Unit 4: Looping

1. Do-while Loop

Exercise 1: Write a program that outputs your name 10 times.

Question: What happens if we are asked to write 100 times? Should we write the same line 100 times?

General form of `do-while` loop is:

```
do
{
  (expression)
  (expression)
    :    :
}
while (condition);
```

`do-while` loop repeats the expressions between the curly braces while the `condition` is satisfied.

Example 1: Write a program that outputs numbers starting from x + 1 to x + 100 where x is given.

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

int main()
{
  int counter;               // counter for the loop
  int x;
  cin >> x;                  // read x from keyboard
  counter = 1;               // initially the counter is 1

  do                         // start of the repetition block
  {
    cout << x + counter << endl;
    counter = counter + 1;   // increase the counter by 1
  }
  while (counter <= 100);    // while counter is less than or
                             //    equal to 100 execute this loop
}
```

NOTE:

Be careful about the first and the last steps. In the above example, initially `counter` is 1. In the last step, after writing x + 100 to screen, `counter` becomes 101 and violates the condition.

The following table shows the output and the value of `counter` at each step in the loop. Suppose user inputs `6` from the keyboard.

Step	*Output*	`counter`
Initial		1
1	7	2
2	8	3
:	:	:
:	:	:
99	105	100
100	106	101

The following code also does the same work. This time counter starts from zero. As soon as loop starts, `counter` is incremented by 1. So, x + 1 is written on screen in the first step. The condition is counter < 100 since the loop must end as soon as counter reaches to 100.

```
counter = 0;                    // initially the counter is 0
do                              // start of the repetition block
{
  counter = counter + 1;
  cout << x + counter << endl;
}
while (counter < 100);  // while counter is less than 100
```

Example 2: Write a program that reads numbers and outputs if they are greater than 5. Program will stop when 0 is given.

Sample input:

```
11 8 2 2 4 7 5 0
```

Sample output:

```
11
8
7
```

Solution:

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

int main()
```

```
{
  int number;
  do
  {
    cin >> number;            // read the number from keyboard
    if (number > 5)           // if the number is greater than 5
      cout << number << endl;
  }
  while (number != 0);        // while number is not equal to 0
                              //    execute this loop
}
```

Exercise 2: Write a program that finds the count of the given numbers. Program will stop when 0 is given.

Sample input:

```
11 8 2 2 4 7 5 0
```

Sample output:

```
7
```

Exercise 3: Write a program that reads numbers and outputs if they are greater than x. Program will stop when 0 is given. The user will first input the number x.

Sample input:

```
7
8 2 2 4 7 5 11 0
```

Sample output:

```
8
11
```

Exercise 4: Write a program that outputs the numbers from 1 to n. The user should give the value of n.

Sample input:

```
6
```

Sample output:

```
1
2
3
4
5
6
```

Exercise 5: Write a program that finds the n^{th} power of 2 where n will be given as input and n > 0.

Sample input:

```
5
```

Sample output:

```
32
```

Question: If the user enters 5, how many times does the loop repeat?

The following table shows the values of the variables `counter` and `power` at each step in the loop.

Step	`counter`	`power`
Initial	0	1
1	1	2
2	2	4
3	3	8
4	4	16
5	5	32

Exercise 6: Write a program that finds the sum of n numbers given. The user should enter n, then the numbers from keyboard.

Sample input:

```
5
22 1 12 4 8
```

Sample output:

```
47
```

The following table shows the values of the variables `counter`, `number`, and `sum` at each step in the loop for the sample input.

Step	`counter`	`number`	`sum`
Initial	0		0
1	1	22	22
2	2	1	23
3	3	12	35
4	4	4	39
5	5	8	47

Exercise 7: Write a program that reads numbers and outputs if they are between 17 and 35 inclusively. Otherwise program stops.

Sample input:

Here '>' specifies the inputs by the user, the rest is the output by program.

```
>19
 19
>25
 25
>32
 32
>16
```

There are some shortcut operations in place of some arithmetic operations. They are given in the following table:

Operator	*Example*	*Equivalent expression*
`++`	`++number`	`number = number + 1`
`--`	`--number`	`number = number – 1`
`+=`	`number += 10`	`number = number + 10`
`-=`	`number -= 10`	`number = number - 10`
`*=`	`number *= 10`	`number = number * 10`
`/=`	`number /= 10`	`number = number / 10`

2. While and For Loops

Besides `do-while` loop we have two more loop types; `while` loop and `for` loop. We can write the program given at the beginning using while loop as follows:

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

int main ()
{
  int number, counter = 0, isEven, square;
  cin >> number;

  while (counter <= 10)
  {
    isEven = number % 2;
```

```
    if (isEven == 0)
    {
      square = number * number;
      cout << square << endl;
    }
    ++number;
    ++counter;
  }
}
```

The only difference with `do-while` loop is the location of condition check; `do-while` checks at the end whereas `while` checks at the beginning. Hence, to get the same result while loop has to be executed one more time. That's why, we changed the condition from '`counter < 10`' to '`counter <= 10`'.

`do-while` always enters the loop at least once, however if the conditions doesn't hold `while` loop will not be executed even once. Check the following codes:

```
int counter = 1;
while (counter == 0)
{
  cout << "Hello!" << endl;
  ++counter;
}
```

```
int counter = 1;
do
{
  cout << "Hello!" << endl;
  ++counter;
}
while (counter == 0);
```

The code one the left doesn't enter the loop since '`counter = 1`' at the beginning and the condition '`counter == 0`' doesn't hold. However, the right one enters the loop once and prints '`Hello!`' since the condition is checked at the end.

Another loop type is `for` loop.

General form of `for` loop is:

```
for (initialization; loop condition; loop counter update)
{
  (expression)
  (expression)
    :    :
}
```

In the `initialization`, a loop counter is defined and initialized. Loop repeats the expressions between the curly braces while the `loop condition` is satisfied. The loop counter is updated in the `loop counter update` section. Note that loop counter is updated at the end of each run of the expression in the loop.

The code with the other loop types can be written with `for` loop as follows:

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

int main ()
{
  int number, isEven, square;
  cin >> number;

  for (int counter = 0; counter <= 10; ++counter)
  {
    isEven = number % 2;
    if (isEven == 0)
    {
      square = number * number;
      cout << square << endl;
    }
    ++number;
  }
}
```

As you see, sometimes it is more practical to use `for` loop since it has loop counter definition and initialization, condition check, and loop counter update sections.

We can also leave empty any of the initialization, loop condition, and loop counter update sections. For instance, we can change the `for` loop in the above code as follows:

```
  int counter = 0;
  for ( ; counter <= 10; ++counter)
```

We just defined the counter before the `for` loop and left the initialization part empty.

3. Infinite Loops

Consider the program below:

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

int main()
{
  int number;
  number = 1;

  do
  {
    if (number == 1)
      number = 61;
    number += 2;
  }
  while (number > 50);
}
```

Question: How many times is the loop repeated? Why?

Consider the program below:

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

int main()
{
  int counter;
  counter = 1;

  do
  {
    counter += 2;
  }
  while (counter != 18);
}
```

Question: How many times is the loop repeated? Why?

NOTE: Be careful about the loop condition; avoid infinite loops.