**Halstead’s Software 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 operands. **Halstead’s metrics** are included in a number of current commercial tools that count software lines of code. By counting the tokens and determining which are operators and which are operands, the following base measures can be collected :

n1 = Number of distinct operators.  
n2 = Number of distinct operands.  
N1 = Total number of occurrences of operators.  
N2 = Total number of occurrences of operands.

**Token Count**

The size of the vocabulary of a program, which consists of the number of unique tokens used to build a program is defined as:

n = n1+n2

**Program Length**

N=N1+N2

The length of the program in terms of total number of tokens used is N.

### Counting rules for C language –

1. Comments are not considered.
2. The identifier and function declarations are not considered
3. All the variables and constants are considered operands.
4. Global variables used in different modules of the same program are counted as multiple occurrences of the same variable.
5. Local variables with the same name in different functions are counted as unique operands.
6. Function calls are considered as operators.
7. All looping statements e.g., do {…} while ( ), while ( ) {…}, for ( ) {…}, all control statements e.g., if ( ) {…}, if ( ) {…} else {…}, etc. are considered as operators.
8. In control construct switch ( ) {case:…}, switch as well as all the case statements are considered as operators.
9. The reserved words like return, default, continue, break, sizeof, etc., are considered as operators.
10. All the brackets, commas, and terminators are considered as operators.
11. GOTO is counted as an operator and the label is counted as an operand.
12. The unary and binary occurrence of “+” and “-” are dealt separately. Similarly “\*” (multiplication operator) are dealt separately.
13. In the array variables such as “array-name [index]” “array-name” and “index” are considered as operands and [ ] is considered as operator.
14. In the structure variables such as “struct-name.member-name” or “struct-name -> member-name”, struct-name and member-name are taken as operands and ‘.’, ‘->’ are taken as operators. Same names of member elements in different structure variables are counted as unique operands.
15. All the hash directives are ignored.

**Example –** List out the operators and operands and also calculate the values of software science measures

int sort (int x[ ], int n)

{

int i, j, save, im1;

/\*This function sorts array x in ascending order \*/

If (n< 2) return 1;

for (i=2; i< =n; i++)

{

im1=i-1;

for (j=1; j< =im1; j++)

if (x[i] < x[j])

{

Save = x[i];

x[i] = x[j];

x[j] = save;

}

}

return 0;

}

**Explanation –**

|  |  |  |  |
| --- | --- | --- | --- |
| **operators** | **occurrences** | **operands** | **occurrences** |
| int | 4 | sort | 1 |
| () | 5 | x | 7 |
| , | 4 | n | 3 |
| [] | 7 | i | 8 |
| if | 2 | j | 7 |
| < | 2 | save | 3 |
| ; | 11 | im1 | 3 |
| for | 2 | 2 | 2 |
| = | 6 | 1 | 3 |
| – | 1 | 0 | 1 |
| <= | 2 | – | – |
| ++ | 2 | – | – |
| return | 2 | – | – |
| {} | 3 | – | – |
| n1=14 | N1=53 | n2=10 | N2=38 |

Therefore,

N = 91

n = 24

**Data Structure Metrics**

**The Amount of Data**

One method for determining the amount of data is to count the number of entries in the cross-reference list.

A variable is a string of alphanumeric characters that is defined by a developer and that is used to represent some value during either compilation or execution.

1. Prepare a cross reference table of the program. A cross reference table consists of having a column listing out the names of all the variables in a program. In each row, the line numbers in which the variable at that row is found, are listed out.

Consider the following program to find the factorial of a number:

1. #include <stdio.h>
2. int main()
3. {
4. int n, i;
5. unsigned long factorial = 1;
6. printf("Enter an integer: ");
7. scanf("%d",&n);
8. // show error if the user enters a negative integer
9. if (n < 0)
10. printf("Error! Factorial of a negative number doesn't exist.");
11. else
12. {
13. for(i=1; i<=n; ++i)
14. {
15. factorial \*= i; // factorial = factorial\*i;
16. }
17. printf("Factorial of %d = %f", n, factorial);
18. }
19. return 0;
20. }

Cross reference table:

|  |  |
| --- | --- |
| Variable | Line numbers |
| n | 4, 8, 11, 16, 20 |
| i | 4, 16, 16, 16, 18 |
| factorial | 5, 18, 20 |

2. Identify live variables and prepare a table showing the count of each variable at each executable line in the program code. A variable is live from its first to its last references within a procedure.

|  |  |  |
| --- | --- | --- |
| Line number | Live variable | Count |
| 7 |  | 0 |
| 8 | n | 1 |
| 9 | n | 1 |
| 10 | n | 1 |
| 11 | n | 1 |
| 12 | n | 1 |
| 13 | n | 1 |
| 14 | n | 1 |
| 15 | n | 1 |
| 16 | n, i | 2 |
| 17 | n, i | 2 |
| 18 | n, i, factorial | 3 |
| 19 | n, factorial | 2 |
| 20 | n, factorial | 2 |
| 21 |  | 0 |
| 22 |  | 0 |
| 23 |  | 0 |
|  |  | 19 |

3. Calculate the total count of live variables from step 2 of the counts of live variables on each line of code in the program.

Total count of live variables = 19

4. Count the number of executable lines of code in the program. Exclude the # directives, function names, starting brackets, ending brackets of the program and declarations from a C program.

Count of number of executable lines = 17

It is thus possible to define the average number of live variables, which is the sum of the count of live variables divided by the count of executable statements in a program. This is a complexity measure for data usage in a procedure or program.

LV = 19/17.

**Sharing of data among modules**

Every system can be considered to include a set of modules which when combined together form a complete system. Upon combining these modules, it is found that the modules need to interact with each other. During interaction, modules usually pass data or control information to other modules through messages. Each message may carry a certain amount of data, which depends on the degree of coupling between modules. Thus modules become dependent on each other as a result.

Also, the ripple effect is evident in this case. The ripple effect is an effect similar to still water in a pond. IF you throw a stone into this pond, it will cause ripples throughout the surface of the pond. Similarly, changes in a software cause ripples across the entire software system.

Another aspect is that there are two types of variables in a system. These are local and global variables. Local variables are alive in a single module or unit. Thus, these are usually shared only as parameters in a function call. On the other hand, global variables are available throughout the system and can be modified by any module. Thus having global variables in a system causes problems in maintaining and changing the system if required.

Thus, it is required that this dependency between modules may be measured using some metrics. These are information flow metrics.

**Information Flow Metrics**

Every software system has components. A component may be defined as any element identified by decomposing a software system into its constituent parts. Since components in a system have been created out of the system therefore they are related to each other and interact with each other. Thus, when a component calls other components, it is said to be coupled with other components. Also, each component has to perform one or more functions and a component is said to be cohesive in this context. Thus if a component performs only one function, it is said to be highly cohesive.

Thus, to measure degree of coupling there are metrics called information flow metrics. These metrics model the degree of cohesion and coupling for a particular system component.

**The Basic Information Flow Model**

Information Flow metrics are applied to the Components of a system design. For a component ‘A’ we can define three measures,

1. ‘FAN IN’ is simply a count of the number of other Components that can call, or pass control, to Component A.

2. ‘FANOUT’ is the number of Components that are called by Component A.

3. This is derived from the first two by using the following formula. We will call this measure the INFORMATION FLOW index of a Component A, abbreviated as IF(A).

IF(A) = [FAN IN(A) x FAN OUT (A)]2

**Q1. What do you understand by token count? Consider a program having**

* **Number of distinct operator : 12**
* **Number of operands : 5**
* **Total number of operator occurrences : 20**
* **Total number of operand occurrences : 15**

**Calculate the Halstead software metrics, namely, token count and program length for the above programs.**

Q2. Consider the following program:

#include<stdio.h>

void main()

{

int a, b, sum, product;

sum=a+b;

product=a\*b;

printf(“The sum is %d.”, sum);

printf(“The product is %d.”, product);

}

Calculate the data structure metrics for the program.