Compiler For Mathematical Operations Using English Like Sentences

Arlagadda Naga Likhith[1], Kothuru Gurunadh[2], Vimal Chinthapalli[3], Meena Belwal[4]
Department of Computer Science & Engineering, Amrita School of Computing, Amrita Vishwa Vidyapeetham, Bengaluru, India
nagalikhith.arlagadda@gmail.com[1] , guruvenkat99@gmail.com[2] , vimal17042002@gmail.com[3] , b_meena@blr.amrita.edu[4]

2023 7th International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS) | 979-8-3503-4314-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSITSS60515.2023.10334225

Abstract: This work revolves around the creation of a unique tool designed to interpret and execute mathematical operations articulated in English-like sentences. The tool developed using Lex and YACC, popular tools for creating lexical analyzers and parsers. The aim is to make mathematical computations more accessible and user-friendly. It bridges the gap between complicated mathematical grammar and everyday language by converting specified English words into executable code.

Keywords: Lex, YACC, Lexical Analysis, Parser, Scanner, Syntax, Semantics.

I. INTRODUCTION

The goal of this project is to implement an innovative method for calculating mathematical equations. This solution enables users to enter instructions using natural English phrases rather than complicated coding languages. All people, regardless of their background, should find it easier and more convenient to perform mathematical tasks.

Lexical Analysis:
As part of the lexical analysis phase, the Lex tool is used. To identify different types of tokens in English-like expressions, it uses various regular expressions, including integers, mathematical operators, key terms, punctuation, and others. Tokens identified are not only associated with specific types, but they are also used during the parsing process.

Parsing:
YACC tool is employed for the parsing stage. In this process, a grammar set resembling natural English sentences is created. This grammar outlines the structure of sentences, such as the sequence of operations, operands, and specific terms. The YACC code lays down these grammatical guidelines along with the actions to be taken when a rule aligns.

Syntax and Semantics:
For phrases that resemble those in real English, the established grammar rules provide an organized format that ensures they follow a predetermined order. Corresponding acts enable the semantics, or meaning, of the statements to be understood. For instance, the tool executes the appropriate addition calculation and modifies the output when it finds the "ADD" command. The grammar similarly establishes rules for various mathematical operations and control instructions, making their execution simpler and influencing the entire calculating process.

Lex Tool:
Lex serves as a potent instrument for lexical analysis, the process of dissecting text into tokens or significant segments. It enables developers to formulate regular expressions that recognize patterns in the input data and associate them with specific actions. Lex is proficient at generating efficient lexical analyzers in a range of programming languages, making it an essential asset for developing compilers, interpreters, and text-processing software. Figure 1 illustrates the functioning of the Lex compiler.

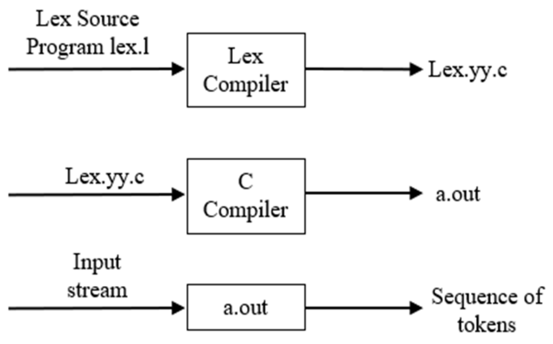


Figure 1: Working of Lex

YACC:
YACC, also known Yet Another Compiler Compiler, complements Lex by generating parsers based on a formal grammar specification. It accepts a grammar file that defines the syntax of a language or a programming construct as input and generates parsing code in a target programming language. YACC employs the LALR(1) parsing algorithm to construct efficient parsers capable of handling complex grammar. Figure 2 depicts the working of YACC compiler.

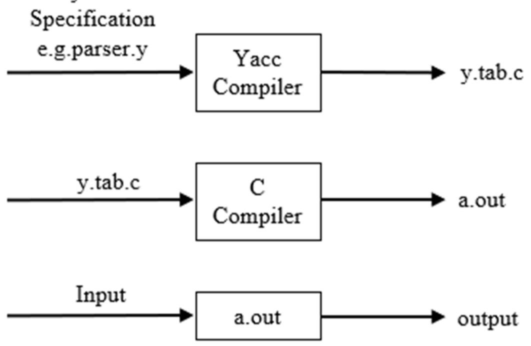
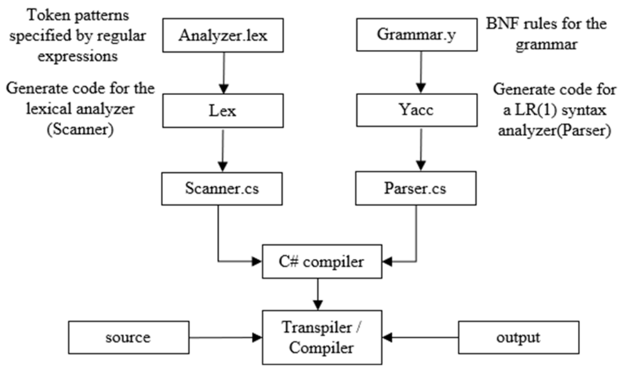


Figure 2: Working of YACC

The incentive to create a tool for converting English mathematical expressions to answers is to simplify the process of performing mathematical operations for non-technical users. Many individuals, particularly those without a programming or math background, may find it daunting to write and understand traditional code or mathematical syntax. By developing a tool that can interpret English sentences and convert them into executable code, it becomes more convenient for non-technical users to interact with the system and get answers to their mathematical queries.

Communication between the scanner and parser is a crucial aspect of the compiler design. The scanner, also known as the lexical analyzer, reads the input stream and breaks it down into a series of tokens. These tokens, which represent the smallest meaningful units of the program, are then passed onto the parser.

The parser, in turn, takes this stream of tokens and verifies if they follow the syntax rules of the language. It constructs a parse tree, a hierarchical structure that represents the syntactic structure of the input based on its grammar. This parse tree is then used by the interpreter or compiler for further processing. Figure 3 depicts how communication is done between scanner and parser.


Token patterns specified by regular expressions
Analyzer.lex
Grammar.y
BNF rules for the grammar
Generate code for the lexical analyzer (Scanner)
Lex
Yacc
Generate code for a LR(1) syntax analyzer(Parser)
Scanner.cs
Parser.cs
C# compiler
source
Transpiler / Compiler
output
Figure 3: Communication between Scanner and Parser

This tool holds potential benefits in a multitude of scenarios, marking the future trajectory of the work. Natural language can be employed for performing calculations in educational contexts, for example, reducing the requirement for specialized programming skills. Voice and natural language integration could further ease the process by translating spoken words into the needed language grammar. This is especially useful in applications that require continuous mathematical computations, as it allows users to express their calculations in a style that feels intuitive and familiar. This tool can serve as a gateway for those new to programming by bridging the gap between human language and programming constructs. It can ignite users' interest in coding and computational thinking, leading to deeper exploration of traditional programming languages over time.

Future Upgrades: Speech-to-Text Integration:
One of the key enhancements on the horizon for this initiative is the incorporation of speech-to-text technology. This feature would empower users to articulate their mathematical problems vocally, which the system would then convert into written text for further processing and computation. This upgrade would make the system even more approachable and user-centric, especially benefiting those who find the act of typing to be a hurdle.
The Role of Natural Language Processing (NLP):
Another area with potential for development is the incorporation of Natural Language Processing (NLP) methods. NLP is a branch of artificial intelligence that focuses on easing communication between machines and human language. The system could understand and manage more complex and varied sentences constructed in English-like language by integrating NLP capabilities. This would make the system more adaptable and resilient by enabling it to handle a wider range of mathematical operations and even interpret context, synonyms, and other linguistic complexities.

Extending Mathematical and Logical Capabilities:
Additionally, there is a chance for the system to improve to include more difficult mathematical and logical operations, such as complex conditional logic and logarithmic and trigonometric functions. This evolution would make the tool a comprehensive platform for conducting a wide variety of mathematical computations, meeting diverse user needs.

Ultimately, the aspiration is to democratize the execution of mathematical operations, rendering them as uncomplicated and accessible as possible, particularly for those with limited technical expertise. The development of this tool, which translates English mathematical expressions into actionable solutions, broadens the scope and inclusivity of mathematical computations. It aims to make these tasks transparent and user-friendly for individuals from various backgrounds and levels of expertise.

II. LITERATURE SURVEY

In the field of natural language processing and compiler design, it's been amazing converting English-like words into practical mathematical processes. The development of this discipline may be traced back to several seminal studies that changed our understanding and capabilities.

M. Upadhyaya et al. [1] set out to create a simple calculator compiler using the well-known tools Lex and YACC. This endeavor was motivated by a desire to bridge the gap between plain language and mathematical computing. They successfully tokenized arithmetic expressions using regular expressions and generated grammar rules in YACC by leveraging Lex for lexical analysis and YACC for parsing. Their work demonstrates the power of these technologies, particularly when it comes to dealing with faults and guaranteeing accurate compilation.

Brown et al. [2] built on this basis to provide a full introduction to the world of Lex and YACC. Their research goes deeply into fundamental principles, offering insight on advanced topics like mistake recovery and optimization. Waite [3] expanded on this underlying knowledge by demonstrating how Lex and YACC could be used to construct a full compiler. Waite's research did not end there; he also looked at the possibilities of other parser generation tools, most notably ANTLR.

Speaking of ANTLR, Ortín et al. [4] and another study in 2019 [5] both embarked on a comparative analysis between Lex/YACC and ANTLR. Their findings were consistent in highlighting ANTLR's superiority in terms of speed, accuracy, ease of use, and flexibility. Such comparative studies are crucial in guiding researchers and developers in choosing the right tools for their specific applications.

Diving into the technical intricacies, a team [6] presented a detailed report on the challenges and solutions related to parsing and scanning MATLAB code within the MATCH compiler. Their insights are invaluable for those looking to develop compilers for MATLAB and other dynamically typed languages.

For those new to the field, Ikotsireas [7], a publication in 2018 [8], and ShareOK [9] all offer concise introductions to Lex and YACC. These works cover the fundamental concepts and provide practical examples, making them perfect starting points for budding researchers.
For those seeking a more comprehensive understanding, "A Compiler-based Approach for Natural Language to Code Conversion" [10] by S. Sridhar and S. Sanagavarapu. This work promises a deep dive into how NLP can be integrated with compiler for more user accessibility.

"Parser Tools," [11] this work explores the developments in parsing techniques in the field of telecommunications. It also discusses different types of parsers.

"ONESTOP," a versatile tool for general tasks with visual support, was introduced by Ganesan, Gowtham, Subikshaa Senthilkumar, and Senthil Kumar Thangavel [12]. Their work contributes to improving the accessibility and effectiveness of diverse tasks with visual assistance by tackling the problems of executing operations with visual aids.

Sankaravelayuthan, Rajendran, and K. Krishnakumar [13] investigated shallow parsing and machine translation in Malayalam. Their findings provide insight on approaches for improving machine translation quality and crossing the language barrier through more accurate and context-aware translations.

Shailashree K. Sheshadri, Deepa Gupta, and Marta R. Costa-Jussá [14] used pre-trained embeddings in neural machine translation for Kashmiri to English and Hindi. Their work makes use of innovative approaches to improve translation accuracy and fluency, making cross-lingual communication easier.

Varshini, Surisetty Hima, Gottimukkala Sarayu Varma, and M. Supriya [15] investigate a recognizer and parser for simple Telugu texts utilizing the CYK algorithm. Their work helps to further the development of tools for efficient sentence parsing and comprehension in regional languages.

Reddy, Bandi Rupendra, Daka Chandra Rup, Mathi Rohith, and Meena Belwal [16] focus on Indian Sign Language generation from live audio or text for Tamil. Their study helps the cause of inclusivity and communication for those with hearing impairments by overcoming the gap between spoken and sign languages.

The studies [12-16] highlight attempts to improve language translation efficiency and accessibility, as well as to develop novel methods for turning sign language into text. The researchers are helping to create a more interconnected world by breaking down language barriers and making communication more fluid, which benefits various societies and fosters understanding across linguistic and sensory boundaries.

Finally, the road to developing a compiler for mathematical operations using English-like phrases is littered with ground-breaking research and inventions. The literature reviewed here provides a glimpse into the accomplishments made as well as the fascinating possibilities that lie ahead.

III. METHODOLOGY

The work, "Mathematical Operations Using English-Like Sentences," builds a system that reads and executes mathematical operations described in English-like words using Lex and YACC tools. This work's technique is outlined below:

A. Lexical Analysis with Lex: The first step is to use Lex, a lexical analysis tool. The process of breaking down incoming text into meaningful pieces known as tokens is known as lexical analysis. Tokens can represent numbers, arithmetic operations, keywords, punctuation marks, and other information.

Lex identifies these tokens using a set of predefined regular expressions. Each token is assigned to a certain token type, which is then used in the parsing process. The Lex code has numerous tokens, each associated with a specific mathematical operation or command, such as ADD for addition, SUBTRACT for subtraction, MULTIPLY for multiplication, DIVIDE for division, MODULO for modulus operation, and several others.

B. Parsing with YACC: The second step is parsing, it is done with YACC (Yet Another Compiler Compiler). Parsing involves developing a grammar for English-like sentences. Grammar rules determine sentence structure, including the order of operations, operands, and keywords.

The YACC code defines these language rules as well as the actions that must be taken when a rule is matched. The grammar contains rules for adding numbers, removing numbers from operands, multiplying numbers, dividing the result by a number, and modulus operations on the result by a number.

C. Syntax and Semantics: The grammatical rules created during the parsing process ensure that the English-like sentences are formatted correctly. Through related actions, the semantics of the phrases are captured. When the tool sees a "ADD" statement, for example, it conducts the addition operation and modifies the output accordingly. Similarly, the grammar sets rules for various arithmetic operations and control flow statements, allowing them to be executed and their impact on the entire computation to be evaluated.

D. Conditional Statements: Conditional statements are also supported by the system, allowing users to execute commands based on situations. The YACC code defines the syntax for an “if" statement, which includes conditions for less than, greater than, equal to, and not equal to. These conditions' actions are also described, allowing the system to undertake different activities dependent on the condition's outcome.

E. Integration and Testing: After developing the lexical analyzer and parser, they are integrated and tested. The goal is for the system to be able to correctly comprehend and execute English-like statements describing mathematical operations. User feedback is collected and used to make any necessary system improvements. Table 1 illustrates how Operations are treated as Tokens.

TABLE I
TOKEN TABLE

Operations	Return as
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Modulo	MOD
Show me the answer	SHOW
,	COMMA
from	FROM
to it	TO
By	BY
The result	THE_RESULT
with	WITH
.	DOT
?	QMARK
if	IF
then	THEN
else	ELSE
<	LT
>	GT
==	EQ
!=	NEQ

The methodology of this work is designed to make mathematical operations as accessible and user-friendly as possible. By converting English math expressions into answers, the tool caters to a wider audience and makes math calculations more inclusive, straightforward, and accessible to individuals of different backgrounds and skills. Figure 5 depicts the flow diagram of how the system works. And Figure 4 depicts basic Grammar accepted by the compiler.

```
ADD numbers DOT
SUB NUMBER FROM NUMBER DOT
MUL numbers DOT
DIV NUMBER WITH NUMBER DOT
MOD THE_RESULT BY NUMBER DOT
SHOW DOT
```

Figure 4: Basic Grammar

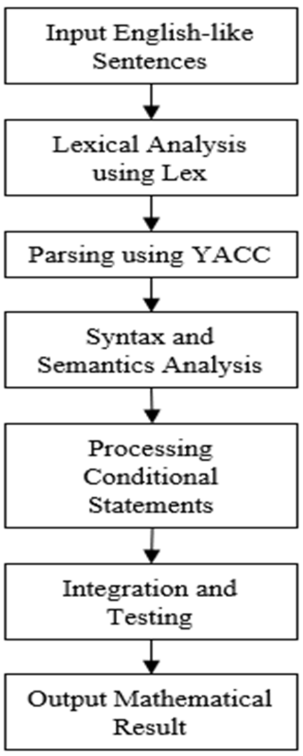

Input English-like Sentences
Lexical Analysis using Lex
Parsing using YACC
Syntax and Semantics Analysis
Processing Conditional Statements
Integration and Testing
Output Mathematical Result

Figure 5. Flow diagram

The grammar rules specify the structure of the input sentences and the actions to be taken when each rule is matched. For example, the "ADD numbers DOT" rule adds the numbers provided in the input and assigns the result to the global variable "result." Similarly, other rules handle subtraction, multiplication, division, modulo, and displaying the result. This work also supports conditional statements like "IF," "THEN," and "ELSE" which perform actions based on certain conditions.

IV. RESULTS

The custom-built compiler designed for executing mathematical tasks using language resembling everyday English has proven both reliable and high performing. By allowing for straightforward user input like "Add 5, 10, 15," the program swiftly carries out the calculation and returns the result promptly.

The compiler's functionality extends beyond mere addition. It is also capable of handling subtraction, multiplication, division, and modulo operations, all while maintaining an intuitive and user-friendly interface.

One noteworthy feature is the compiler's ability to process conditional statements, allowing for the dynamic execution of commands based on predetermined conditions. For instance, a command such as "if 3 10 then Add 5" will increment the existing result by 5, provided that the current value is less than 10. If the condition isn't satisfied, the command is ignored, providing a versatile and adaptable approach to solving mathematical problems.

In summary, the test results validate the compiler's effectiveness and efficiency in executing mathematical tasks using language that resembles everyday English. Its successful implementation highlights its utility for broader applications, including educational settings or any scenario requiring intuitive and easy-to-use mathematical computing. A table presents the different operations carried out by the compiler along with their respective outputs.

TABLE II
OPETATIONS PERFORMED AND THEIR OUTPUT

Input	Output
Add 5,6	11
Multiply 3 to it.	33
Subtract 13 from result.	20
Divide the result by 5.	4
Multiply 2 to it.	8
Modulo of the result by 5.	3
Show me the answer	3
If 21>33 then Add 3, 6 else Multiply 6 to it.	18

V. CONCLUSION

The Compiler for Mathematical Operations Using English-Like Sentences provides a simple interface for carrying out mathematical operations. It is specifically designed for people with little experience with programming, and it makes use of intuitive syntax. In order to construct a compiler that can comprehend English-like words and do the associated math operations, this project successfully combines the strength of Lex and YACC.

There is plenty of room to improve the system in the future by supporting a wider range of mathematical operations and by extending the language rules to cover more complex circumstances. Although the system's current implementation mainly concentrates on simple conditional statements and basic arithmetic, with further updates it has the potential to develop into a more adaptable and powerful tool.

In conclusion, this project marks a substantial development in the user-friendly, approachable, and user-centered development of math calculations. It makes it easier to carry out mathematical operations and opens up new possibilities for computational thinking for those with less specialized knowledge. The accomplishment of this project establishes a solid platform for subsequent research in the area with the goal of providing a more accessible and understandable method for performing mathematical calculations.

VI. FUTURE SCOPE

This project offers a lot of opportunities for further development. The prospective integration of speech-to-text features and Natural Language Processing, along with the addition of support for more advanced calculations, could fundamentally change the way users engage with mathematical tasks. This marks a crucial advancement toward a future where interfacing with computational platforms could be as simple and natural as speaking in English.

REFERENCES

[1] M. Upadhyaya, "Simple calculator compiler using Lex and YACC," 2011 3rd International Conference on Electronics Computer Technology, Kanyakumari, India, 2011, pp. 182-187, doi: 10.1109/ICECTECH.2011.5942077.

[2] D. Brown, J. Levine, and T. Mason, "Lex & Yacc," O'Reilly Media, Inc., 1992.

[3] W. Waite, "Beyond Lex & Yacc: How to Generate the Whole Compiler," ResearchGate, May 12, 2014, [Online]. Available: https://www.researchgate.net/publication/305665640_Beyond_Lex_Yacc_How_to_Generate_the_Whole_Compiler

[4] F. Ortín, J. Quiroga, O. Rodríguez-Prieto, and M. García, "An empirical evaluation of Lex/Yacc and ANTLR parser generation tools," PLoS ONE, vol. 17, no. 3, e0264326, 2022, doi: 10.1371/journal.pone.0264326.

[5] "A Comparative Study of Lex/Yacc and ANTLR," IASJ International Journal of Applied Science and Technology, vol. 9, no. 1, pp. 11-18, 2019, [Online]. Available: https://www.iasj.net/iasj/article/29208

[6] P. G. Joisha, et al., "The Design and Implementation of a Parser and Scanner for the MATLAB Language in the MATCH Compiler," Technical Report, Centre for Parallel and Distributed Computing, 1999.

[7] I. Ikotsireas, "Yacc Introduction," [Online]. Available: https://web.wlu.ca/science/physcomp/ikotsireas/CP465/W3-BNF-LEX-YACC/Yacc_Introduction.pdf

[8] "Lex and Yacc," ProQuest, 2018, [Online]. Available: https://www.proquest.com/openview/0a2a52de35da6a3cf4bba6f33093f06c/1?pq-origsite=gscholar&cbl=18750&diss=y

[9] "Lex and Yacc," ShareOK, [Online]. Available: https://shareok.org/handle/11244/17346

[10] S. Sridhar and S. Sanagavarapu, "A Compiler-based Approach for Natural Language to Code Conversion," 2020 3rd International Conference on Computer and Informatics Engineering (IC2IE), Yogyakarta, Indonesia, 2020, pp. 1-6, doi: 10.1109/IC2IE50715.2020.9274674.

[11] "Parser Tools," PLT: Purdue Laboratory for Education in Telecommunications, 2017, [Online]. Available: https://plt.cs.northwestern.edu/snapshots/current/pdf-doc/parser-tools.pdf

[12] Ganesan, Gowtham, Subikshaa Senthilkumar, and Senthil Kumar Thangavel. "ONESTOP: A Tool for Performing Generic Operations with Visual Support." In Proceedings of the International Conference on ISMAC in Computational Vision and Bio-Engineering 2018 (ISMAC-CVB), pp. 1565-1583. Springer International Publishing, 2019.

[13] Sankaravelayuthan, Rajendran, and K. Krishnakumar. "A Comprehensive Study of Shallow Parsing and Machine Translation in Malaylam." Coimbatore: Amrita Vishwa Vidyapeetham, Coimbatore, 2019.

[14] Sheshadri, Shailashree K., Deepa Gupta, and Marta R. Costa-Jussá. "Neural Machine Translation for

Kashmiri to English and Hindi using Pre-trained Embeddings." In 2022 OITS International Conference on Information Technology (OCIT), pp. 238-243. IEEE, 2022.

[15] Varshini, Surisetty Hima, Gottimukkala Sarayu Varma, and M. Supriya. "A Recognizer and Parser for Basic Sentences in Telugu using CYK Algorithm." In 2023 3rd International Conference on Intelligent Technologies (CONIT), pp. 1-5. IEEE, 2023.

[16] Reddy, Bandi Rupendra, Daka Chandra Rup, Mathi Rohith, and Meena Belwal. "Indian Sign Language Generation from Live Audio or Text for Tamil." In 2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS), vol. 1, pp. 1507-1513. IEEE, 2023.