Final Assignment

CSE-0302 Summer 2021

Niaz Ahmed Nayeem
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
E-mail: niazahmed.net@gmail.com

***Abstract*—Assignment Index Terms—code in c/c++**

I. Introduction

This assignment is given by Compiler design course. The assignment is done with c and c++ code. In this report I learn how to detect syntax error and also learn about CFG. And I also learn about Predictive Parsing. In which I can now parse a given grammar.

II. LITERATURE REVIEW

The term "lexical" in lexical analysis process of the compilation is derived from the word "lexeme", which is the basic conceptual unit of the linguistic morphological study. In computer science, lexical analysis, also referred to as lexing, scanning or tokenization, is the process of transforming the string of characters in source program to a stream of tokens, where the token is a string with a designated and identified meaning. It is the first phase of a two-step compilation processing model known as the analysis stage of compilation process used by compiler to understand the input source program. The objective is to convert character streams into words and recognize its token type. The generated stream of tokens is then used by the parser to determine the syntax of the source program. A program in compilation phase that performs a lexical analysis process is termed as lexical analyzer, lexer, scanner or tokenizer. Lexical analyzer is used in various computer science applications, such as word processing, information retrieval systems, pattern recognition systems and language-processing systems. However, the scope of our review study is related to language processing. Various tools are used for automatic generation of tokens and are more suitable for sequential execution of the process. Recent advances in multi-core architecture systems have led to the need to reengineer the compilation process to integrate the multi-core architecture. By parallelization in the recognition of tokens in multiple cores, multi cores can be used optimally, thus reducing compilation time. To attain parallelism in tokenizationon multi-core machines, the lexical analyzer phase of compilation needs to be restructured to accommodate the multi-core architecture and by exploiting the language constructs which can run parallel and the concept of processor affinity. This paper provides a systematic analysis of literature to discuss emerging approaches and issues related to lexical analyzer implementation and the adoption of improved methodologies. This has been achieved by reviewing 30 published articles on the implementation of lexical analyzers. The results of this review indicate various techniques, latest developments, and current approaches for implementing auto generated scanners and hand-crafted scanners. Based on the findings, we draw on the efficacy of lexical analyzer implementation techniques from the results discussed in the selected review studies and the paper provides future research challenges and needs to explore the previously under-researched areas for scanner implementation processes.

III. Proposed Methodology

I used C and C++. In those programming languages, I used to file for input and output.

IV. Conclusion and Future Work

In the future, I will learn more about CFG. And also learn about C and C++, I will use the framework, and use this for making anther big projects.

Acknowledgment

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] elm, R., Maurer, D. (1995). Compiler design. Reading: AddisonWesley Publishing Company.
[2] Grune, D., Van Reeuwijk, K., Bal, H. E., Jacobs, C. J., Langendoen, K. (2012). Modern compiler design. Springer Science Business Media
[3] Hoare, C. A. R., Jifeng, H., Sampaio, A. (1993). Normal form approach to compiler design. Acta informatica, 30(8), 701-739.
[4] Hoare, C. A. R., He Jifeng, and Augusto Sampaio. "Normal form approach to compiler design." Acta informatica 30.8 (1993): 701-739.
[5] Bozkus, Z., Choudhary, A., Fox, G., Haupt, T., Ranka, S. (1993, November). Fortran 90D/HPF compiler for distributed memory MIMD computers: Design, implementation, and performance results. In Supercomputing'93: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing (pp. 351-360). IEEE.
[6] Bozkus, Zeki, et al. "Fortran 90D/HPF compiler for distributed memory MIMD computers: Design, implementation, and performance results." Supercomputing'93: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing. IEEE, 1993.
[7] Millstein, R. E. (1971). Compiler Design for the ILLIAC 4. MASSACHUSETTS COMPUTER ASSOCITAES INC WAKEFIELD.

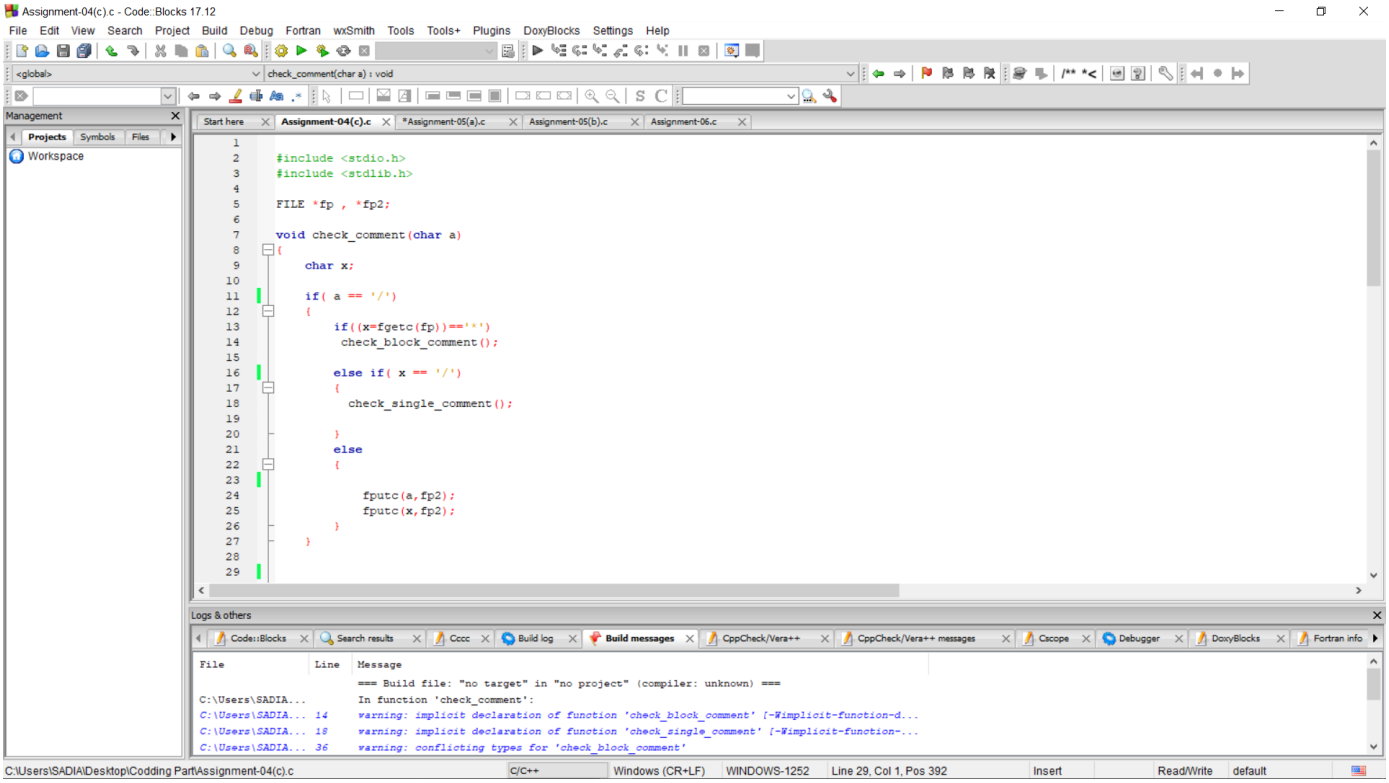

Fig. 1. Assignment-04

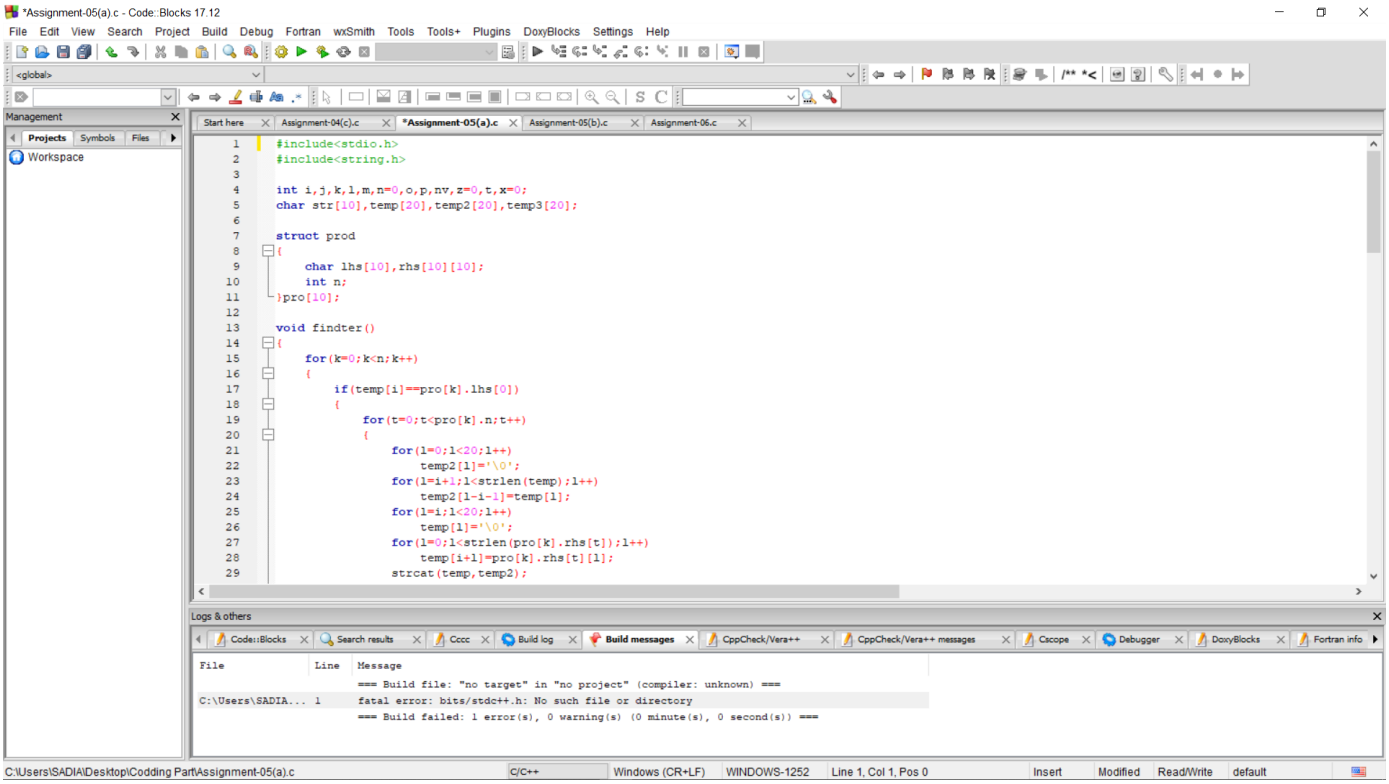

Fig. 2. Assignment-05(a)

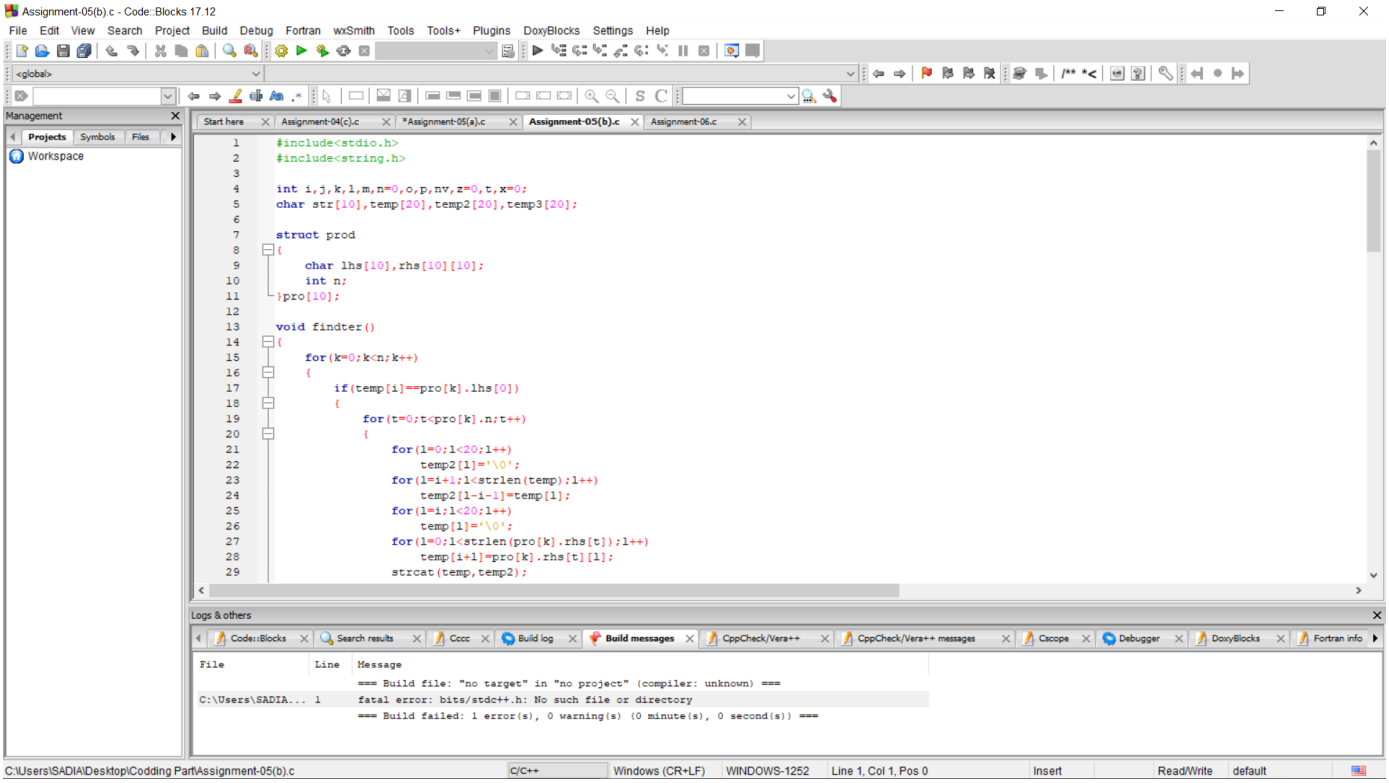

Fig. 3. Assignment-05(b)

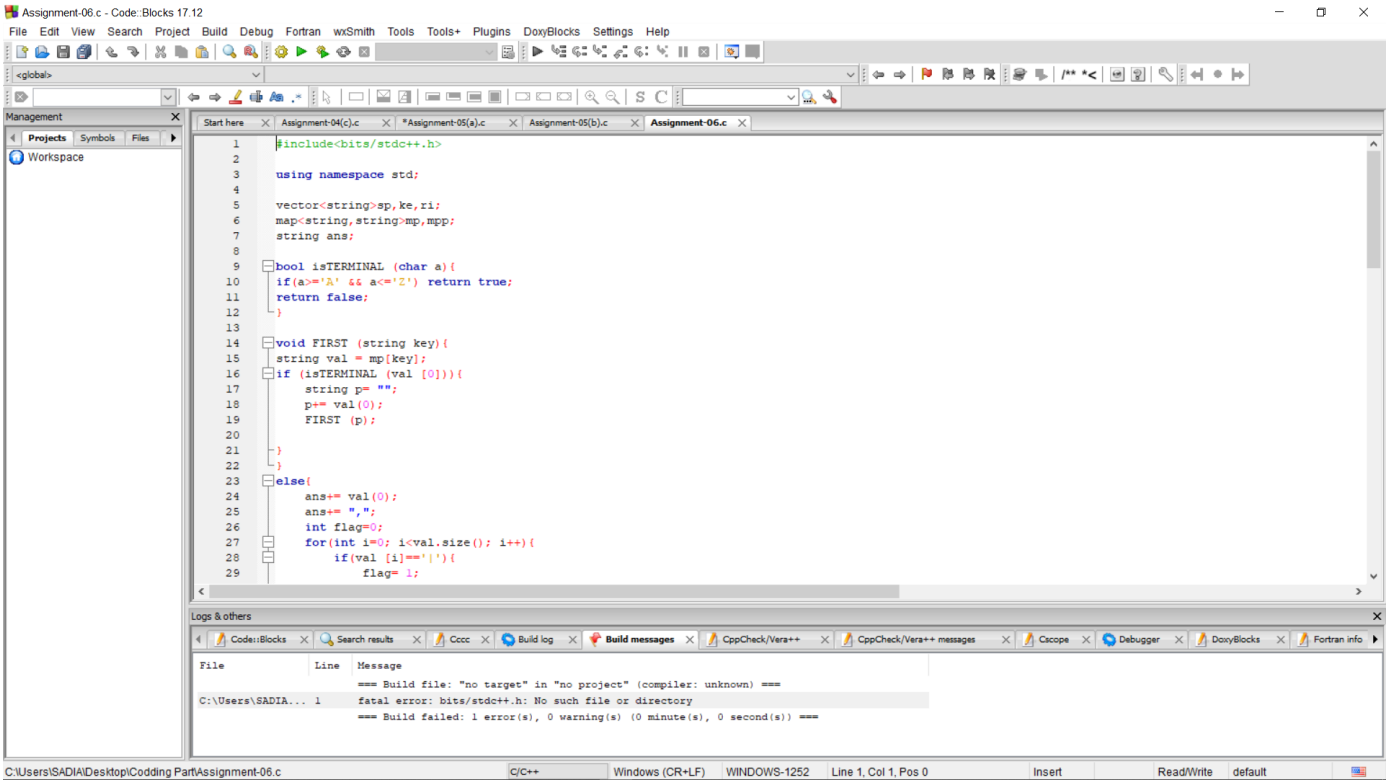

Fig. 4. Assignment-06