CISS240: Introduction to Programming
Assignment 8

Objectives

1. Write simple (non-nested) for-loops
2. Write simple (non-nested) for-loops containing if/if-else statements.
3. Write simple (non-nested) for-loops contained inside if/if-else statements.

Q1. Write a program that prompts the user for an integer n and prints a table of squares, cubes, and fourth powers from 1 to n. In the output below, the column widths are all set to 10. Assume that the user will always enter an integer greater than 0.

Test 1.

```
3
         i       i^2       i^3       i^4
 --------- --------- --------- ---------
         1         1         1         1
         2         4         8        16
         3         9        27        81
```

Test 2.

```
5
         i       i^2       i^3       i^4
 --------- --------- --------- ---------
         1         1         1         1
         2         4         8        16
         3         9        27        81
         4        16        64       256
         5        25       125       625
```

Test 3.

```
10
         i       i^2       i^3       i^4
 --------- --------- --------- ---------
         1         1         1         1
         2         4         8        16
         3         9        27        81
         4        16        64       256
         5        25       125       625
         6        36       216      1296
         7        49       343      2401
         8        64       512      4096
         9        81       729      6561
        10       100      1000     10000
```

Q2. Write a program that prompts the user for two integers, `start` and `end`, and prints a table of squares, cubes, and fourth powers from `start` to `end`. In the output below, the column widths are all set to 10. Assume that the user always enters integers greater than 0.

Test 1.

```
1 3
         i       i^2       i^3       i^4
 --------- --------- --------- ---------
         1         1         1         1
         2         4         8        16
         3         9        27        81
```

Test 2.

```
5 10
         i       i^2       i^3       i^4
 --------- --------- --------- ---------
         5        25       125       625
         6        36       216      1296
         7        49       343      2401
         8        64       512      4096
         9        81       729      6561
        10       100      1000     10000
```

Test 3.

```
10 20
         i       i^2       i^3       i^4
 --------- --------- --------- ---------
        10       100      1000     10000
        11       121      1331     14641
        12       144      1728     20736
        13       169      2197     28561
        14       196      2744     38416
        15       225      3375     50625
        16       256      4096     65536
        17       289      4913     83521
        18       324      5832    104976
        19       361      6859    130321
        20       400      8000    160000
```

Q3. The following program finds the integer roots within a range for a given cubic polynomial. Recall that a cubic polynomial is a polynomial of degree three. For instance

$$2x^3 + 5x^2 + (-1)x + (-6)$$

is a cubic polynomial. We say that a real number r is a root of the above polynomial if when you substitute $x = r$ into the polynomial, you get zero:

$$2r^3 + 5r^2 + (-1)r + (-6) = 0$$

For instance $x = 1$ is a root since

$$2(1)^3 + 5(1)^2 + (-1)(1) + (-6) = 2 + 5 - 1 - 6 = 0$$

The program accepts 6 integers a, b, c, d, e, f and lists the integers from e to f (inclusive) which are roots of the polynomial

$$ax^3 + bx^2 + cx + d$$

Test 1.

```
2 5 -1 -6 0 2
1
```

(i.e., 1 is the only integer root in $[0, 2]$ for $2x^3 + 5x^2 + -1x + -6$.)

Test 2.

```
0 1 0 1 -10000 10000

```

(i.e., $0x^3 + 1x^2 + 0x + 1$ does not have any integer root in $[-10000, 10000]$.)

Test 3.

```
1 0 -1 0 -100 100
-1 0 1
```

(i.e., -1, 0, and 1 are the integer roots of $1x3 + 0x2 + -1x + 0$ in $[-100, 100]$.)

Q4. Write a program that accepts an integer value n from the user and then poses n multiplication problems to the the user. Each question involves the product of two random numbers between 0 and 9. If the user answers correctly, he/she gets a point. After asking n questions, the program prints the score.

To make testing easier, instead of doing this:

```
...

int main()
{
    srand((unsigned int) time(NULL));

    ... YOUR CODE ...

    return 0;
}
```

you MUST do the following (i.e., fix the seeding of the random generator):

```
...

int main()
{
    srand((unsigned int) 0);

    ... YOUR CODE ...

    return 0;
}
```

Test 1.

```
1
8 * 9 = 72
1
```

Test 2.

```
1
8 * 9 = 0
0
```

Test 3.

```
3
8 * 9 = 72
8 * 7 = 56
5 * 7 = 35
3
```

Test 4.

```
3
8 * 9 = 0
8 * 7 = 0
5 * 7 = 35
1
```

Test 5.

```
3
8 * 9 = 72
8 * 7 = 0
5 * 7 = 35
2
```

Test 6.

```
3
8 * 9 = 2
8 * 7 = 2
5 * 7 = 2
0
```

Test 7.

```
10
8 * 9 = 1
8 * 7 = 1
5 * 7 = 1
5 * 5 = 1
0 * 2 = 1
3 * 0 = 1
2 * 1 = 1
7 * 1 = 1
5 * 5 = 1
7 * 0 = 1
0
```

Q5. Write a program that prints all 3–by–3 magic squares. (The program does not accept any input.) The following is part of the printout containing 2 magic squares:

```
+-+-+-+
|2|7|6|
+-+-+-+
|9|5|1|
+-+-+-+
|4|3|8|
+-+-+-+
+-+-+-+
|2|9|4|
+-+-+-+
|7|5|3|
+-+-+-+
|6|1|8|
+-+-+-+
```

Make sure the format of the printing of the magic squares matches the above partial sample output. Also, the magic squares MUST be printed in the following order:

In the above printout

```
+-+-+-+
|2|7|6|
+-+-+-+
|9|5|1|
+-+-+-+
|4|3|8|
+-+-+-+
```

appears before

```
+-+-+-+
|2|9|4|
+-+-+-+
|7|5|3|
+-+-+-+
|6|1|8|
+-+-+-+
```

because the first magic square is related in an obvious way to the integer 276951438 which is less than 294753618 and 294753618 is related in an obvious way to the second magic square.

Obviously, your program is meant to discover the magic square on its own. If you simply write a program to print the above magic squares then that's considered cheating and you will

get an immediate −10. Your program must go through all the possible 9–digit configuration of the magic square in order to discover the magic squares.

Your program must be reasonably efficient. In other words, points will be taken off if you have not thought about ways to optimize your program.

Note. A 3–by–3 magic square is a 3–by–3 grid filled with digits from 1 to 9 in such a way that all digits appear exactly once, the sum of each row, column, and diagonal gives you the same value. For instance, the following:

```
+-+-+-+
|2|7|6|
+-+-+-+
|9|5|1|
+-+-+-+
|4|3|8|
+-+-+-+
```

is a magic square since each row, column and diagonal adds up to 15. In general, one can also talk about 4–by–4 magic squares, 5–by–5 magic squares, etc. But this question is solving for only the 3–by–3 case.

Q6. Recall that a sequence is simply an ordered list.

A sequence of numbers is in ascending order if the integers are arranged from the smallest to the largest. In other words, they are increasing as you go through the sequence. For instance, $2, 9, 11$ is a sequence in ascending order.

Consecutive integers are integers that follow right after each other in an uninterrupted succession, i.e., they always differ by 1. For instance, $4, 5$ and 6 are consecutive. Note that they also happen to be ascending.

Based on the discussion above, it should be obvious that a sequence of consecutive integers in ascending order is thus a list of integers that are arranged from the smallest to the largest such that each term is 1 larger than the preceding term. $4, 5, 6$ is an example of such a sequence. $7, 8, 9$ and $10, 11, 12$ are couple of other examples that fit the bill.

Write a program that accepts a strictly positive (> 0) integer n from the user and then, finds and prints the longest sequence of consecutive digits in ascending order in the integer n going from left to right.

For instance, the longest sequence of consecutive ascending digits in the integer 123475634 is 1234, 456 is the longest consecutive ascending sequence in 1294563 and the longest sequence of consecutive digits in the ascending order in the integer 123956780 is 5678.

If there is a tie, i.e., there are two or more sequences with the same number of consecutive ascending digits which are longer than all other such sequences, print the one that occurs first. Therefore, if the user enters `126784563`, the program should print `678`.

You must discard leading zeroes in the input but zeroes inside the integer can lead the sequence. For instance, if the user enters `01234`, your program should print `1234` but if the user enters `56701234`, your program should print `01234`.

Try all the test cases and devise some more of your own. Note that the largest value that an `int` variable can hold is $2,147,483,647$ so you should only test your code with integers that are at most 9 digits long or you will make the world explode!

Test 1.

```
126784563
678
```

Test 2.

```
123475634
1234
```

Test 3.

```
1294563
456
```

Test 4.

```
123956780
5678
```

Test 5.

```
34040123
0123
```

Test 6.

```
0123456
123456
```

Test 7.

```
56701234
01234
```

Test 8.

```
1203424
12
```

Test 9.

```
78945678
45678
```

Test 10.

```
10437586
1
```

Test 11.

```
11234566
123456
```

Test 12.

```
3456789
3456789
```

You can see that every integer is itself a sequence of digits. Therefore, you are actually finding the longest consecutive sequence of digits from a longer sequence that isn't necessarily consecutive. In other words, you essentially are searching for a particular subsequence in a larger sequence. This has very important applications that you will see in the future. For instance a huge part of computational biology (bioinformatics) is concerned with finding patterns in DNAs.