Level 2: Nested Loops

Loops are used when we want to do the same job more than once. It might be completely the same or might have slight differences. But we can do each job by running the same piece of code. For example, the code on the left prints 10 asterisks '*' which is basically the same job at each loop. The code on the right prints the numbers from 1 to 10 which is the same job with different parameters. In both cases the pieces of code run in each loop are the same.

```
for (int i = 0; i < 10; ++i)        for (int i = 0; i < 10; ++i)
  fout << "*";                        fout << i << " ";
```

What if in each loop instead of doing one job, we want to do several jobs which are again basically the same. '*The hands of an analog clock*' is a good example to this. From noon to midnight, the hour hand of a clock shows 12 different numbers in this interval. If we consider 0 and 12 means the same for the hour hand, we can get all those values by using a `for` loop.

```
for (int hourhand = 0; hourhand < 12; ++hourhand)
  fout << hourhand << endl;
```

Suppose we would like to write all the minutes in an analog clock. We know that there are 60 minutes in an hour, and for each number the hour hand points in the clock, the long hand points 60 different numbers. Incrementing the minute by one is a job. Doing it 60 times is a loop of the same kind of job. In other words, incrementing the hour 12 times, and incrementing the minute 60 times whenever the hour is incremented is a loop of jobs in another loop. This is called nested loops. For instance, the code below prints all the minutes between 00:00 and 11:59.

```
for (int hourhand = 0; hourhand < 12; ++hourhand)
  for (int minutehand = 0; minutehand < 60; ++minutehand)
    cout << hourhand << ":" << minutehand << endl;
```

Example: Write a program which prints all two-digit numbers that can be composed of {0, 1, 2, 3, 4, 5}.

Solution 1: We will try to do it without using nested loops – only using one `for` loop. In this problem two digit numbers start from 10 and goes up to 55. Be careful that 1s digit starts from 0 and goes up to 5. Hence, we will only write the two digit numbers between 10 and 55 which has 1s digit less than or equal to 5.

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

int main ()
{
  for (int number = 10; number <= 55; ++number)
    if (number % 10 <= 5)
      cout << number << endl;
}
```

Solution 2: Now, we will use nested loops instead. Using nested loops, we just need to make a loop for 10s digit from 1 to 5, and for each increment in 10s digit, we make an inner loop for 1s digit from 0 to 5.

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

int main ()
{
  for (int tens = 1; tens <= 5; ++tens)
    for (int ones = 0; ones <= 5; ++ones)
      cout << tens << ones << endl;
}
```

The second code seems easier to think than the first one. The first code uses an indirect way to solve the problem, whereas in the second code the solution method is obvious.

A Better Example: Write a program which prints all two-digit numbers that can be composed of {0, 1, 2, 3, 4, 5} in the following order:

50 51 52 53 54 55 40 41 42 43 44 45 … 10 11 12 13 14 15

Be careful that 1s digit increases while 10s digit decreases.

Solution 1: The solution is much harder than the previous one. We have to create a formula to do that. We will again start from 10 and go up to 55. However, this time we will write ‘`6 – tens digit`’ and 1s digit together so the 10s digit decreases while 1s digit increases.

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

int main ()
{
  for (int number = 10; number <= 55; ++number)
    if (number % 10 <= 5)
```

```
      cout << 6 – (number / 10) << number % 10 << endl;
}
```

Solution 2: Now, we will use nested loops instead. Using nested loops, we just need to make a loop for 10s digit from 5 down to 5, and for each decrement in 10s digit, we make an inner loop for 1s digit from 0 to 5.

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

int main ()
{
  for (int tens = 5; tens >= 1; --tens)
    for (int ones = 0; ones <= 5; ++ones)
      cout << tens << ones << endl;
}
```

Compared to the first solution, the second solution is much easier to think.

Exercise 1: Write a program which prints all two-digit odd numbers that can be composed of {0, 1, 2, 3, 4, 5, 6} using nested loops.

* *Exercise 2*: Solve the same problem without using nested loops.

Exercise 3: Write a program which prints all three-digit numbers that can be composed of {0, 1, 2, 3, 4} using nested loops.

* *Exercise 4*: Solve the same problem without using nested loops.

Exercise 5: Write a program that reads two numbers N and M from the input file '*figurein.txt*'. Then, your program will print a figure in '*figureout.txt*' that has N number of rows and M number of columns. The first row will have with M number of 1s, and the second row will have M number of 2s. It will continue until the Nth row will have M number of Ns. See the example below for clarity.

Sample Input (figurein.txt):

```
3 4
```

Sample Output (figureout.txt):

```
1111
2222
3333
```

** *Exercise 6*: Solve the same problem without using nested loops.

Exercise 7: Write a program that reads a number N from the input file ‘*figurein.txt*’. Then, your program will print a figure in ‘*figureout.txt*’ that has `N` rows in triangle from. The first row will have only 1, and the second row will have 1 and 2. It will continue until the `N`th row will have the numbers from 1 to N. See the example below for clarity. Be careful about the blank between numbers.

Sample Input (figurein.txt):

```
4
```

Sample Output (figureout.txt):

```
1
1 2
1 2 3
1 2 3 4
```

** *Exercise 8*: Solve the same problem without using nested loops.

Exercise 9: Write a program that reads a number N then N numbers from the input file ‘*figurein.txt*’. Your program will print a figure in ‘*figureout.txt*’ that has N rows. The numbers given in the input specify the number of ‘*’ in their row consequently. See the example below for clarity.

Sample Input (figurein.txt):

```
5
3 4 6 0 3
```

Sample Output (figureout.txt):

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

***
```

Exercise 10: Write a program that reads n < 1000 then n numbers each of which is between 1 and 1000 inclusively from the input file ‘*infile.txt*’. Then, your program will find the numbers that exist more than once and write them in ‘*outfile.txt*’.

** *Exercise 11*: Solve the same problem without using nested loops.

Challenging Problems

Exercise 12: Adapted from Amicable Numbers (BIO'96)

Two numbers are said to be "amicable" if they are different and the sum of the divisors of each number (including 1 but excluding the number itself) equals the other number.

For example: 2620 is divisible by 1, 2, 4, 5, 10, 20, 131, 262, 524, 655, and 1310; these add up to 2924. 2924 is divisible by 1, 2, 4, 17, 34, 43, 68, 86, 172, 731, and 1462; these add up to 2620. Therefore 2620 and 2924 are amicable. Find the amicable numbers between 1 and 5,000.

Exercise 13: Write a program which finds the number of primes that are less than the given number.

Sample input:
```
20
```

Sample output:
```
8
```

There are 8 primes less than 20; 2, 3, 5, 7, 11, 13, 17, and 19.

Exercise 14: Write a program that finds the maximum sum subsequence given a sequence of numbers. You will first read *n* then *n* numbers from the input file *numbers.txt*. Write the maximum sum to *sum.txt*. There will be maximum 100 numbers, and each number will be less than 100.

Sample input:
```
20
9 -7 0 11 -99 2 2 3 1 0 3 1 0 2 -8 5 4 -7 3 1
```

Sample output:
```
15
```

The subsequence is 2 2 3 1 0 3 1 0 2 -8 5 4.