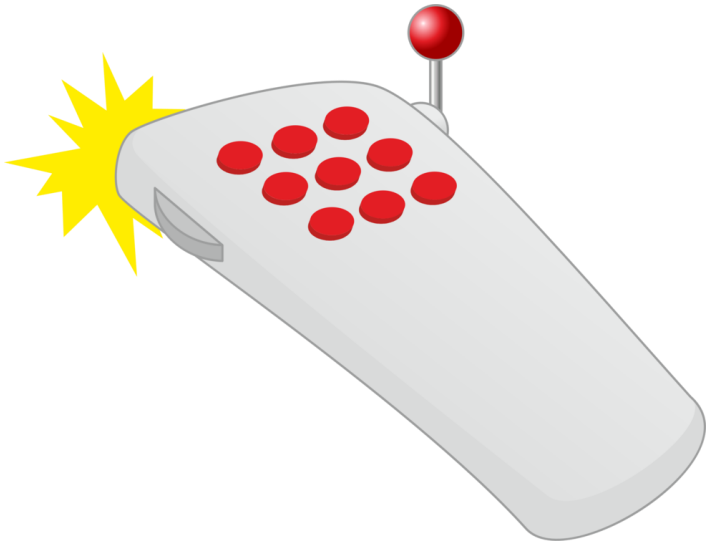
Software Maintenance Measurement

You can't control what you can't measure

➢-Tom DeMarco



Measuring Maintainability

- Source code is the most commonly used way of measuring maintainability
  - Readily available
  - Easy to collect and automate

Size

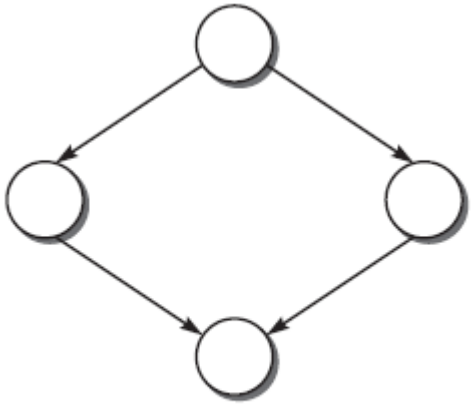
- **Lines Of Code (LOC):** the count of program lines of code excluding comment or blank lines
- During maintenance, the focus is on the ‘delta’ lines of code (number of lines of code that have been added or modified during a maintenance process)
- **Advantage:**
  - easy to calculate
- **Disadvantage:**
  - dependent on the programming language in question
  - does not reflect cost or productivity

McCabe's Cyclomatic Complexity

- A quantitative measure of the number of linearly independent paths through a program's source code
  - An independent path is any path through the graph that introduces at least one new set of processing statements or new conditions. An independent path must move along at least one edge that has not been traversed before the path is defined
- Cyclomatic complexity (CC) = E - N + 2P
  - E = number of edges
  - N = number of nodes
  - P = number of components

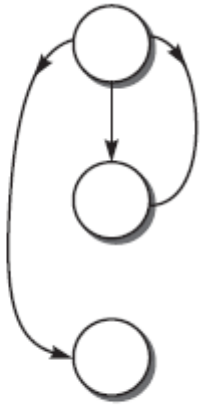
Flow Graph Notations

(a) Sequence

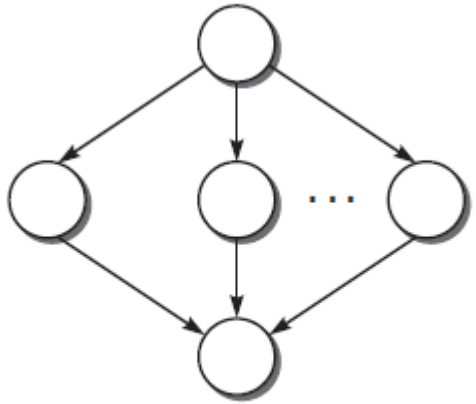


(b) If-Then-Else

(c) Do-While



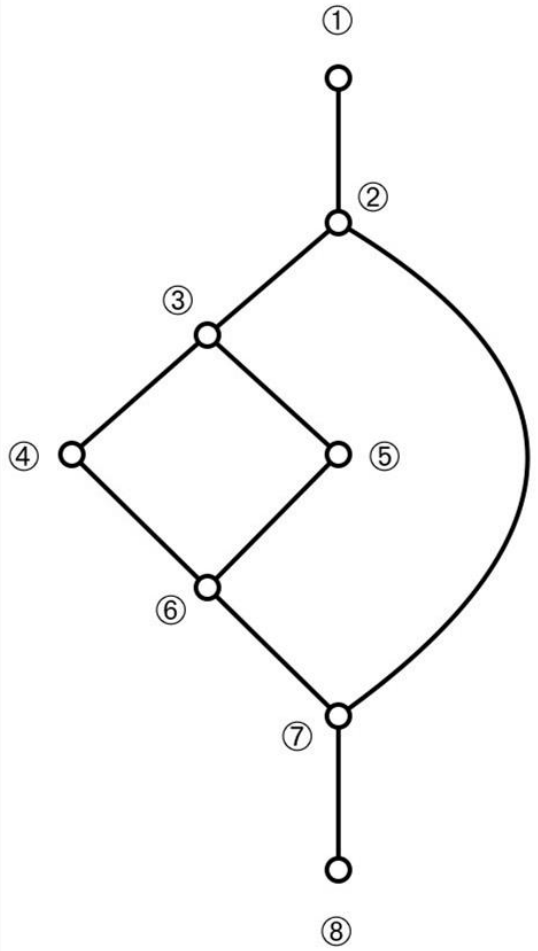
(d) While-Do



(e) Switch-Case

McCabe's Cyclomatic Complexity

```
① start
② if (X) then
③     if (Y) then
④         perform A
          perform B
      else
⑤         perform C
          perform D
⑥     endif
⑦ endif
⑧ end
```



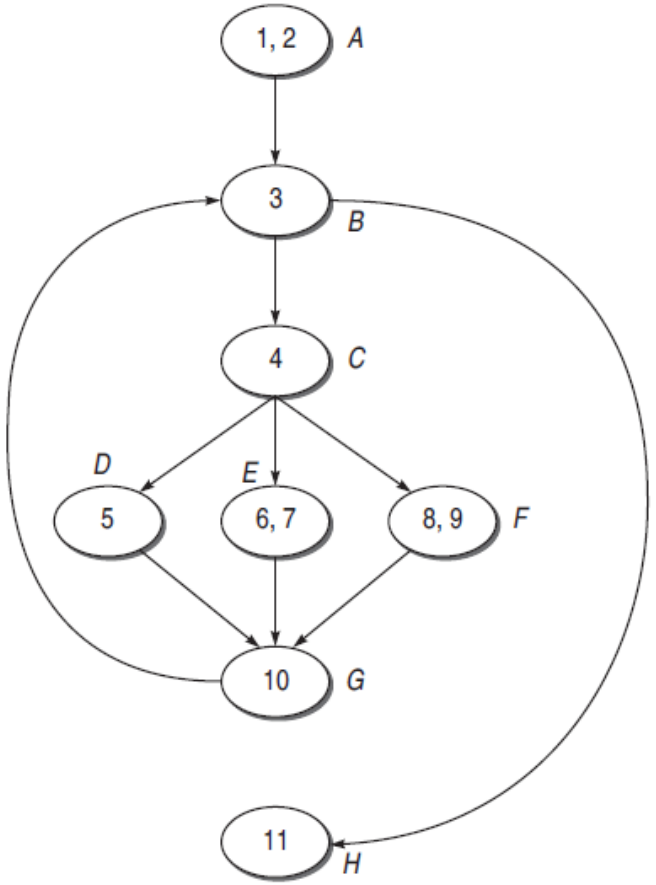
- number of nodes = 8
- number of edges = 9
- cyclomatic number = 9-8+(2*1) = 3

Practice Problem

```
main()
{
    char string [80];
    int index;
1.  printf("Enter the string for checking its characters");
2.  scanf("%s", string);
3.  for(index = 0; string[index] != '\0'; ++index)   {
4.      if((string[index] >= '0' && (string[index] <='9'
5.                  printf("%c is a digit", string[index]);
6.      else if ((string[index] >= 'A' && string[index] <'Z'))  ||
                ((string[index] >= 'a' && (string[index] <'z')))
7.                  printf("%c is an alphabet", string[index]);
8.      else
9.                  printf("%c is a special character", string[index]);
10.     }
11. }
```

Solution

```
    main()
    {
        char string [80];
        int index;
1.      printf("Enter the string for checking its characters");
2.      scanf("%s", string);
3.      for(index = 0; string[index] != '\0'; ++index)   {
4.          if((string[index] >= '0' && (string[index] <='9'
5.                      printf("%c is a digit", string[index]);
6.          else if ((string[index] >= 'A' && string[index] <'Z'))  ||
                    ((string[index] >= 'a' && (string[index] <'z')))
7.                  printf("%c is an alphabet", string[index]);
8.          else
9.                  printf("%c is a special character", string[index]);
10.         }
11. }
```



Cyclomatic complexity = e – n + 2 * P
= 10 – 8 + 2 = 4

McCabe's Cyclomatic Complexity

- **Advantage:**
  - helps to identify highly complex programs that may need to be modified in order to reduce complexity
  - can be used as an estimate of the amount of time required to understand and modify a program
- **Disadvantage:**
  - takes no account of the complexity of the conditions in a program, e.g., multiple use of Boolean expressions, and over-use of flags

Halstead Metrics

- A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operand
  - An operand is a variable or constant
  - An operator is an entity that can either change the value of an operand or the order in which it is changed
- The basic measures are
  - n1 = count of unique operators
  - n2 = count of unique operands
  - N1 = count of total occurrences of operators
  - N2 = count of total occurrence of operands

Counting Rules for C language

- Comments are not considered
- The identifier and function declarations are not considered
- All the variables and constants are considered operands
- Global variables used in different modules of the same program are counted as multiple occurrences of the same variable
- Local variables with the same name in different functions are counted as unique operands
- Functions calls are considered as operators
- All looping statements e.g., do {...} while ( ), while ( ) {...}, for ( ) {...}, all control statements e.g., if ( ) {...}, if ( ) {...} else {...}, etc. are considered as operators
- In control construct switch ( ) {case:...}, switch as well as all the case statements are considered as operators

Counting Rules for C language

- The reserve words like return, default, continue, break, sizeof, etc., are considered as operators
- All the brackets, commas, and terminators are considered as operators
- GOTO is counted as an operator and the label is counted as an operand
- The unary and binary occurrence of "+" and "-" are dealt with separately
- In the array variables such as "array-name [index]" "array-name" and "index" are considered as operands and [ ] is considered an operator
- In the structure variables such as "struct-name, member-name" or "struct-name -> member-name," struct-name, member-name are considered as operands and '.', '->' are taken as operators. Some names of member elements in different structure variables are counted as unique operands
- All the hash directive is ignored

Halstead Metrics

- **Halstead Program Length:** total number of operator occurrences and the total number of operand occurrences

$$N = N1 + N2$$

- **Halstead Vocabulary:** total number of unique operator and unique operand occurrences

$$n = n1 + n2$$

- **Program Volume:** represents the size necessary for storing the program

$$V = Size * (\log_2 vocabulary) = N * \log_2(n)$$

Halstead Metrics

- **Program Difficulty:** how difficult to handle the program is

$$D = (n1 / 2) * (N2 / n2)$$

- **Programming Effort:** Measures the amount of mental activity needed to translate the existing algorithm into implementation in the specified program language

$$E = \text{Difficulty (D)} * \text{Volume (V)}$$

Halstead Metrics

```
main()
{
   int a, b, c, avg;
   scanf("%d %d %d", &a, &b, &c);
   avg = (a+b+c)/3;
   printf("avg = %d", avg);
}
```

n1 = 12, n2 = 7, n = 19

N1 = 27, N2= 15, N = 42

V = 42* $\log_2$ 19 = 178.4

D = (12/2) * (15/7) = 12.85

E = 12.85* 178.4 = 2292.44

Halstead Metrics

- **Advantage:**
  - easy to calculate
  - can be used as good predictors of programming effort and number of bugs in a program
- **Disadvantage:**
  - ignores the high-level structures (or chunks) that expert programmers use to understand programs

Guidelines for Selecting Maintenance Measures

- **Clearly defined objectives:** Prior to deciding on the use of a measurement for maintenance-related purposes, it is essential to define clearly and unambiguously what objectives need to be achieved. These objectives will determine the measures to be used and the data to be collected.
- **Personnel involvement:** The purpose of measurement in an organization needs to be made clear to those involved in the programme. And the measures obtained should be used for that purpose and nothing else. For example, it needs to be made clear whether the measurement is to improve productivity, to set and monitor targets, etc.
- **Ease of use:** The measures that are finally selected to be used need to be easy to use, should not take too much time to administer, and possibly subject to automation