Subject #7: Loops

- Write the following program in a file named *factorial.c*:

```
#include <stdio.h>

/* calculate the factorial of an integer. */

main()
{
    int n, fact ;

    /* read n */
    printf("enter a number:\n");
    scanf("%d",&n) ;

    printf("%d! = ", n) ;

    /* perform the calculation */
    fact = 1 ;
    while (n > 1) {
        fact *= n ;
        n-- ;
    }

    /* print the result */
    printf("%d\n", fact) ;
}
```

- Compile and run the program.
- The *while* statement is used to perform a statement or a group of statements as long as a certain condition is true. The form of the *while* loop is:

  while (*condition*)
      loop operation

  The loop operation can be a single statement, for example:

```
while (x < 1000)
    x *= 2 ;
```

or a block of statments, enclosed in braces:

while (*condition*) {
    . . .
}

- Here is a different way to perform the same calculation:

```
#include <stdio.h>

/* calculate the factorial of an integer. */

main()
{
    int n, fact ;
    int i ;

    /* read n */
    printf("enter a number:\n") ;
    scanf("%d",&n) ;

    /* perform the calculation */
    fact = 1 ;
    for (i = 1; i <= n; i++)
        fact *= i ;

    /* print the result */
    printf("%d! = %d\n", n, fact) ;
}
```

- The *for* statement has the following structure:

for (*statement1* ; *condition* ; *statement2*)
    statment3 ;

This is completely equivalent to:

statement1;
while (*condition*) {
    statement3 ;

    statement2 ;
}

statement3 can be replaced by a group of statements, enclosed by braces.

- Note that in the first version of the program, n changed in the loop and therefore had to be printed before it. In the second version, the variable i changed but n did not.

- The choice between the *while* and *for* loops is up to you — choose the one that makes the program more readable. The *for* statement is typically used when the loop requires an initialization that is a single statement, and a single statement that is related to it and should be performed each time at the end of the loop. The most common use of the *for* statement is to perform a certain operation several times:

```
int i ;

for (i = 0; i < n; i++) {
    ...
}
```

  Note that it would usually be undesirable to change the value of i within the loop.

- Note that the computation of the factorial might result with an overflow for rather small values of `n`. You can try by yourself and see at what value this happens. For large values of `n`, the factorial can be computed as a *float* using, for example, the Stirling formula.

- Create a file break.c with the following program. Try to understand what the program will do. Then see what it actually does by compiling and running the program.

```
#include <stdio.h>

#define MAXN 100

main()
{
    int i, j ;

    for (i = 1; i <= MAXN; i++) {
        for (j = 2; j < i; j++) {
            if (i % j == 0)
                break ;
        }
        if (j == i)
            printf("%d\n", i) ;
    }
}
```

- This program demonstrates that a loop can be put within another loop. This is called *nesting*. When nesting is used, indentation is very important to make the program readable by people.

- The *break* statement terminates the execution of the innermost loop in which it is inserted.

- It is best to avoid "magic numbers" like 100 from appearing at the body of the program. Therefore, MAXN is defined as a *symbolic constant* in the **#define** line. Every time the token MAXN is encountered, it is replaced with 100. Therefore, it does not have to be declared, and the **#define** line should not end with a semicolon. It is customary to write symbolic constants with capital letters.

- By the use of the symbolic constants, the program is clearer, and if we decide to change the range of numbers checked, we can do it very easily.
- Note that the program could be made more *efficient* by making j run from 2 to the square root of i. Here is a modified version of the program:

```
#include <stdio.h>
#include <math.h>

#define MAXN 100

main()
{
    int i, j ;
    int root ;

    for (i = 1; i <= MAXN; i++) {
        root = sqrt(i) ;
        for (j = 2; j <= root; j++) {
            if (i % j == 0)
                break ;
        }
        if (j > root)
            printf("%d\n", i) ;
    }
}
```

- The function sqrt() returns the square root of an expression of type *double*, in a form of a *double*. A *double* is similar to a *float*, with a higher precision. In the line

  ```
  root = sqrt(i);
  ```

  the variable **i** is an *int*, so it is *cast* to a *double*. The variable **root** is also of type *int*, and the assignment converts the *double* to an *int* by keeping only the integer part.
- The *continue* statement is similar to the *break* statement, but instead of terminating the loop, it terminates the current iteration of the loop, and causes the next one to begin.

Classwork: Converting binary numbers into decimal
Write a program that reads a binary number as an integer and prints its decimal value. For example, the binary number 1011 should be converted into the decimal number 11.