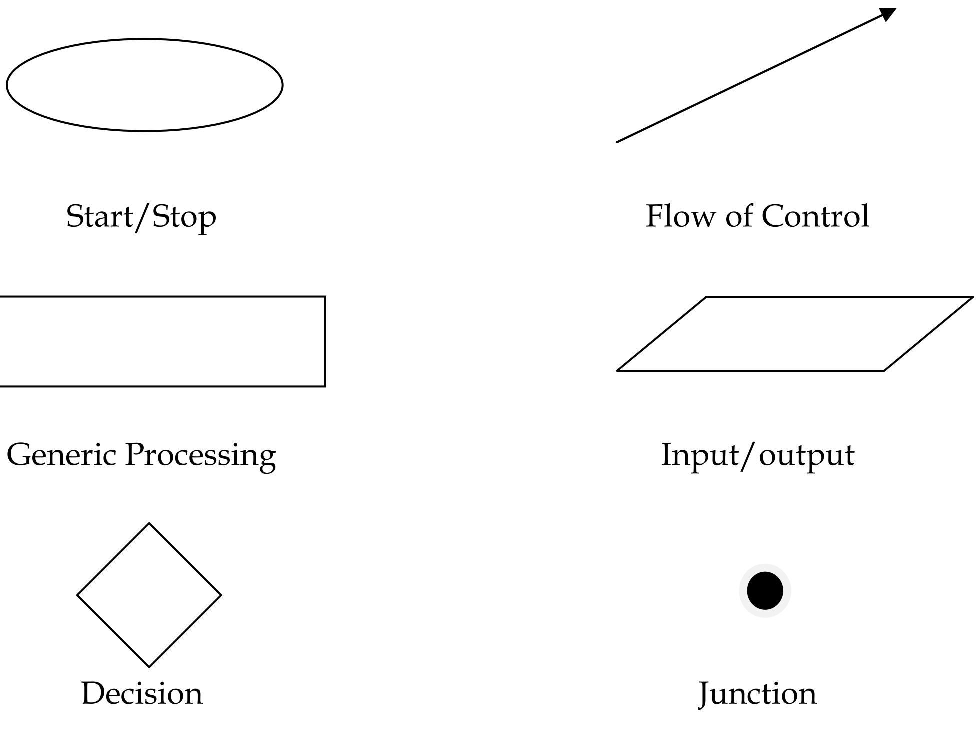
Algorithm: an algorithm is an ordered set of steps to be performed for accomplishing a task. In computers, an algorithm is a set of steps required to be performed to solve a computational problem. An algorithm is written independent of any programming language. For this purpose, pseudo code is used. For example, the below algorithm finds the largest number is a given list of numbers,

```
largest $\leftarrow$ L0
 for each item in the list (Length(L)$\geq$1), do
   if the item > largest, then
     largest $\leftarrow$ the item
 return largest
```

Flowchart: a flowchart is the pictorial representation of an algorithm. The advantage of using a pictorial representation is that it is "easy to understand". A flowchart can be any semantically correct combination of the following shapes,



The C Programming Language: the C language was written by Dennis Ritchie around the year 1970 on a PDP-7 system at AT&T Bell Laboratories. It is one of the earliest High Level Languages developed and still remains the most widely used. Some characteristics of the C language are,

1. A small set of KEYWORDS (the C language uses 32 keywords). A keyword is a word that has special significance for the compiler and its use elsewhere in the source code is restricted. For example. The keyword *int* is used for

declaration of integer variables. The word *int* cannot be used in a program elsewhere.

2. It supports a large number of Arithmetic (+, -, *, /) and Logical operations (AND, OR, NOT).
3. The C language can be used to write native code. This feature makes it the standard language for writing system level programs. Native code is code written for a particular machine and uses the system calls available with the hardware directly.
4. The C pre-processor is a very strong feature that allows programmers to use macros (#define), conditional compilation (#ifdef) and library inclusion (#include).
5. C supports functions which allow a programmer to structure a program better.

The C++ Programming Language: it is an enhancement to the C programming language. C++ includes object-oriented features and is sometimes referred to as C with Classes. C++ supports programming constructs like,

1. Virtual Functions
2. Friend Functions
3. Multiple Inheritance
4. Exception Handling

The Java Programming Language: it is a completely object-oriented programming language developed by the Sun Microsystems. The key ingredient to the success of Java is WORA (write once, run anywhere). What this means is that code written in Java once compiled can be run on any machine. This is made possible by including a JVM (Java Virtual Machine). The Java Compiler outputs Bytecode while is interpreted using the Java Interpreter and this interpreted code runs on the JVM in contrast to the other languages (compiled code runs directly on the machine).

Other important features of the Java Programming Language are,

1. Robust and Secure
2. Portable
3. Multithreaded

Data Types: for a programming language, a data type defines the kind of values a variable can store. Common data types available to nearly all modern programming languages are,

1. Integer data types (short, long, int)
2. Floating point numbers (float, double)

3. Characters (char)

In some programming languages like Java and C++, String and Boolean are also available. A programmer is not restricted by the data types available with the compiler and can use the available data types to create custom data types using programming constructs like Structures, Unions, etc.

A very important attribute of a data type is its SIZE. The size of a data type determines the range of values that can be stored in a variable of that type. For example, a 1-byte signed integer can store values ranging from -128 to 127 while a 2-byte signed integer can store values from -32768 to 32767. The size of a data type is not fixed and varies based on,

1. Machine Architecture
2. Compiler Implementation

However, for the Turbo C implementation the following sizes are defined,

1. Integer: 2 bytes
2. Characters: 1 byte
3. Floating Point Numbers of Single Precision: 4 bytes
4. Long Integers: 4 bytes

Assignment Statements: an assignment operation is used to set the value of a variable. For example, consider the statement given below,

```
X = 10;
```

In the above statements, the variable X is assigned the value 10.

To make an assignment, the "=" (equal to) operator is used. The LHS of the "=" operator should have the variable name and is called the *lvalue* of the assignment. Keep in mind that the *lvalue* cannot take literals like 50, 100, -20 or any other expression that does not correspond to a memory location capable of storing a value. The RHS can contain any expression that evaluates to a value consistent with the data type of the variable in the LHS.

In computer programming, it is a frequent practice to update a variable like shown below,

```
X = X + 10;
```

Here, the 10 is added to X and stored in the same variable. To allow for easy implementation of such operations, C provides a variant of the assignment operator "+=" and the above statement can be rewritten as, X+=10;

Similarly for subtraction, multiplication and division we have "-=", "*=" and "/=".

Conditional Statements: Based on a programmer specified condition (or a set of conditions), a conditional statement performs different computations. A condition is a expression which evaluates to either TRUE or FALSE. In other words, depending on whether a condition evaluates to TRUE or FALSE, a conditional statement performs different operations.

The conditions are evaluated during execution of the program. A program that uses conditional statements could behave differently for different combinations of condition variables.

In C, we have 2 types of constructs for this purpose,

1. If-Else Statements (Ladder)
2. Switch-Case Statements

If-Else works like shown below,

```
if(condition)
{
        //statements to execute if condition is true
}
else    //optional
{
        //statements to execute if condition is false
}
```

For evaluating whether a condition is true or false, different languages use different mechanisms. For languages that support Boolean data types, an expression that evaluates to a Boolean variable is used. In C, there is no Boolean data type so the evaluation is towards an integer. If the value of the condition expression is 0 then the condition is false, any non-zero value denotes true.

The switch-case construct is shown below,

```
switch(n) {
        case 1: operations when n=1
        case 2: operations when n=2
                        .
                        .
                        .
        case n: operations when n=n
        default: operations when n does not match any case value
}
```

The switch-case statement is specifically useful for implementing a multi-way branching at a single point. For example, in a menu driven program this functionality is required.

There is another operator in C called the Ternary Operator "?" and is implemented as follows,

```
(condition) ? (operation when true) : (operation when false)
```

Loops: a loop is used when it is desired that a particular piece of code be executed multiple times. For example, when processing multiple data points it is necessary that the same treatment is given to all data points and loops come in handy during such situations.

A loop has three stages,

1. Initialization – where the control variable(s) for the loop are set.
2. Evaluation – when the condition that controls the loop is checked to decide if the loop should continue of terminate.
3. Updation – when the control variable is updated for the next iteration of the loop.

In C we have 3 kinds of loops,

1. For Loop (generally used when the number of iterations is known or fixed. In other words, when the termination criteria involves a specific number of iterations, For Loop is generally used)
2. While Loop (this is generally used when there is a condition that must be true/false for the loop to stop)
3. Do-While Loop (Both While and For Loop check for the truth of the condition that control the loop at the beginning of the loop, Do-While does so at the end of the loop. This makes Do-While execute at least once even if the condition was false at the beginning itself)

For Loop, it is implemented as shown below

```
for(initialization; condition; updation) {
        //operations
}
```

A program that prints a table of 5 till 10 can be written as,

```
for(i=1; i<11; i++) {
        printf("%d X %d = %d", 5, i, 5*i);
```

```
    }
```

A while loop is implemented as,

```
    while(condition) {
            //operations
    }
```

For the same problem as above,

```
    i=0;
    while(i<11) {
            printf("%d X %d = %d", 5, i, 5*i);
            i++;
    }
```

The do-while loop is implemented similar to the while loop but the condition is placed at the end of the loop as shown below,

```
    do {
            //operations
    }while(condition);
```

Notice the semicolon at the end of the while statement.

Subprogram (Function or Subroutine): the C language allows us to structure a program such that it becomes easier to understand, maintain and modify. For this purpose, one of the features that C provides us is the concept of a Function or subprogram.

A programmer can declare and define functions in a program and whenever that functionality is required, just call the function that was previously written.

A generic function declaration is given below,

```
    return-type function-name (parameters) {
            //operations
    }
```

- The return type of the function denotes the kind of value a function returns.
- Parameters are a list of n inputs to the functions separated by a comma, each having its own data type.

For example, consider a program to add 2 numbers,

```
    void main() {
            int a=5, b=10;
```

```
        printf("the sum is : %d", a+b);
}
```

The same program can be written using functions,

```
int sum (int a, int b) {
        return a+b;
}

void main() {
        int a=5, b=10;
        printf("the sum is : %d", sum(a,b));
}
```

Using this approach, we can reduce the amount of time and effort required to maintain this program. In the example, we are using a very simple function that is used only once. In practice, a function could be very complex and used multiple times. Using the subprogram approach, we can ensure that any change in the processing requirements is addressed by making changes at one place. If subprogram approach is not used, we might have to make a lot of changes.

Coding Style: the programs written today could end up being maintained for a very long time to come. It is important that we follow a set of guidelines while writing our programs so that any person who is reading our program is able to understand what is being done. We primarily a three prong strategy to make our code understandable,

1. Choosing the right names for our variables, functions.
2. Commenting when required.
3. Proper indentation.

Consider the following declaration,

```
int a, b, c;
```

In the above declaration it is unclear what kind of values these variables will store except the fact that these are integers. In the contrary, consider the below given declaration,

```
int sum, input_1, input_2;
```

It is somewhat clear that one of the three variables is used for storing sum and two variables store some input but the exact nature of these variables is still not known. Consider the declaration given below,

```
int sum, input_1, input_2;     //sum stores the sum of input_1 and input_2
```

So adding a comment further enhances the understandability of our code.

When writing complex code indentation is very helpful in determining which part of code falls in what block. Consider the following example,

```
if (a==0)
{
printf("unindented code");
a=5;
}
```

In contrast consider the same code when indentation is given,

```
if (a==0)
{
        printf("indented code");
        a=5;
}
```

In the second snippet, the contents of the 'if' part are more clear. As the length of the program increases, indentation becomes more important.