**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:

* **Entry Block:**  
  Entry block allows the control to enter into the control flow graph.
* **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:**



**Example:**

if A = 10 then

if B > C

A = B

else A = C

endif

endif

print A, B, C

Flowchart of above example will be:



Control Flow Graph of above example will be:



**Advantage of CFG:**  
There are many advantages of a control flow graph. It can easily encapsulate the information per each basic block. It can easily locate inaccessible codes of a program and syntactic structures such as loops are easy to find in a control flow graph.

Unreachable code

[Jump to HYPERLINK "https://en.wikipedia.org/wiki/Unreachable\_code"navigation HYPERLINK "https://en.wikipedia.org/wiki/Unreachable\_code"Jump HYPERLINK "https://en.wikipedia.org/wiki/Unreachable\_code" to search](https://en.wikipedia.org/wiki/Unreachable_code)

In [computer programming](https://en.wikipedia.org/wiki/Computer_programming), **unreachable code** is part of the [source code](https://en.wikipedia.org/wiki/Source_code) of a program which can never be executed because there exists no [control flow](https://en.wikipedia.org/wiki/Control_flow) path to the code from the rest of the program.[[1]](https://en.wikipedia.org/wiki/Unreachable_code)

Unreachable code is sometimes also called *dead code*,[[2] HYPERLINK "https://en.wikipedia.org/wiki/Unreachable\_code"[3]](https://en.wikipedia.org/wiki/Unreachable_code) although [dead code](https://en.wikipedia.org/wiki/Dead_code) may also refer to code that is executed but has no effect on the output of a program.[[4]](https://en.wikipedia.org/wiki/Unreachable_code)

Unreachable code is generally considered undesirable for several reasons:

* It uses memory unnecessarily
* It can cause unnecessary use of the CPU's [instruction cache](https://en.wikipedia.org/wiki/Instruction_cache)
* This can also decrease [data locality](https://en.wikipedia.org/wiki/Data_locality)
* Time and effort may be spent testing, maintaining and documenting code which is never used
* Sometimes an automated test is the only thing using the code.

However, unreachable code can have some legitimate uses, like providing a library of functions for calling or jumping to manually via a [debugger](https://en.wikipedia.org/wiki/Debugger) while the program is halted after a [breakpoint](https://en.wikipedia.org/wiki/Breakpoint).

Unreachable code can exist for many reasons, such as:

* programming errors in complex conditional branches
* a consequence of the internal transformations performed by an [optimizing compiler](https://en.wikipedia.org/wiki/Optimizing_compiler);
* incomplete testing of new or modified code
* Legacy code
* Code superseded by another implementation
* Unreachable code that a programmer decided not to delete because it is mingled with reachable code
* Potentially reachable code that current use cases never need
* Dormant code that is kept intentionally in case it is needed later
* Code used only for debugging.

Legacy code is that which was once useful but is no longer used or required. But unreachable code may also be part of a complex library, module or routine where it is useful to others or under conditions which are not met in a particular scenario.

An example of such a conditionally unreachable code may be the implementation of a general string formatting function in a compiler's runtime library, which contains complex code to process all possible arguments, of which only a small subset is actually used. Compilers will typically not be able to remove the unused code sections at compile time, as the behavior is largely determined by the values of arguments at run time.

Examples

In this fragment of C code:

int foo (int X, int Y)

{

return X + Y;

int Z = X \* Y;

}

Another example

int N = 2 + 1;

if (N == 4)

{

/\* unreachable \*/

}

Another Example

goto fail bug

Apple's SSL/TLS from February 2014 contained a major security flaw known formally as CVE-2014-1266 and informally as the "goto fail bug".[5][6] The relevant code fragment[7] is:

static OSStatus

SSLVerifySignedServerKeyExchange(SSLContext \*ctx, bool isRsa, SSLBuffer signedParams,

uint8\_t \*signature, UInt16 signatureLen)

{

OSStatus err;

...

if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)

goto fail;

if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)

goto fail;

goto fail;

if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)

goto fail;

fail:

SSLFreeBuffer(&signedHashes);

SSLFreeBuffer(&hashCtx);

return err;

}

Here, there are two successive calls to goto fail. In the syntax of the C language, the second is unconditional, and hence always skips the call to SSLHashSHA1.final. As a consequence, err will hold the status of the SHA1 update operation, and signature verification will never fail.[5]

Here, the unreachable code is the call to the final function. There are several coding practices that could have prevented this fault, such as code review, the proper use of indentation or block structure, and test coverage analysis.[6] Applying the Clang compiler with the option -Weverything includes unreachable code analysis, which would trigger an alarm for this code.[6]