CS 3310 Design and Analysis of Algorithms
Project #1
Due: 1/11
(Total: 100 points)

Student Name: Loc Nguyen

Date: 04/06/2023

Important:
- *Please read this document completely before you start coding.*
- Also, please read the submission instructions (provided at the end of the project description) carefully before submitting the project.

Project #1 Description:

Program the following algorithms that we covered in the class:

a. Classical matrix multiplication
b. Divide-and-conquer matrix multiplication
c. Strassen's matrix multiplication

You can use either Java, C++ for the implementation. The objective of this project is to help student understand how above three algorithms operates and their difference in run-time complexity (average-case scenario). The project will be divided into three phases to help you to accomplish above tasks. They are Part 1: Design and Theoretical analysis, Part 2: Implementation, and Part 3: Comparative Analysis.

This project will ask student to conduct the matrix multiplication on the numeric data type. You can implement the above three algorithms in your choice of data structures based on the program language of your choice. Note that you always try your best to give the most efficient program for each problem. Let the matrix size be n x n. Carry out a complete test of your algorithms with n = 2, 4, 8, 16, 32, 64, 128, 256, … (up to the largest size of n that your computer can handle). The size of the input will be limited in the power of 2. They are 2, 4, 8, 16, 32 up to 2^k where 2^k is the largest size of the input that your computer can handle or your program might quit at size of 2^k due to running out of memory. The reason for the power of 2 is to ease the programming component of the assignment.

Submission Instructions:

After the completion of all three parts, Part 1, Part 2 and Part 3, submit (upload) the following files to Canvas:

- ***three program files of three algorithms or one program file including all three algorithms (your choice of programming language with proper documentation)***
- ***this document in pdf format (complete it with all the*** <Insert > ***answers)***

Part 1: Design & Theoretical Analysis (30 points)

a. Complete the following table for theoretical worst-case complexity of each algorithm. Also need to describe how the worst-case input of each algorithm should be.

Algorithm	theoretical worst-case complexity	describe the worst-case input
Classical matrix multiplication	$O(n^3)$ will be worst case with 3 nested for-loop	Input independent
Divide-and-conquer matrix multiplication	$O(n^3)$ will be worst case with 8 sub-problems and 8 recursive calls	Input independent
Strassen's matrix multiplication	$O(n^{\log 7})$ will be worst case using 7 multiplications and 18 additions/subtractions for 4x4 matrix	Input independent

b. Design the program by providing pseudocode or flowchart for each algorithm.

Classical:

```
For(i=0,i<4;i++)
    For(j=0;j<4;j++)
        For(k=0;k<4;k++)
            matrixC[i][j] += matrixA[i][k]*matrixB[k][j]
```

Divide and Conquer:

```
If (numOfCol ==1)
    matrixC[0][0] = matrixA[0][0]*matrixB[0][0]
else
    subDivider = numOfCol / 2
#Additions
    For(i=0,i<subDivider;i++)
        For(j=0;j<subDivider;j++){
                a00[i][j] = matrixA[i][j]
                a01[i][j] = matrixA[i][j + subDivider]
            a10[i][j] = matrixA[subDivider + i][j]
            a11[i][j] = matrixA[i + subDivider][j + subDivider]
            b00[i][j] = matrixB[i][j]
            b01[i][j] = matrixB[i][j + subDivider]
            b10[i][j] = matrixB[subDivider + i][j]
            b11[i][j] = matrixB[i + subDivider][j + subDivider]
        }
```

Then, add the product of each submatrices together such as a00*b00 + a01*b01, so on and so on

Strassen:

Compute using Strassen algorithm as below:

A,b,c,d are submatrix of matrixA and e,f,g,h are submatrix of matrix.

p1 = strassen(a, f - h)
p2 = strassen(a + b, h)
p3 = strassen(c + d, e)
p4 = strassen(d, g - e)
p5 = strassen(a + d, e + h)
p6 = strassen(b - d, g + h)
p7 = strassen(a - c, e + f)

Then, compute final values of 4 quadrants for matrixC

matrixC11 = p5 + p4 - p2 + p6
matrixC12 = p1 + p2
matrixC21 = p3 + p4
matrixC22 = p1 + p5 - p3 - p7

c. Design the program correctness testing cases. Design at least 10 testing cases to test your program, and give the expected output of the program for each case. We prepare for correctness testing of each of the three programs later generated in Part 2.

Testing case #	Input	Expected output	Classical matrix multiplication (√ if CORRECT output from your program)	Divide-and-conquer matrix multiplication (√ if CORRECT output from your program)	Strassen's matrix multiplication (√ if CORRECT output from your program)
1	Matrix A 4 0 2 9 Matrix B 8 4 7 0	Matrix C 32 16 79 8	√	√	√

2	A 2 8 7 7 B 8 9 1 9	 C 24 90 63 126	√	√	√
3	A 7 6 7 7 8 5 9 1 8 4 2 8 2 5 3 4 B 7 1 4 0 5 6 6 5 2 6 3 7 6 0 0 2	Matrix C 135 85 85 93 105 92 89 90 128 44 62 50 69 50 47 54	√	√	√
4	A 3 9 6 2 4 4 7 6 0 1 2 6 6 0 6 2 8 8 3 4 2 8 6 1 4 9 7 0 5 9 1 5 9 0 8 0 3 4 9 6 3 0 3 3 7 7 9 4 9 1 7 3 6 0 5 2 4 8 0 0 5 6 7 6	C 185 227 155 177 182 246 67 159 95 147 79 160 88 138 92 86 159 159 107 203 176 173 56 167 174 198 139 148 180 226 54 137 184 214 150 211 181 215 51 135 151 219 129 230 146 206 85 143 160 160 108 193 140 172 77 107 152 198 131 178 141 212 58 160	√	√	√

	B 4 0 1 7 4 1 1 3 5 3 3 0 4 6 0 6 7 6 5 1 7 7 0 1 0 3 0 9 5 1 8 1 6 8 4 8 1 9 6 4 2 4 2 5 5 3 0 4 6 8 5 8 4 7 0 8 2 9 6 4 5 8 4 0				
5	A 1 8 8 3 3 7 7 7 4 6 0 2 5 8 7 1 B 9 3 1 1 4 2 0 8 5 0 3 9 3 8 2 4	C 90 43 31 149 111 79 38 150 66 40 8 60 115 39 28 136	√	√	√
6	A 0 6 3 2 3 9 1 5 1 5 5 3 7 4 5 5	Matrix C 61 15 81 59 88 18 122 88 88 22 90 70 125 33 105 90	√	√	√

	B 4 2 2 3 3 1 9 6 9 3 5 5 8 0 6 4				
7	A 5 6 8 2 9 4 8 9 0 9 8 7 8 8 9 9 B 9 1 2 4 4 6 3 3 7 0 6 4 4 9 2 6	Matrix C 133 59 80 82 189 114 96 134 120 117 89 101 203 137 112 146	√	√	√
8	A 7 5 2 6 2 2 8 1 0 1 3 2 9 6 9 4 B 3 2 4 9 5 4 7 1 5 3 0 1 0 3 6 2	Matrix C 56 58 99 82 56 39 28 30 20 19 19 8 102 81 102 104	√	√	√

9	A 2 1 0 1 8 3 6 7 6 6 2 5 9 6 3 6 4 1 2 1 2 7 4 4 8 6 4 2 3 2 7 1 5 8 4 7 3 6 0 8 4 0 5 0 3 8 3 2 8 2 1 7 5 8 4 3 2 5 9 3 6 6 9 6 B 8 4 1 2 1 1 3 8 7 1 5 1 1 6 5 3 7 9 6 1 6 3 9 4 8 2 9 5 1 5 3 8 4 7 6 4 9 4 9 8 8 1 7 0 7 5 0 3 3 5 1 6 0 1 6 7 3 3 4 7 7 5 2 0	Matrix C 126 121 119 127 146 101 136 142 255 160 216 141 194 172 192 225 149 90 111 76 113 88 88 116 202 139 123 97 88 100 167 193 264 129 224 125 169 182 155 178 158 111 119 57 131 84 106 121 246 129 189 119 145 137 139 222 255 211 219 153 202 167 241 220	√	√	√
10	A 7 8 3 7 B 4 3 7 3	C 84 45 61 30	√	√	√

d. Design testing strategy for the programs. Discuss about how to generate and structure the randomly generated inputs for experimental study in Part 3.
Hint 1: The project will stop at the largest input size n (which is in the form of 2^k) that your computer environment can handle. It is the easiest to use a random generator to help generate numbers for the input data sets. However, student should store all data sets and use the same data sets to compare the performance of all three Matric Multiplication algorithms.
Hint 2: Note that even when running the same data set for the same Matrix Multiplication program multiple times, it's common that they have different run times as workloads of your computer could be very different at different moments. So it is desirable to run each data set multiple times and get the average run time to reflect its performance. The average run time of each input data set can be calculated after an experiment is conducted in m trails; but the result should exclude the best and worst run. Let X denotes the set which contains the m run times of the m trails, where $X = \{x_1, x_2, x_3 \ldots x_m\}$ and each x_i is the run time of the i^{th} trial. Let x_w be the largest time (worst case) and x_b be the smallest time (best case). Then we have

$$Average\ Run\ Time = \frac{\sum_{i=1}^{m} Xi \ - Xw - Xb}{m-2}$$

The student should think about and decide how many trials (the value of m) you will use in your experiment. Note that the common choice of the m value is at least 10.

1. How you generate and structure the randomly generated inputs?
   Using a random matrix value generator
2. What value of *m* you plan to use?

I'm planning to use m of 10

Part 2: Implementation (35 points)

a. Code each program based on the design (pseudocode or flow chart) given in Part 1(b).

<Generate three programs with proper documentation and store them in three files. Note: They are required to be submitted to Canvas as described in the submission instructions>
<No insert here>

b. Test you program using the designed testing input data given in the table in Part 1(c), Make sure each program generates the correct answer by marking a "√" if it is correct for each testing case for each program column in the table. Repeat the process of debugging if necessary.

<Complete the testing with testing cases in the table @Part 1(c)>
<No insert here>

c. For each program, capture a screen shot of the execution (Compile&Run) using one testing case to show how this program works properly

Classical

```
Trial 9
Matrix A
8 3
2 2

Matrix B
7 2
3 3

Matrix C
65 25
20 10

Trial 10
Matrix A
7 1
4 1

Matrix B
4 8
9 3

Matrix C
37 59
```

Divide and Conquer

```
Trial 8
Matrix A
7 6 7 7
8 5 9 1
8 4 2 8
2 5 3 4

Matrix B
7 1 4 0
5 6 6 5
2 6 3 7
6 0 0 2

Matrix C
135 85 85 93
105 92 89 90
128 44 62 50
69 50 47 54

0.009998798370361328
Trial 9
Matrix A
1 1 2 5
```

Strassen

```
Trial 6
Matrix A
2 7 8 3 7 3 7 6
7 9 5 7 6 9 2 8
6 1 8 6 4 5 9 3
4 3 2 7 0 4 2 2
3 9 2 5 0 9 5 9
2 2 1 2 1 0 6 7
3 0 4 4 8 4 5 8
2 5 1 0 1 1 8 5

Matrix B
7 6 3 9 6 5 1 2
5 8 3 6 4 1 3 0
9 3 8 7 5 6 3 1
4 3 9 4 4 3 5 9
4 5 1 5 1 6 8 0
2 8 2 0 4 4 0 1
6 6 2 3 0 8 1 5
4 7 5 2 2 2 4 4

Matrix C
233 244 175 196 123 196 149 101
253 320 219 232 189 199 166 133
235 221 186 193 134 219 116 136
117 133 122 106 94 92 64 95
188 276 170 148 138 145 102 123
109 127 86 82 48 92 63 81
175 200 143 142 94 171 136 107
122 151 73 94 52 105 56 66
```

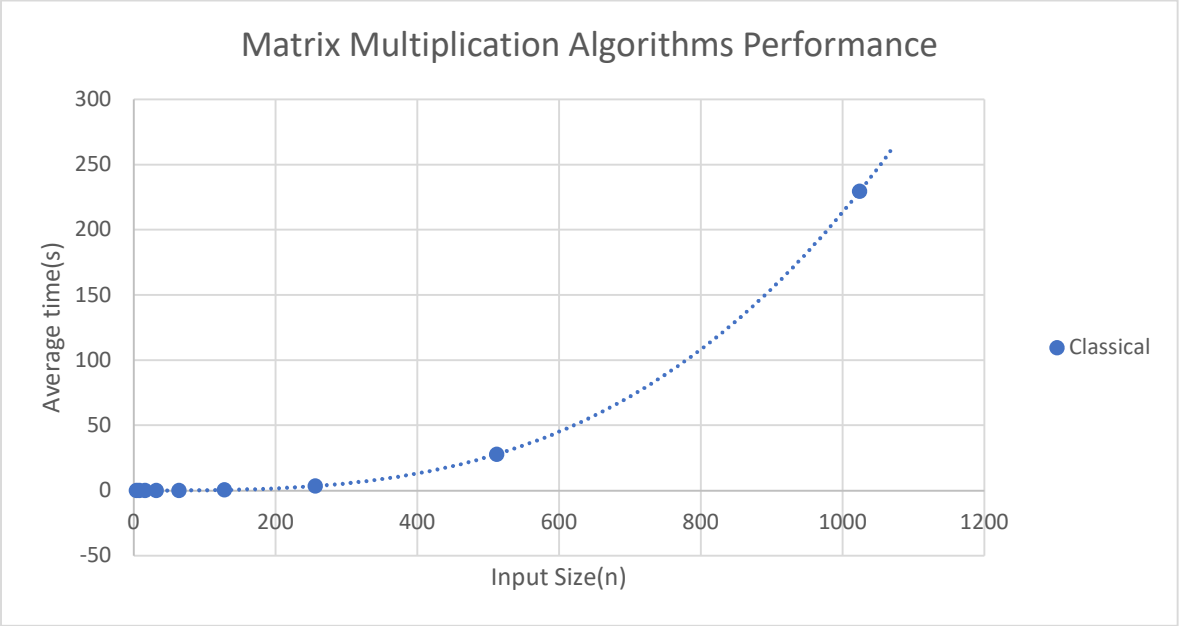
By now, three working programs for the three algorithms are created and ready for experimental study in the next part, Part 3.

Part 3: Comparative Analysis (35 points)

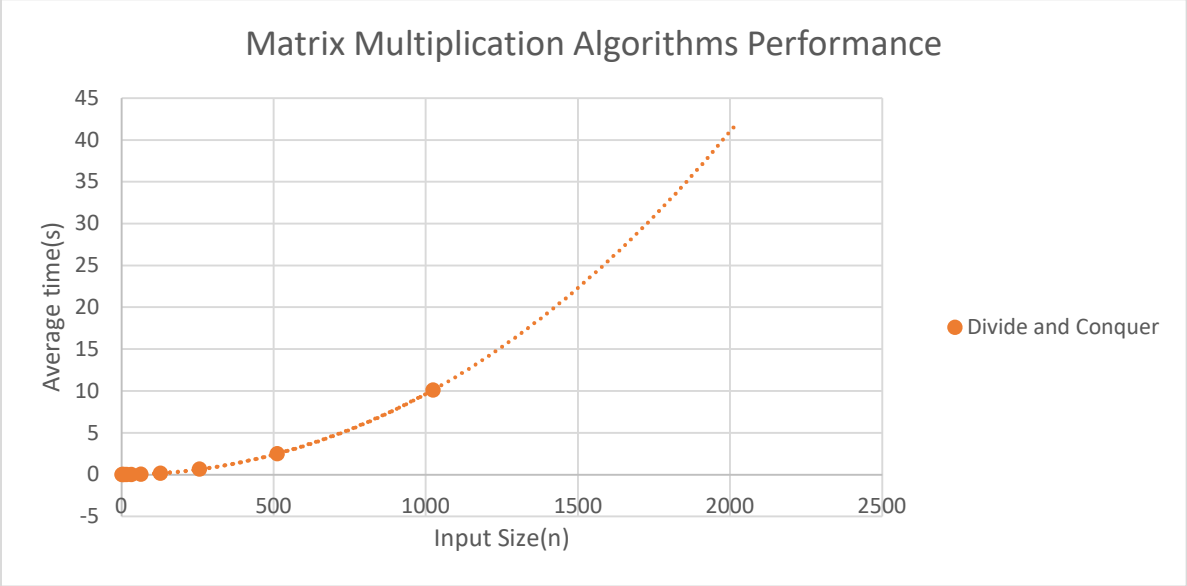
a. Run each program with the designed randomly generated input data given in Part 1(d). Generate a table for all the experimental results as follows.

Input Size n	Average time (Classical matrix multiplication)	Average Time (Divide-and-conquer matrix multiplication)	Average Time (Strassen's matrix multiplication)
2	0.00000	0.00000	0.00000
4	0.00012	0.00000	0.00000
8	0.00016	0.00012	0.00013
16	0.00188	0.001	0.00087
32	0.00957	0.00275	0.00250
64	0.06358	0.01037	0.01013
128	0.46192	0.03963	0.03925
256	3.54381	0.16359	0.15682
512	27.84373	0.66949	0.63406
1024	229.53435	2.50126	2.51620
2048	Too long	10.12456	10.02228

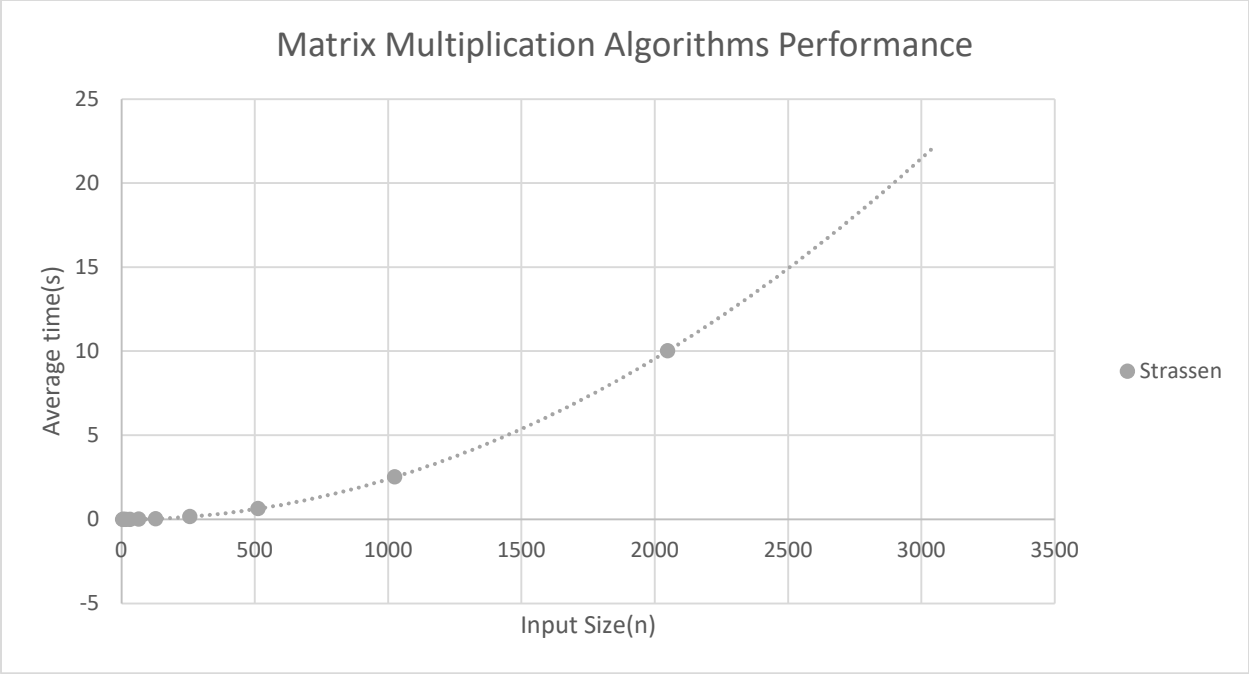
b. Plot a graph of each algorithm and summarize the performance of each algorithm based on its own graph.



Classical grows the fastest and eventually has the slowest execution time.

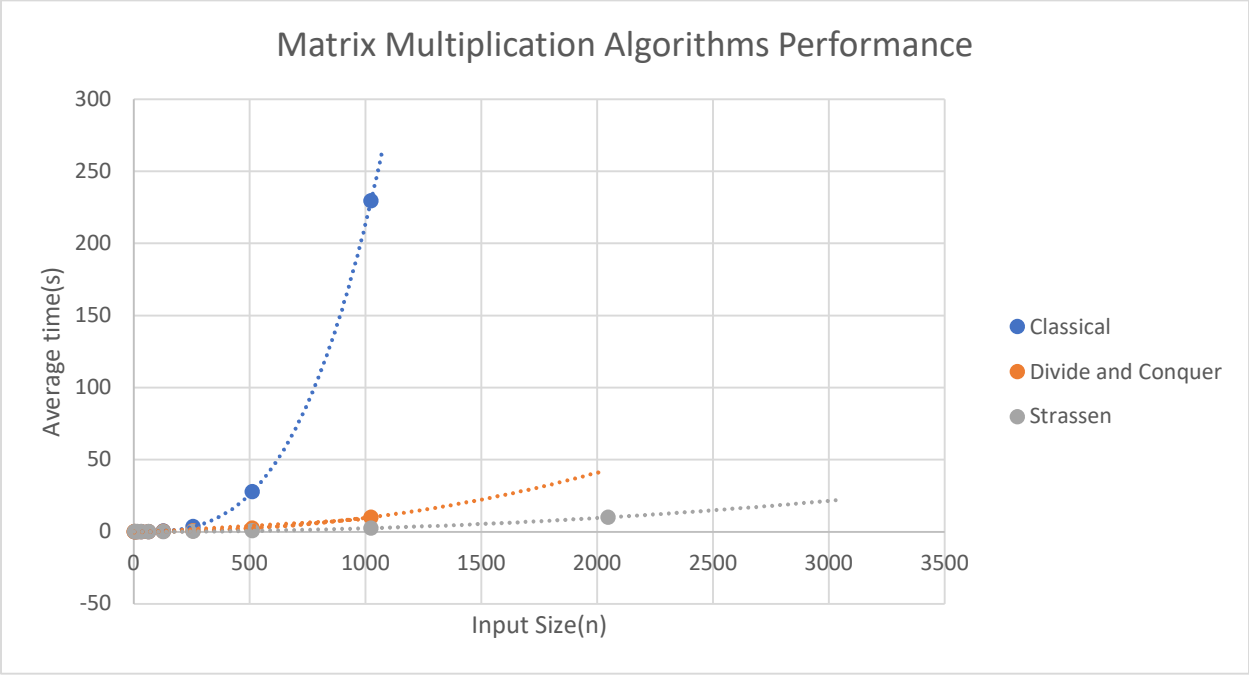


Divide and conquer grows a little slower and still relatively fast even with 2048x2048 matrices.



Strassen is the fastest with the lowest execution time

Plot all three graphs on the same graph and compare the performance of all three algorithms. Explain the reasons for which algorithm is faster than others.



Strassen is the fastest algorithm because of parallelization and less computationally intensive. Divide and conquer comes in second despite the same time complexity in the long run because it does speed up processing time. Classical is slowest because it's computationally intensive, involving many steps.

c. Compare the theoretical results in Part 1(a) and empirical results here. Explain the possible factors that cause the difference.

Though the time complexity for divide and conquer vs classical is the same, it slightly differs in reality with classical being slower. Strassen remains the fastest among the three which is on point with theoretical analysis

d. Give a spec of your computing environment, e.g. computer model, OS, hardware/software info, processor model and speed, memory size, ...

OS Name Microsoft Windows 10 Pro
AMD Ryzen 5 3600 6-Core Processor, 3593 Mhz, 6 Core(s), 12 Logical Processor(s)
Installed Physical Memory (RAM) 24.0 GB

e. Conclude your report with the strength and constraints of your work. At least 200 words. Note: It is reflection of this project. Ask yourself if you have a chance to re-do this project again, what you will do differently (e.g. your computing environment, programing language, data structure, data set generation, …) in order to design a better performance evaluation experiment.

During testing, I have to continually refine the output process from having to print out the matrix for double checking and streamlining it later by printing only the trial number the machine is running and the average time. The difference is noticeable because the processor don't have to spend precious memory outputting huge display of matrices 10 times which slow down and bias the true computational time of the algorithms. I would like to do this project in another language such as Java or C++ where they both might have faster execution time than Python, an interpreted language vs Java compilation.

The data set generation changes throughout the development process. At first, data were hardcoded into matrices when matrix size is small (2x2) but as the size grows toward 64 or 512, I decided to use a random number generator module in Python to generate a random number from 0-9 as a value for the matrix to speed up the process. I utilize functions over data structure in this project because it was simpler but I would like to try approaching it from an object oriented programming perspective. Finally, the timing device must be properly placed within the code so that it can accurately capture the true performance of the algorithm, ignoring execution of function definition.