prime numbers are printed so far, then determine whether a new line should be inserted.

Sample result:

```
    2    3    5    7   11   13   17   19   23   29
   31   37   41   43   47   53   59   61   67   71
   73   79   83   89   97  101  103  107  109  113
  127  131  137  139  149  151  157  163  167  173
  179  181  191  193  197  199
```

Q3. Write a program **CalcPI.cpp** which calculates the value of PI using the formula below. The result is displayed with 10-character width for the first column and 20-character width for the second column. Use 15 decimal places for the value of PI.

PI = 4/1 – 4/3 + 4/5 – 4/7 + 4/9 – 4/11 + 4/13 – ……

Examples:

Use 5 terms: PI = 4/1 – 4/3 + 4/5 – 4/7 + 4/9
Use 10 terms: PI = 4/1 – 4/3 + 4/5 – 4/7 + 4/9 – 4/11 + 4/13 – 4/15 + 4/17 – 4/19

Sample result:

```
     Terms         Value of PI
     -----         -----------
        10   3.041839618929403
       100   3.131592903558554
      1000   3.140592653839794
     10000   3.141492653590034
    100000   3.141582653589720
```

Q4. Write a program **Pattern.cpp** which displays the a pattern according to the user input size (a positive integer). Use nested loop in your answer.

Sample result:

```
Pattern size: 5
*****
*   *
*   *
*   *
*****
```

Think like this:
Treat nested loop counters as x-y coordinates, use if-else to determine each point should be * or space.

Implement the following patterns (e.g. size = 7):

(a)	(b)	(c)	(d)	(e)
`*******`	`*******`	`*******`	`*******`	`*******`
`*     *`	` *`	`     *`	` *   *`	`**   **`
`*     *`	`  *`	`    *`	`  * *`	`* * * *`
`*     *`	`   *`	`   *`	`   *`	`*  *  *`
`*     *`	`    *`	`  *`	`  * *`	`* * * *`
`*     *`	`     *`	` *`	` *   *`	`**   **`
`*******`	`*******`	`*******`	`*******`	`*******`