Loops allow a program to execute the same set of commands multiple times. Without them writing programs that did repetitive tasks would be extremely tedious (even with “cut and paste”).

The three types of loops in C/C++ are for, while, and do-while. As an example, the basic structure of the for loop is:

```c
for( command 1 ; command 2 ; command 3 )
{
    // statements to be
    // executed
    // during the loop.
}
// rest of the program.
// Not part of the loop.
```

As is usual in C/C++ the curly braces define the block after the for and can be omitted if only a single statement is used after the loop. For example,

```c
for( command 1 ; command 2 ; command 3 )
    // Only statement that is in the loop.
// The rest of the program.
```

(Note the indenting that follows the for statement. As with most formatting this is not required for the program to compile correctly, but it does make the program more readable; without proper indentation it is almost impossible for a person to understand what a program does. You will lose points if you don’t correctly indent your programs.)

`command 1` is executed once at the beginning of the loop. It is used to initialize the loop and commonly defines a ‘counter’—a variable that is used to count the number of times a loop has been executed. A typical command is:

```c
int i = 0
```
which defines a new variable `i` and sets its value to 0.

`command 2` is tested at the end of every loop. If it is false (zero) the loop ends. Thus it should be a predicate. A typical command is:

```c
i < 100
```
which ends the loop when `i` is equal to or greater than 100.

`command 3` is executed at the end of every loop, before the conditional (command 2) is tested. It is usually used to increment the counter. A typical command is:

```c
i++;
```
which increases i by one. This shorthand operator will be discussed in more detail during lecture.

For example, the following loop repeats exactly 100 times.

```c
for( int j = 0 ; j < 100 ; j++ ) // loop 100 times
{
    // commands to be
    // executed
    // during the loop.
}
```

**Exercise 4.1**

This exercise illustrates the importance of looping for highly repetitive tasks. Only the for loop is used.

1. *Without* using any kind of loop, write a program that has the user enter (read) exactly five integers, and calculates and prints the sum of the integers.

2. Write a program that will do the same thing using a for loop.

3. Write a new program, still using a for loop, that allows the user to enter up to 100 integers and again prints the sum. However, if at any point the user enters a zero then the loop halts and the sum of the numbers already entered is printed. You can end the loop early by setting the counter variable equal to any value that will cause the loop to halt. (Imagine how hard it would be to write this program without the ability to loop.)

4. Rewrite the program in the previous step to use:
   (a) a while loop.
   (b) a do-while loop.

5. Turn in all three programs with sample output showing test cases, etc.

**Exercise 4.2**

Nested loops are loop constructs that have one loop within another loop. They are often used for printing tables of data. In using nested loops don’t forget to indent properly to make it clear what sections of code are part of which loop.

In this exercise we will look at the ASCII values for different characters by building a table of a portion ASCII values.

The table should roughly look like:
This table shows, for example, that the ASCII value 40 (40+0) represents the character '(' and the ASCII value 41 (40+1) represents the character ')'. We will not worry about the values below 40 or above 119 (many of which don’t print).

To create this table you will need to use nested loops. For the rows you may want to count from 40 to 110 in steps of 10. The other loop, for the columns, will need to count from 0 to 9.

The value for each entry in the table will be the character corresponding to the ASCII value of the sum of the two loop variables. For example, if the loop variables have values 100 and 5 respectively then the table entry will be the character corresponding to the ASCII value 105. To print this character use the `static_cast` command:

```cpp
cout << static_cast<char>(x+y) ;
```

where `x` and `y` are the two loop variables.

You will need to be a little careful about where you put spaces and new lines, to get the table to look correct.

As always turn in the code and the output. In this case, because there is no input, there is only one output file.