Subject #4: A First Program in C

The files containing the C programs should be created, and then *compiled*, *linked* and *run*.
To get acquainted with the process, follow the next steps:

1. use **xemacs** to create the file *first.c* containing the following program:

```
#include <stdio.h>

main()
{
    /* declaration block */
    int i, j ;

    /* execution   block */
    i = 5 ;
    printf("Please enter an integer number:\n") ;
    scanf("%d", &j) ;
    printf("At the beginning, i = %d and j = %d\n", i, j) ;
    j = i ;
    i = j ;
    printf("At the end,       i = %d and j = %d\n", i, j) ;
}
```

2. % **ls** | Look at the contents of the directory to see that the file *first.c* exists
3. % **cat** *first.c* | Look at the contents of the file
4. % **gcc** *first.c* | Compile and Link the file
5. If errors are reported during compilation edit the file again and correct the errors.
6. Look at the contents of the directory again to see that the program **a.out** was created.
7. % **a.out** | Run the most recently compiled program. Enter an integer number when asked, and press **Enter**.
8. Look at the output produced by the program. What is wrong ?
   Correct the program so that the values of *i* and *j* would be exchanged.
9. Compile the program again and check that your correction fixed the program.
10. Run again the program with the input coming from a file instead of the keyboard:
    - Use XEmacs to create a file named *input.dat*. In the file write a single line with an integer number in it. Save the file.
    - In the shell, write:
      % **a.out** < *input.dat* | Run the most recently compiled program with the file *input.dat* as input.

Subject #5: Types, Variables, Operations and Expressions

- Each variable has a type. The basic arithmetic types are
  - *int*, for integers
  - *float*, for floating–point real numbers
- Each variable has to be *declared* first, so that memory is allocated to it, and its type is known. The declarations are put in a declarations block.
- After the declarations block comes the execution block, where the variables can be used for computations, printing, etc. Every variable must be assigned a value before being used, or else it will contain garbage. Assignment can be performed also in the declaration.
- The escape sequences used to print and read those types using *printf* and *scanf* are
  - %d for *int*s (in decimal notation)
  - %f for *float*s
- Using operations, one can build various expressions from variables and constants. We shall now list the binary arithmetic operations:

– `i + j`	Addition
– `i - j`	Subtraction
– `i * j`	Multiplication
– `i / j`	Division

- An operation between two equal types gives a result of the same type. When an operation is performed between an *int* and a *float*, the *int* is first promoted to a *float*.
- Division of two *int*s gives an *int* result which is the integer part of the ratio.
- When we have an expression including several operations, we must know what is the order in which the operations are applied. Each operation has a *precedence* – operations with higher precedence are carried out before those of lower precedence. When there are several consecutive operations of the same precedence, we should know whether they *associate* from left to right or vice versa.
- The precedence and associativity of the binary arithmetic operations are the conventional ones. The multiplicative operations have higher precedence than the additive ones, and they all associate from left to right.
- A table of the precedence and associativity of all the operations taught in this page is given in its end.
- Parentheses can be used to override the normal precedence and associativity.
- There are also two unary sign operators, `+` and `-`.
- Explicit *casting* can be used to convert an expression to a different type. For example, if `i` is an *int*, then `(float) i` is a float equal to it. If `x` is a *float*, then `(int) x` is an *int* equal to its integer part.
- The assignment operator `=` translates between types exactly as the cast operation.

- Write the following program in a file *operators.c* .

```
#include <stdio.h>

main()
{
    int i = 7, j, k ;
    int a, b, c ;

    a = 1 + 2 * 3 ;
    b = 4 - 5 + 6 ;
    c = +8 * -i ;

    printf("1 + 2 * 3 = %d \n", a) ;
    printf("4 - 5 + 6 = %d \n", b) ;
    printf("+8 * -i = %d \n\n", c) ;

    a = (4*3)/2 ;
    b = 4*(3/2) ;
    c = 4*3/2 ;

    printf("(4*3)/2 = %d \n", a) ;
    printf("4*(3/2) = %d \n", b) ;
    printf("4*3/2 = %d \n\n", c) ;

    j = (int) 99.9 ;
    k = -99.9 ;
    printf("j = %d, k = %d \n", j ,k) ;
}
```

  Before running the program, compute all the printed results. Then compile and run the program.

- There are additional assignment operators corresponding to the binary arithmetic operations. For example, `i += 10` is equivalent to `i = i + 10`.
- All the assignment expressions return the value being assigned. Therefore, the command `i = j = 3` makes sense. The expression `j = 3` assigns the value 3 to the variable `j`, and also returns the value 3. This value is then assigned to the variable `i`.
- There are also the increment `++` operator and the decrement `--` operator. For example, `i--` is equivalent to `i -= 1` (and to `i = i - 1`). They come both as prefix and as postfix forms. `i++`, for example, increments `i` *after* its value is used, while `++i` increments `i` *before* its value is used.

- Add to your program a declaration of the following variables:

```
int jj, kk ;
```

and add the following lines at the end of your program:

```
jj = j++ ;
kk = ++k ;
printf("j = %d, k = %d, jj = %d, kk = %d \n", j, k, jj ,kk) ;
kk += jj ;
printf("Now kk = %d, jj = %d\n", kk, jj) ;
```

Compile and run the program. What happened to each of the variables?

- **Precedence and Associativity of Some Operators**

Operators	*Associativity*
`++ -- + - (unary) (type)`	`right to left`
`* /`	`left to right`
`+ - (binary)`	`left to right`
`= += -= *= /=`	`right to left`

Operators in a higher row have higher precedence.

Note that unary `+`, `-` have higher precedence than the binary forms.

Every operator has a precedence and associativity. Here we listed only the operators taught up to now.

For example, consider a variable `i` of type `float`:

```
i = 3 * 4 / 5;
```

is evaluated as:

```
i = ( 3 * 4 ) / 5;
```

which gives `2` (because the numbers are evaluated as intergers), and

```
i = -5 * 3 - 2;
```

is evaluated as:

```
i = ( (-5) * 3 ) - 2;
```

which gives `-17`.