A Comparative Performance Analysis of User-Defined Function Categories in C Programming

Bojken Shehu

Faculty of Information Technology, Tirana Polytechnic University, Tirana, Albania,
bshehu@fti.edu.al

Abstract: User-defined functions are fundamental to modular programming in C, enabling code reuse, maintainability, and structured program design. These functions are commonly classified based on the presence or absence of parameters and return values, resulting in four primary categories: functions without parameters and without return values, functions with parameters and without return values, functions without parameters but with return values and functions with both parameters and return values. Understanding the performance implications of these categories across compilation and execution is essential for developing efficient software.

This study presents a comparative performance analysis of user-defined function categories in C. A total of seven representative programs with diverse computational characteristics were implemented and evaluated in all four function categories. Performance assessment focused on three aspects: compile-time duration, runtime execution in seconds, and combined time reflecting overall program efficiency.

The results indicate that functions incorporating both parameters and return values generally exhibit slightly higher runtime and combined times due to additional stack operations and data handling, while functions without parameters and without return values tend to be more efficient. Compile-time differences were smaller but noticeable for programs with complex function structures. These findings highlight the influence of function design on compile-time and runtime performance, offering valuable guidance for programmers seeking to optimize modularity and efficiency in C programming.

Keywords: C Programming Languages, Modular Programming, User-defined Functions, Compile time, Run time, Performance Analysis.

1 Introduction

In the C programming language, functions play a fundamental role in structured program design. They enable developers to divide complex programs into smaller and more manageable components, thereby improving readability, maintainability, and code reuse [1]. By using user-defined functions, programmers can organize

code into logical units that perform specific tasks, which significantly simplifies program development and debugging [2].

User-defined functions in C can be classified according to the presence or absence of parameters and return values. Based on these characteristics, four primary categories can be identified: functions without parameters and without return values, functions with parameters and without return values, functions without parameters but with return values, and functions with both parameters and return values [3]. Each category represents a different mechanism of communication between the function and the calling program.

Although these categories are conceptually simple, their performance behavior may vary depending on how parameters and return values are handled during program compilation and execution. Operations such as parameter passing, return value handling, and stack management may introduce additional computational overhead that can influence program performance [4].

The objective of this study is to analyze how different categories of user-defined functions affect program performance. Seven programs with different computational characteristics were implemented across all four function categories in order to evaluate their behavior under various conditions.

1 Methodology

This section describes the experimental setup used to evaluate the performance of different categories of user-defined functions in C programming. The methodology includes the description of the programs used in the experiments, the function categories considered, the performance metrics used for evaluation, and the experimental environment in which the programs were executed.

2.1 Programs Description

Seven programs representing different computational tasks were selected for this study in order to evaluate the performance of user-defined function categories under various conditions, Table 1. The selected programs include common computational problems frequently used in programming analysis, such as array operations, numerical calculations, and algorithmic procedures. The evaluated programs are listed in Table 1.

Program	Program Description
P1	Array Sum.
P2	Array Maximum
P3	Factorial Calculation
P4	Fibonacci Calculation
P5	Matrix Multiplication
P6	Prime Number Check
P7	Sorting Algorithm

Table 1.
Seven programs representing different computational tasks.

Each program performs the same computational task across all implementations to ensure a fair comparison between the different function categories.

2.2 Function Categories

In this study, four categories of user-defined functions in the C programming language were analyzed. These categories are based on the presence or absence of parameters and return values.

The four categories considered in the experiment are:

a) Functions without parameters and without return values
b) Functions with parameters and without return values
c) Functions without parameters but with return values
d) Functions with parameters and return values

Each program was implemented using all four categories in order to evaluate how the structure of a function influences program performance.

2.3 Performance Metrics

To evaluate the performance of each implementation, three metrics were measured:

- ✓ Compile Time – the time required to compile the program before execution
- ✓ Run Time – the time required for the program to execute the computational task
- ✓ Combined Time – the total time obtained by summing compile time and run time

These metrics were selected to provide a comprehensive view of program performance during both compilation and execution phases.

2.4 Experimental Environment

All experimental programs were implemented in the C programming language and compiled using the Dev-C++ integrated development environment on a computer equipped with an Intel Core i5 processor, 8 GB RAM, and a 64-bit Windows operating system. Default compiler optimization settings were used during all experiments.

The performance measurements were obtained by executing each program five times in order to ensure consistent and reliable results. For each implementation, compile time and runtime were recorded, and the reported values correspond to the average of the five executions. Combined time was calculated as the sum of compile time and runtime. The measured values were expressed in seconds to maintain consistency across all evaluated programs and analyzed using tables and graphical representations.

2 Results

The experimental evaluation was conducted across seven programs implemented using the four categories of user-defined functions. The following tables and figures present the performance measurements obtained from the execution of the evaluated programs. The experimental results are presented in Tables 2–5.

Program numbers	Compile Time (s)			
	No parameters / No return	With parameters / No return	No parameters / With return	With parameters / With return
P1	0.011	0.012	0.011	0.013
P2	0.012	0.013	0.012	0.014
P3	0.011	0.012	0.011	0.013
P4	0.012	0.013	0.012	0.014
P5	0.012	0.013	0.012	0.014
P6	0.011	0.012	0.011	0.013
P7	0.012	0.013	0.012	0.014

Table 2.
Compile Time Results

Program numbers	Run Time (s)			
	No parameters / No return	With parameters / No return	No parameters / With return	With parameters / With return
P1	0.146	0.171	0.156	0.181
P2	0.151	0.176	0.162	0.186
P3	0.134	0.159	0.144	0.169
P4	0.141	0.166	0.152	0.176
P5	0.154	0.181	0.164	0.191
P6	0.148	0.170	0.158	0.182
P7	0.152	0.176	0.162	0.188

Table 3.
Run Time Results

As shown in Figure 1**,** the execution time trends across the evaluated programs indicate that functions without parameters and without return values demonstrate the lowest execution time. In contrast, functions with both parameters and return values exhibit slightly higher execution times due to additional parameter passing

and return value handling. The overall trend remains consistent across all tested programs.

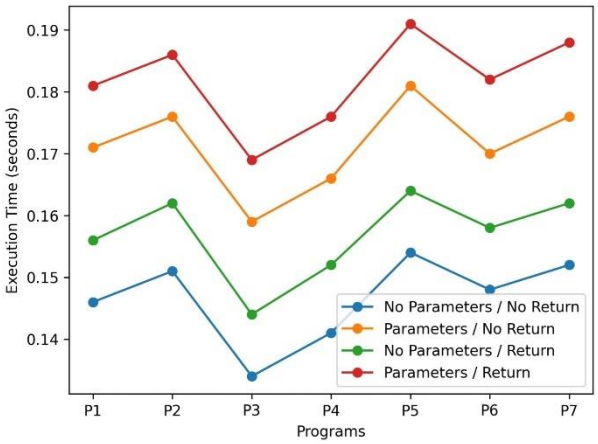

Figure 1

Execution time comparison across the seven evaluated programs

Program numbers	**Combined Time (s)**			
	No parameters / No return	**With parameters / No return**	**No parameters / With return**	**With parameters / With return**
P1	0.156	0.182	0.166	0.193
P2	0.162	0.188	0.172	0.199
P3	0.146	0.172	0.156	0.183
P4	0.152	0.178	0.162	0.189
P5	0.167	0.193	0.177	0.204
P6	0.159	0.182	0.169	0.195
P7	0.164	0.189	0.174	0.202

Table 4.

Combined Time Results

Figure 2 illustrates the execution time trends across the evaluated programs. The results indicate that functions without parameters and without return values consistently demonstrate the lowest execution time, while functions with both parameters and return values show slightly higher execution overhead.

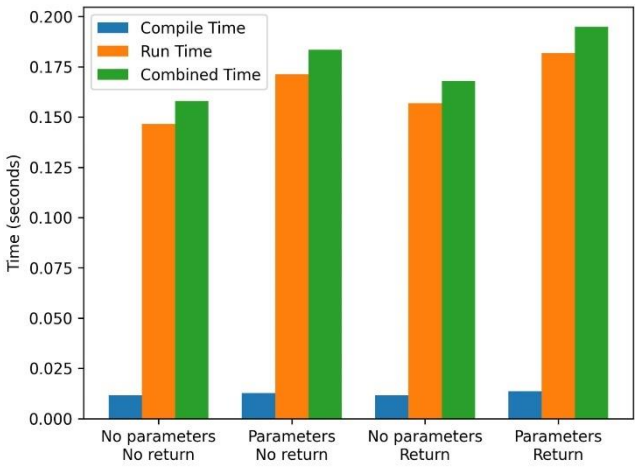

Figure 2

Comparison of compile time, run time, and combined time

Figure 3 and Table 5 presents the average combined execution time across the seven tested programs. The results indicate that functions with parameters and return values exhibit the highest combined time, while functions without parameters and without return values demonstrate the lowest overall execution overhead.

Function Categories	**Avg Compile Time (s)**	**Avg Run Time (s)**	**Avg Combined Time (s)**
No parameters / No return	0.01157	0.14657	0.158
With parameters / No return	0.01257	0.17129	0.18343
No parameters / With return	0.01157	0.15686	0.168
With parameters / With return	0.01357	0.181	0.195

Table 5.

Average performance across seven programs

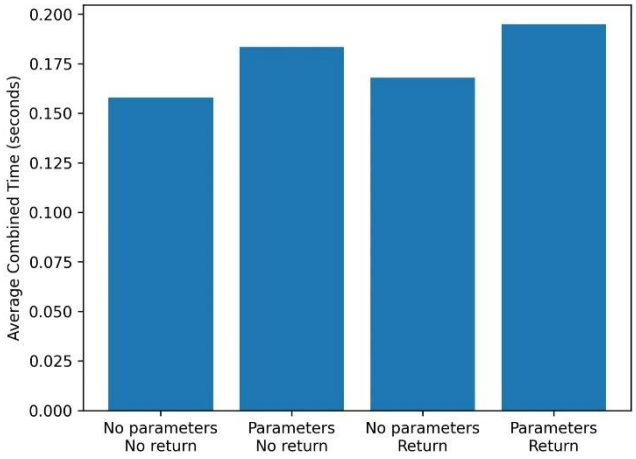

Figure 3

Average combined execution time across seven programs

The experimental evaluation was conducted across seven programs implemented using the four categories of user-defined functions in C. The results indicate that compile-time differences among the four function categories are relatively small. This behavior is expected because the compilation process primarily focuses on syntax verification and translation of source code into machine instructions [5].

More noticeable differences were observed in runtime performance. Functions without parameters and without return values generally demonstrated the lowest execution time. Since these functions do not require parameter passing or return value handling, the overhead associated with stack operations is reduced.

Functions that include parameters or return values showed slightly higher execution times. In particular, functions with both parameters and return values exhibited the highest runtime measurements among the tested categories. This behavior can be attributed to additional data transfer and stack manipulation during function calls.

The observed performance trends remain consistent across all evaluated programs, indicating that the structural characteristics of user-defined functions have a measurable but relatively moderate impact on execution performance. Similar experimental analyses of programming structures have also been reported in previous programming studies [9].

3 Discussion

The results obtained from the experimental evaluation reveal several important patterns regarding the performance of user-defined function categories in C programming. Although the computational tasks performed by each program were identical across the four function categories, differences in performance were observed in runtime execution.

Functions without parameters and without return values generally demonstrated slightly lower execution times. This behavior can be explained by the absence of parameter passing and return value handling during the function call. Since no data needs to be transferred between the calling program and the function, fewer stack operations are required, resulting in more efficient execution.

Functions that include parameters require additional operations to transfer input values from the caller to the function. Similarly, functions that return values require extra steps to send results back to the caller. These additional operations introduce minor overhead during program execution [6].

Although the measured differences in execution time are relatively small, the results demonstrate that parameter passing and return value handling introduce slight overhead during function calls. Similar effects have been discussed in previous studies related to software maintainability and program structure optimization [10].

Conclusion

This study presented a comparative analysis of four categories of user-defined functions in C programming across seven different programs. The evaluation focused on three performance metrics: compile time, run time, and the combined time of both processes.

The results indicate that functions without parameters and without return values generally provide the lowest execution overhead, while functions with parameters and return values introduce slightly higher runtime due to additional stack operations and data transfer. Overall, the observed performance differences remain relatively moderate, suggesting that developers should prioritize code readability and modular design while applying performance optimizations mainly in critical sections of the program [7]. Although the measured differences are relatively small, the results confirm that the structural characteristics of functions can influence execution performance in C programming.

These findings emphasize the importance of considering function structure when designing efficient and maintainable C programs.

Future studies may extend this analysis by evaluating additional programming environments, larger datasets, or more complex program structures to further investigate the relationship between function design and program performance [8].

References

[1] Kernighan, B. W., Ritchie, D. M: The C Programming Language, 2nd ed, 1988. Prentice Hall.

[2] Prata, S: *C* Primer Plus, *6th* ed, 2013. Addison-Wesley.

[3] King, K. N: C Programming: A Modern Approach, 2nd ed, 2008. W. W. Norton.

[4] Deitel, P., Deitel, H: C How to Program, 8th ed, 2016. Pearson.

[5] Harbison, S. P.,Steele, G. L: C: A Reference Manual, 5th ed, 2002. Prentice Hall.

[6] Schildt, H: C The Complete Reference, 4th ed, 2000. McGraw-Hill.

[7] Gustedt, J: Modern C. Manning Publications. 2019.

[8] Seacord, R. C: Secure Coding in C and C++. Addison-Wesley, 2013.

[9] Downey, A. B: Think C: Computer Science in C, 2015. Green Tea Press.

[10] Feathers, M: Working Effectively with Legacy Code, 2004, Prentice Hall.