# Language Tutorial

## Hello World using LiTeC

```
print(“Hello World”);
```

## Program Structure

The program starts executing from the first line of code. The further execution of the code is sequential except when it encounters jump statements, loops, functions etc.

## Compile and Execute LiTeC Program

To compile and execute a LiTeC program, follow these steps:

```
yacc -d parser.y
lex lex.l
gcc symbol_table.c ast.c semantic.c lex.yy.c y.tab.c -o parser
```

## Data types in LiTeC

bool, int, double and char are the available data types in LiTeC.

## Operators in LiTeC

All the basic arithmetic operators and relational operators are supported by LiTeC.

| Operators | Symbol | Description |
|---|---|---|
| Arithmetic Operators | + | Addition |
| | - | Subtraction |
| | * | Multiplication |
| | / | Division |
| | % | Modulo |
| Relational Operators | = | Checks Equality |

|  |  |  |
|---|---|---|
|  | < | Checks Lesser Than |
|  | > | Checks Greater Than |
| Logical Operators | ^ | And Operator |
|  | \| | Or Operator |
|  | ~ | Not Operator |
| Assignment Operator | : | Is Assigned |
| Address of | & | Returns address |

## Semicolons and Blocks in LiTeC

In LiTeC, every statement must end with a semicolon that indicates the end of a logical entity.

For example,

```
a = (+ b c);
k = fact(a);
```

A block of statements represents a set of statements that are logically connected. Such a block of statements must be enclosed within curly braces.

For example,

```
{
        a = 2;
        b = 3;
        c = (+ a  b);
        print("%i", c);
}
```

## LiTeC Identifiers

An identifier in LiTeC is used to identify a variable, function, structures and subfiles. An identifier can be any combination of letters, digits and underscores but it must start with a letter only. The keyword ‘declare’ followers by the data type is used to declare an identifier.

For example,

```
declare int a : 2;
```

## Comments in LiTeC

Comments are written to understand what a particular statement or block of code does and the flow of the code. They are ignored by the compiler.

For example,

```
a = (+ b c); /* Addition of two numbers */
```

Enumerated comments help us to uniquely identify a comment and navigate to it easily.

For example,

```
#1 /* Multiplication of two numbers */
p = (* q r);
```

## Decision Making in LiTeC

The decision making statements in LiTeC are:

1. if

   ```
   if(condition)
   {
           statements;
   }
   ```

2. if - else

   ```
   if(condition)
   {
           statements;
   ```

```
}
else
{
	statements;
}
```

3. if - else if - else

```
if(condition)
{
	statements;
}
else if(condition)
{
	statements;
}
else
{
	statements;
}
```

## Loops in LiTeC

Since the for, while and do while loops can be written in terms of the other two, there is only one loop in LiTeC. The syntax is:

```
loop(initialization; test condition; incrementation)
{
	statements;
}
```

## Functions in LiTeC

A function in LiTeC acts as a map between a set of zero or more inputs and one output. The syntax for function declaration is:

```
declare  libraryName_functionName(argument1, argument2, …) ->
return_type
{
        statements;
}
```

## Data Structures in LiTeC

1. Arrays

   An array in LiTeC can store integers, floating point numbers and characters. The datatype of all elements of an array must be the same. To declare an array we need to follow these steps below:

   ```
   declare array int A[10];
   declare array char B[20];
   ```

   To initialize the elements in an array:

   ```
   loop(declare int i : 0; (< i 10); i : (+ i 1))
   {
           A[i] : i;
   }
   ```

2. Matrices

   A matrix is nothing but a two dimensional array. A matrix can be declared as follows:

   ```
   declare matrix int M[3][3];
   ```

   To fill a matrix with numbers:

   ```
   loop(declare int i : 0; (< i 10); i : (+ i 1))
   {
           loop(declare int i : 0; (< i 10); i : (+ i 1))
           {
                   M[i][j] = 1;
           }
   }
   ```

3. Struct

   A struct is used to group variables having different data types but those that are related to each other. The syntax to declare a struct is:

```
declare struct S[int][double];
S[1] : 1;
S[2] : 95;
```

## How to include TeX code?

To include TeX code in LiTeC, we have to write them within the TeX {} block. To include a variable, a statement, function or perform any operation within this block, we must encode them within [: :]

For example,

```
TeX
{
        “\use{packagename}
        \begin{document}

        \section{sectionname}”,
        [: texFunction(parameters list) :],
        “\end{document}”
}
```

## Working with Subfiles in LiTeC

The following example illustrates the way to work with subfiles in LiTeC:

```
{[
   MainFile
   ([
      declare int x : 10;
      declare char a : "b";
   ])
   int main()
   {
        declare int u : (+ x 10);
        declare int v : 15;
        return 0;
    }
]}
{[
    SubFile1
    ([
        declare int y : 20;
        declare char c : "x";
    ])

    int main()
    {
        declare int m : (/ x y);
        declare char b : a;
        return 0;
    }
]}
{[
    SubFile2
    ([
        declare int z : 30;
    ])
    int main()
    {
        declare int m : (% x (* y z));
        declare char b : c;
        c : a;
        return 0;
    }
]}
```