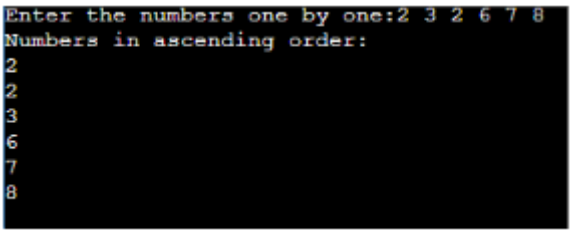
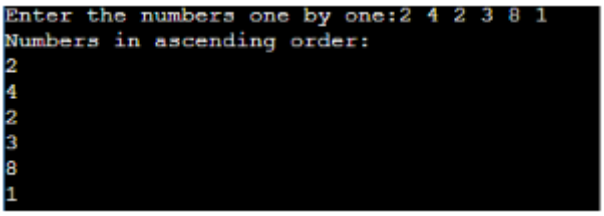
Task

The two C implementations that follow perform the same function: They first ask the user for the number of numbers he would like to sort. Then they take as input the numbers and display them sorted. Embodiment A displays the numbers sorted in ascending order while embodiment B displays the numbers sorted in descending order. The detailed calculation of the KLM time is calculated below:

An example of an input for embodiment A is shown in Figure 1:



An example of an input for embodiment B is shown in Figure 2:



1. For each of the above three routines, construct a detailed table showing the various operators and entailments, and the number of occurrences. This table should have the following format.

Operator	Number of appearances	Operand	Number of appearances
n1 =	N1	n2	N2=

2. Calculate for each of the three routines:

- The ratio of the length estimator to Halstead's program length (Nest/N),

- Halstead's program level (L),

- Halstead's language level (λ),

- the ratio of the number of lines of comments to the number of physical lines of code (Lines of Comments/Physical Lines of Code).

3. For the code of embodiment A, calculate a total value for each of the above metrics by considering the following scenarios: S1. the total values in the metrics are calculated by averaging their values in each routine

S2. The total values in the metrics are calculated from the weighted average of their values in each routine, based on N

Which scenario do you consider more appropriate? Document your answer, considering the more general case of applying these metrics to code that includes many routines.

4. Based on your chosen scenario, compare the two implementations A and B and comment on the result by interpreting the values of the metrics.

A' Implementation

```
/*
* C program to accept numbers as an input from user
* and to sort them in ascending order.
*/
#include <stdio.h>
/* Fuction for getting sorting number in ascending order*/
void sort_numbers_ascending(int number[], int count)
{
int temp, i, j, k;
for (j = 0; j < count; ++j)
{
for (k = j + 1; k < count; ++k)
{
if (number[j] > number[k])
{
temp = number[j];
number[j] = number[k];
number[k] = temp;
}
}
}
printf("Numbers in ascending order:\n");
for (i = 0; i < count; ++i)
printf("%d\n", number[i]);
}
void main()
{
int i, count, number[20], t=0;
printf("How many numbers you are going to enter:");
scanf("%d", &count);
printf("\nEnter the numbers one by one:");
while (t>20)
{
printf("\nThis is a test");
scanf("%d", &count);
}
for (i = 0; i < count; ++i)
scanf("%d", &number[i]);
/* Calling the Function*/
sort_numbers_ascending(number, count);
}
```

B' Implementation

```
/*
*
*
My code
*
*
*/
#include <stdio.h>
void main()
{
int i, num[20], t=0;
int n, count, j, a, x, b;
printf("How many numbers you are going to enter:");
scanf("%d", &count);
printf("\nEnter the numbers one by one:");
/*
*
*
*
Test this code
*
*
*
*/
while (t>20)
{
/*test*/
printf("\nThis is a test");
scanf("%d", &count);
printf("\nThis is my test");
scanf("%d", &count);
}
for(t=20; t<20; t--)
{
scanf("%d", &count);
}
/*My loop begins*/
for (i = 0; i < count; ++i)
scanf("%d", &num[i]);
for (i = 0; i < n; ++i){
for (j = i + 1; j < n; ++j){
if (num[i] > num[j]){
a = num[i];
num[i] = num[j];
num[j] = a;
}
}
}
/*Here are the data*/
printf("Numbers in ascending order:\n");
for (i = 0; i < count; ++i)
printf("%d\n", num[i]);
}
```

Solution

1.

A' Implementation

void sort_numbers_ascending()

Operator	Number of Appearances	Operands	Number of Appearances
int	3	temp	3
;	6	i	5
for(;;)	3	j	8
=	6	k	7
<	3	0	2
++	3	count	4
+	1	"Numbers in ascending order:\n"	1
if()	1	"%d\n"	1
number[]	8		
>	1		
printf()	2		
,	5		
sort_numbers_ ascending()	1		
{}	4		
n1=14	N1=47	n2=8	N2=31

void main()

Operator	Number of Appearances	Operands	Number of Appearances
int	1	i	5
;	8	0	2
for(;;)	1	count	5
=	2	"How many numbers you are going to enter:"	1
<	1	t	2
++	1	"%d"	3
number[]	2	"\nEnter the numbers one by one"	1
>	1	20	2
printf()	3	"\nThis is a test"	1
main()	1	number	1
scanf()	3		
&	3		
,	7		
sort_numbers_ ascending()	1		
while()	1		
{}	2		
n1=16	N1=38	n2=10	N2=23

I counted the operator & separately, since it is possible to have a scanf without it.

B' Implementation

Operator	Number of Appearances	Operands	Number of Appearances
main()	1	i	16
int	2		
num[]	9	t	5
,	13	0	4
=	9	n	3
;	16	count	7
printf()	6	j	7
scanf()	5	a	3
&	5	x	1
while()	1	b	1
>	2	"How many numbers you are going to enter:"	1
for(;;)	5	"%d"	5
<	5	"\nEnter the numbers one by one:"	1
--	1	20	4
++	4	1	1
+	1	"Numbers in ascending order:\n"	1
{}	6	"%d\n"	1
n1=17	N1=91	n2=17	N2=61

2.

Nest = n1log2n1 + n2log2n2 Nest1/N1 = 0,991

N1 = 47 + 31 = 78

N2 = 38 + 23 =61

N3 = 91 + 61 = 152 Nest3/N3 = 0,914

Nest1 = 53,298 + 24 = 77,298

Nest2 = 64 + 33,22 = 97,22

Nest3 = 69,4875 + 69,4875 = 138,975 Nest2/N2 = 1,593

V1 = N1log2n1 = 347,802

V2 = N2log2n2 = 286,7

V3 = N3log2n3 = 773,224

V* = (2+n2)log2(2+n2)

V1* = 33,22

V2* = 43,02

V3* = 80,712

L1 = V1*/V1 = 0,095

L2 V2*/V2 = 0,15

L3 = V3*/V3 = 0,104

λ = LV*

λ1 = L1V1* = 3,15

λ2 = L2V2* = 6,45

λ3 = L3V3* = 8,39

Note: For the first implementation, I calculated the first 7 lines of code in both routines

P-LOC1 = 27 LOC1 = 5 LOC1/P-LOC1 = 0,1851

P–LOC2 = 25 LOC2 = 6 LOC2/P-LOC2 = 0,24

P-LOC3 = 56 LOC3 = 19 LOC3/P-LOC3 = 0,3392

3.

Σ1.

n1 = 15 (14 of the 1st routine, 16 of the 2nd, so MO is 14)

n2 = 9 (8 and 10)

Nest = 58,605 + 28,53 = 87,135

N1 = 42,5 (47 and 38)

N2 = 27 (31 and 23)

N = 69,5

Nest/N = 1,25

n = 24

V = 318,65

V* = 28,53

L = 0,089

λ = 2,53

P-LOC = 26 (25 and 27)

LOC = 5,5 (5 and 6)

LOC/P-LOC = 0,211

Σ2.

The formula for the weighted average is as follows:

$$W = \frac{\sum_{i=1}^{n} w_i X_i}{\sum_{i=1}^{n} w_i}$$

n1=14,9 (I took n1=14, N1=47, n1=16, N1=38)

n2=8,9 (I took n1=8, N1=31, n1=10, N1=23)

Nest=58,065 + 28,07 = 86,135

n=23,8

N1=43

N2=27,6

N=70,6

Nest/N=1,22

V=322,85

V*=28,07

L=0,086

λ=2,41

P-LOC=26,1 (P-LOC1=27, P-LOC2=25, N1oλ=78, N2oλ=61)

LOC=5,4 (LOC1=5, LOC2=6)

LOC/P-LOC=0,206

Conclusion: It is difficult to choose which scenario is more appropriate because the values of the metrics are very close to each other. The Nest/N ratio is greater than unity in both cases due to the small module size, but in S2 it is closer to unity, which is the

ideal. The ideal program level should be L=1, which means we have a high level. Here we observe that in S1 it is closer to 1.

So, I will calculate the implementation effort for both cases, to choose the most appropriate scenario

Program Difficulty: D=1/L

D1=1/0,089=11,235

D2=1/0,086=11,627

E1=D1V1=3.580,03

E2=D2V2=3.753,77

Therefore, the best scenario is considered to be C1, because it is easier to implement and requires less effort.

4.

Nest=138,975

N=152

Nest/N= 0,914

V = 773,224

L = 0,104

V* = 80,712

λ = 8,39

D=9,615

E=7.434,54

We note that the first implementation is better. It may be that for embodiment B the Nest/N ratio is closer to unity and the program level L is larger and closer to unity, but it has a huge volume and this makes the implementation effort almost twice as much as embodiment A.