Cyclomatic Complexity

Cyclomatic complexity of a code section is the quantitative measure of the number of linearly independent paths in it. It is a software metric used to indicate the complexity of a program. It is computed using the Control Flow Graph of the program. The nodes in the graph indicate the smallest group of commands of a program, and a directed edge in it connects the two nodes i.e. if second command might immediately follow the first command.

For example, if source code contains no control flow statement then its cyclomatic complexity will be 1 and source code contains a single path in it. Similarly, if the source code contains one **if condition** then cyclomatic complexity will be 2 because there will be two paths one for true and the other for false.

Mathematically, for a structured program, the directed graph inside control flow is the edge joining two basic blocks of the program as control may pass from first to second.  
So, cyclomatic complexity M would be defined as,

***M = E – N + 2P***

*where,  
E = the number of edges in the control flow graph  
N = the number of nodes in the control flow graph  
P = the number of connected components*

Steps that should be followed in calculating cyclomatic complexity and test cases design are:

* Construction of graph with nodes and edges from code.
* Identification of independent paths.
* Cyclomatic Complexity Calculation
* Design of Test Cases

Let a section of code as such:

A = 10

IF B > C THEN

A = B

ELSE

A = C

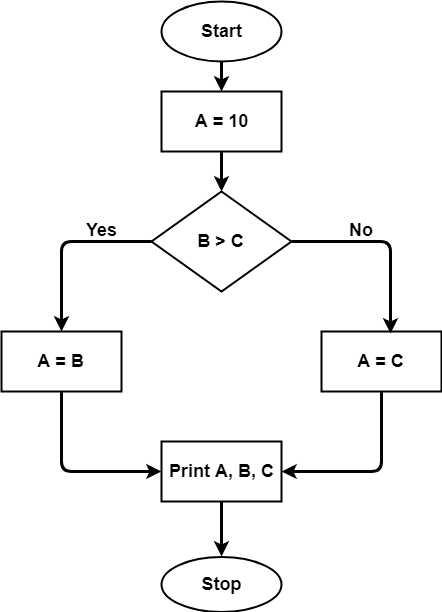
ENDIF

Print A

Print B

Print C

**Control Flow Graph** of above code



The cyclomatic complexity calculated for above code will be from control flow graph. The graph shows seven shapes(nodes), seven lines(edges), hence cyclomatic complexity is 7-7+2 = 2.

**Use of Cyclomatic Complexity:**

* Determining the independent path executions thus proven to be very helpful for Developers and Testers.
* It can make sure that every path have been tested at least once.
* Thus help to focus more on uncovered paths.
* Code coverage can be improved.
* Risk associated with program can be evaluated.
* These metrics being used earlier in the program helps in reducing the risks.

Software Engineering | Control Flow Graph (CFG)

A **Control Flow Graph (CFG)** is the graphical representation of control flow or [computation during the execution of programs](https://www.geeksforgeeks.org/cyclomatic-complexity/) or applications. Control flow graphs are mostly used in static analysis as well as compiler applications, as they can accurately represent the flow inside of a program unit. The control flow graph was originally developed by *Frances E. Allen*.

**Characteristics of Control Flow Graph:**

* Control flow graph is process oriented.
* Control flow graph shows all the paths that can be traversed during a program execution.
* Control flow graph is a directed graph.
* Edges in CFG portray control flow paths and the nodes in CFG portray basic blocks.

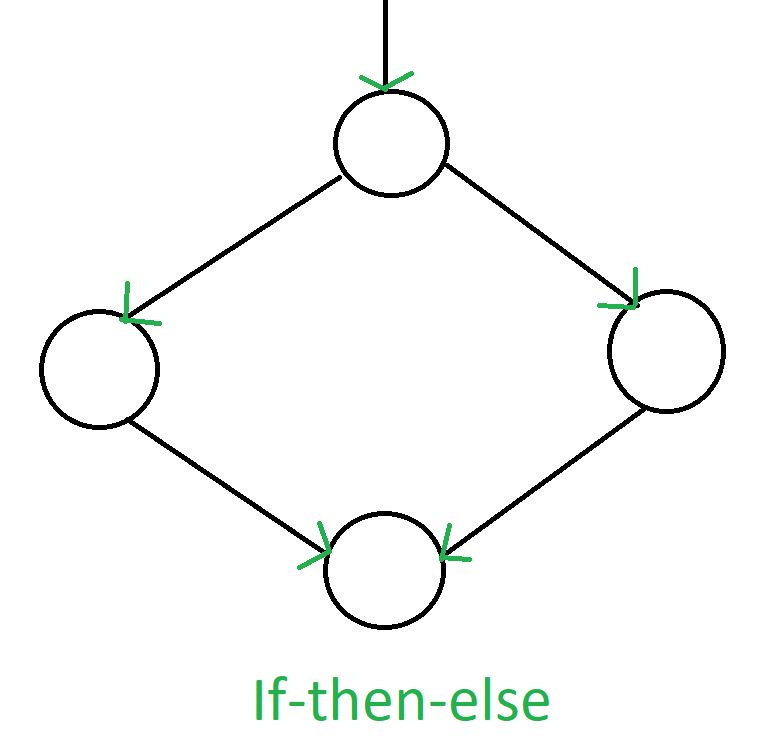
There exist 2 designated blocks in Control Flow Graph:

1. **Entry Block:**  
   Entry block allows the control to enter into the control flow graph.
2. **Exit Block:**  
   Control flow leaves through the exit block.

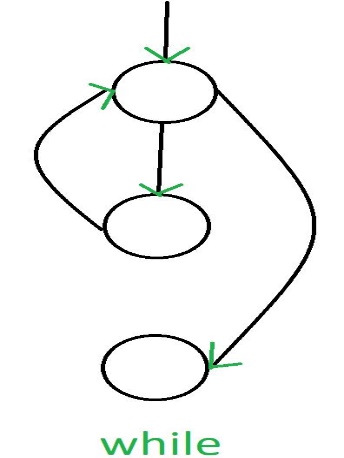
Hence, the control flow graph is comprised of all the building blocks involved in a flow diagram such as the start node, end node and flows between the nodes.

**General Control Flow Graphs:**  
Control Flow Graph is represented differently for all statements and loops. Following images describe it:

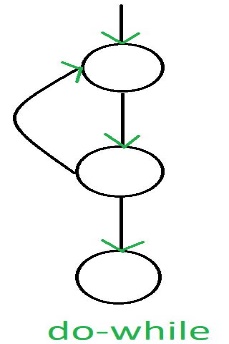
1. **If-else:**



2. **while:**



3. **do-while:**



4. **for:**

