



PROJECT PROPOSAL

"THREE PHASE SELF-COMPILING MINI-COMPILER FOR C++"

COMPILER CONSTRUCTION CS-412
BACHELOR OF SCIENCE
IN
COMPUTER SCIENCE

SUBMITTED BY:
MUHAMMAD ANWAR | 18B-057-CS
SECTION B

UNDER THE GUIDANCE OF
SYED FAISAL ALI
ASSISTANT PROFESSOR
UIT UNIVERSITY, KARACHI

DEPARTMENT OF COMPUTER SCIENCE
UIT UNIVERSITY

ST-13, BLOCK 7, GULSHAN-E-IQBAL, ABUL HASAN ISPHAHANI ROAD, OPPOSITE SAFARI PARK, KARACHI 75300.

Contents

1.	Project Title	3
2.	Introduction.....	3
3.	Context Free Grammar	3
4.	Symbol Table Manager.....	4
5.	Lexical Analyzer and Parser	4
6.	Error Handling	5
7.	Motivation.....	5
8.	Project Details.....	5
a.	Environment	5
b.	Issues and challenges for implementation	5
c.	Deliverables	6
d.	Timeline	6
9.	Working of the Parser	6
i.	Setting Up Environment:	6
ii.	Downloaded Dependencies:	6
10.	Conclusion	8
11.	References.....	8

Project Proposal Outline

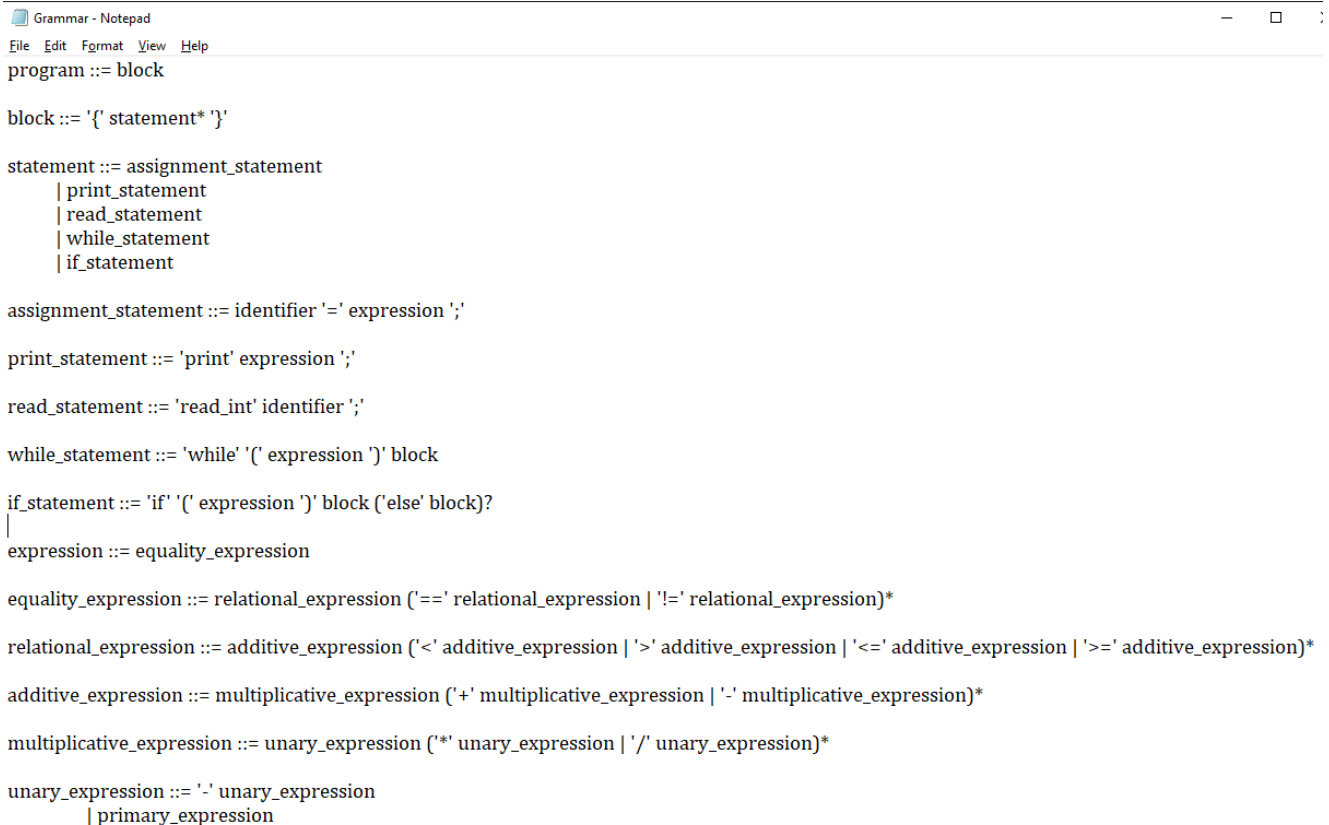
1. Project Title

Three Phase Self-Compiling Mini-Compiler for C++

2. Introduction

In this project, I have built a self-compiling C++ mini compiler. Nowadays, C++ is widely used for competitive programming since it is a general-purpose programming language. The language supports imperative, object-oriented, and generic programming. This compiler is built primarily around the constructs of a "while" loop and a "for" loop. Logic, Boolean, and arithmetic operations are also identified by the compiler. I expect this project to generate a Symbol Table, an Abstract Syntax Tree, and Intermediate Three-Address code with Optimization.

3. Context Free Grammar



```
Grammar - Notepad
File Edit Format View Help
program ::= block

block ::= '{' statement* '}'

statement ::= assignment_statement
           | print_statement
           | read_statement
           | while_statement
           | if_statement

assignment_statement ::= identifier '=' expression ';'

print_statement ::= 'print' expression ';'

read_statement ::= 'read_int' identifier ';'

while_statement ::= 'while' '(' expression ')' block

if_statement ::= 'if' '(' expression ')' block ('else' block)?

expression ::= equality_expression

equality_expression ::= relational_expression ('==' relational_expression | '!=' relational_expression)*

relational_expression ::= additive_expression ('<' additive_expression | '>' additive_expression | '<=' additive_expression | '>=' additive_expression)*

additive_expression ::= multiplicative_expression ('+' multiplicative_expression | '-' multiplicative_expression)*

multiplicative_expression ::= unary_expression ('*' unary_expression | '/' unary_expression)*

unary_expression ::= '-' unary_expression
                  | primary_expression
```

```

primary_expression ::= identifier
                    | integer_literal
                    | '(' expression ')'

identifier ::= [a-zA-Z][a-zA-Z0-9]*

integer_literal ::= [0-9]+

```

4. Symbol Table Manager

```

#include <map>
#include <string>

class SymbolTable {
public:
    // Add a symbol to the table
    void AddSymbol(std::string name, std::string type);

    // Look up a symbol in the table
    std::string LookupSymbol(std::string name);

    // Remove a symbol from the table
    void RemoveSymbol(std::string name);

private:
    std::map<std::string, std::string> symbols_;
};

void SymbolTable::AddSymbol(std::string name, std::string type) {
    symbols_[name] = type;
}

std::string SymbolTable::LookupSymbol(std::string name) {
    if (symbols_.count(name) == 0) {
        return "";
    } else {
        return symbols_[name];
    }
}

void SymbolTable::RemoveSymbol(std::string name) {
    symbols_.erase(name);
}

```

5. Lexical Analyzer and Parser

The mini compiler handles most cases in the early C++ compilers. Some of the expected features of the Lexer are: -

1. Identifies and removes comments.
2. Identifies various operators in the language.
3. Checks for validity of the identifiers.
4. Identifying numeric constants and std::string type

5. Ignores white spaces.
 6. Identifies scope of variables
- Syntax is handled by YACC where grammar rules are specified for the entire language.
 - Semantics are handled using semantic rules for type checking while performing operations, to ensure operations are valid.

6. Error Handling

I am handling syntax errors, which are generated during parsing. We are also handling re-declaration of variables in the same scope and are showing appropriate error messages. We stop parsing the input on encountering these errors and display the line number of the errors, intending the user to resolve the issue. Also handling type mismatch errors.

7. Motivation

The aim of this project is to implement the learning from the course Compiler Construction offered at our university Usman Institute of Technology. I have to make sure that I understood the concepts well by coming up with the course project.

8. Project Details

a. Environment

- i. I will be using Dev-C++ (Compiler for C++) for coding in C++.
- ii. YACC (Yet another Compiler Compiler – a utility in Linux) in BISON with FLEX for generation of grammar.
- iii. I will be implementing Symbol Table Manager with parsing tool YACC The end product will be a Mini Compiler which will handle basic computations.

b. Issues and challenges for implementation

- i. Error handling would be done better, I might face problem in this part of the coding.

- ii. More optimizations could have been implemented; I might not be able to optimize the compiler very well Project will have a limitation on C++ version above 14.

c. Deliverables

- i. The deliverable will be a complete compiler written in C for basic calculations.

d. Timeline

- i. The project submission deadline is 27/01/2023.

9. Working of the Parser

i. Setting Up Environment:

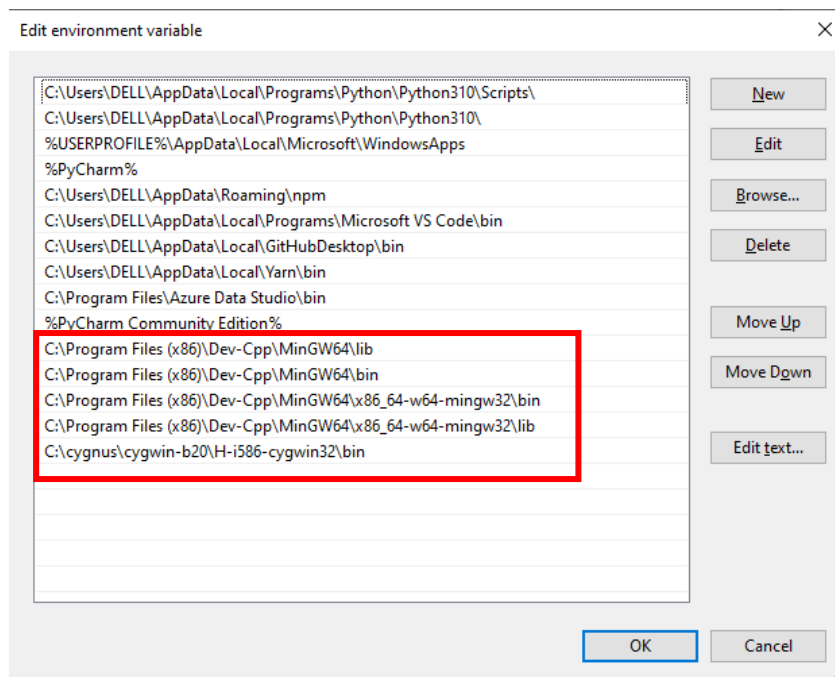


Figure 1

ii. Downloaded Dependencies:

- GNU (Mingw)
- Bison
- Flex

iii. Following Files are Created:

- ast.h
- code_generator.h
- lexer.h
- semantic_analyzer.h
- code_generator.cpp
- compiler.cpp
- lexer.cpp
- semantic_analyzer.cpp
- STM.cpp
- Example1.cpp
- Example2.cpp
- Example3.cpp

iv. Working of Compiler:

See Video:

https://drive.google.com/file/d/1MD2qYU4HqnMfyckWexjCGXZiXHGGEtHG/view?usp=share_link

```
C:\Users\DELL\Desktop\Compiler Project>g++ Example2.cpp

C:\Users\DELL\Desktop\Compiler Project>a.exe
Enter an integer: 4
You entered 4
C:\Users\DELL\Desktop\Compiler Project>g++ Example3.cpp

C:\Users\DELL\Desktop\Compiler Project>a.exe
Enter two integers: 2
5
2 + 5 = 7
C:\Users\DELL\Desktop\Compiler Project>g++ Example4.cpp

C:\Users\DELL\Desktop\Compiler Project>a.exe
Enter dividend: 9
Enter divisor: 2
Quotient = 4
Remainder = 1
C:\Users\DELL\Desktop\Compiler Project>g++ Example5.cpp

C:\Users\DELL\Desktop\Compiler Project>a.exe
Enter an integer: 71
71 is odd.
C:\Users\DELL\Desktop\Compiler Project>g++ Example6.cpp

C:\Users\DELL\Desktop\Compiler Project>a.exe
Enter three numbers: 4
7
2
Largest number: 7
```

10. Conclusion

By doing this project, I have gained a better insight into the phases of compiler. YACC provided us with a better knowledge about bottom-up parser and while performing the different phases of the compiler, our code efficiency in writing code and dealing with complex data structures has significantly improved.

11. References

- [Lex & Yacc, O'Reilly, John R Levine, Tony Mason, Doug Brown](#)