COMPILER DESIGN

PROJECT REPORT

GID 24

PID 18

Develop a parser in Python language that accepts code in C++ and checks for syntax errors and implement in LLVM. (Expected: boolean, char, string).

NAME	En no
Taddi Satya Sai Shyam Sundar	21114102
Sapavath Yashwanth Krishna Naik	21114093
Munugala Charan Tej	21114062

GITHUB REPO link: https://github.com/shyamsundar0212/Compiler_Design_Project

Our Project contains 3 files.

1. Lexer.py
2. Parser.py
3. Text file (for C++code input)

Lexer: Tokenizes the input C++ program into individual tokens stores these tokens in a list cache named "tokens".

Parser: It parses the tokens created by the lexer and maintains a symbol table to check if any variable is undeclared in the scope.

Algorithm

1. Lexer Algorithm:

1. Read the input C++ file line by line.
2. For each line, tokenize it into individual tokens.
3. Identify keywords, identifiers, operators, literals, etc.
4. Store each token in the tokens list cache.
5. Repeat the process for all lines in the input file.
6. Return the list cache tokens.

Input: Input C++ file.

Output: List cache named tokens.

2. Parser Algorithm:

1. Initialize an iterator i starting from 0 to iterate over the tokens list.

2. Implement functions to handle different parts of the C++ syntax:

   a. checkFuncDefOrVarDef: Checks for function or variable declaration/definition.
   b. ignorePreprocessor: Handles preprocessing directives.
   c. ignoreTillSemicolon: Ignores statements until a semicolon is encountered.
   d. checkVarHeader: Matches the pattern <datatype> <identifier> for variable declarations/definitions.
   e. checkScope: Handles scopes enclosed by curly braces {} and updates the symbol table accordingly.
   f. checkExpr: Parses expressions and checks their validity.
   g. checkLval: Parses lvalues in expressions.
   h. checkFuncDecOrValDef: Parses function declarations and variable declarations/definitions.

3. Define **Grammar rules** for the C++ program:

   - Program consists of function definitions, function declarations, variable declarations, variable definitions, preprocessors, and using statements.

- Expressions are parsed to ensure the validity of operators and operands.
- Scopes are handled with proper symbol table management to check for undeclared variables.
- Function declarations and variable declarations/definitions are parsed according to the specified grammar rules.
- ***Program → function def | function decl | variable decl | variable def | preprocessor | using statement*** - BNF GRAMMER

4. Execute the parsing functions iteratively based on the tokens:

- Use the iterator i to iterate over the tokens list.
- Based on the current token, call the corresponding parsing function to handle different parts of the syntax.

5. Return: Parsed structure of the C++ program.

Input: List cache of tokens generated by the Lexer.

Output: Parsed structure of the C++ program.

❖ *Our Parser follows a bottom-up approach where we start by defining the smallest units (tokens) and then build up to larger constructs (expressions, declarations, etc.*

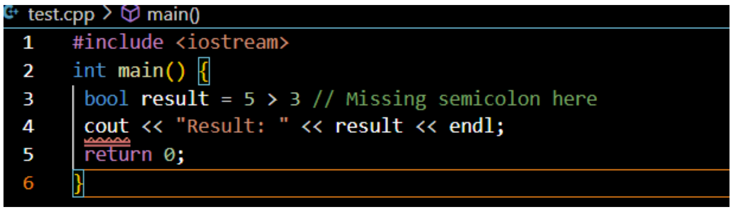
Testing Approach

Each function in the parser is tested independently to ensure its correctness and functionality. Integration testing is then performed to ensure that all functions work together seamlessly to parse the input program correctly.

Conclusion

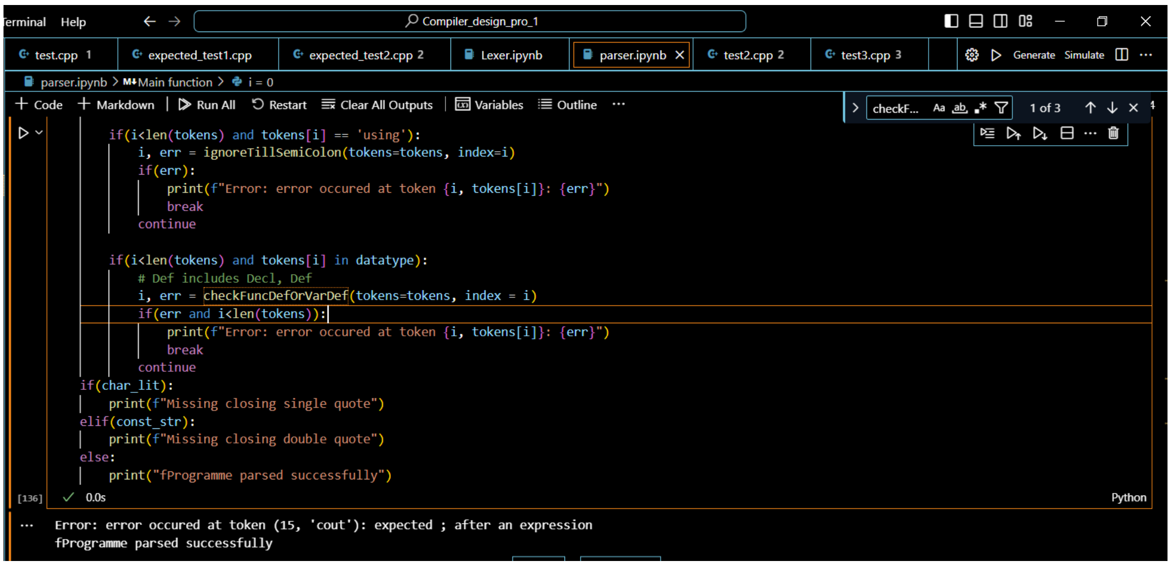
The outlined parser design offers a systematic method for parsing C++ input programs. This design ensures compliance with grammar rules, effective symbol table management, and reliable error handling.

<u>Test case 1</u>: C ++ code with a syntax error in a boolean expression.



```cpp
#include <iostream>
int main() {
 bool result = 5 > 3 // Missing semicolon here
 cout << "Result: " << result << endl;
 return 0;
}
```

Output:

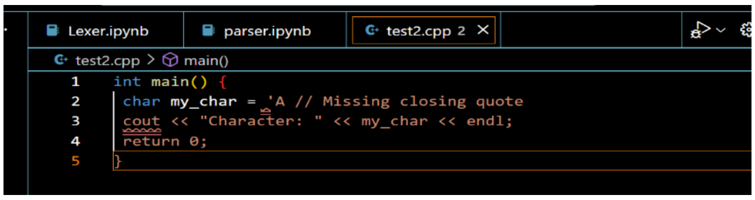


```python
        if(i<len(tokens) and tokens[i] == 'using'):
            i, err = ignoreTillSemiColon(tokens=tokens, index=i)
            if(err):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue

        if(i<len(tokens) and tokens[i] in datatype):
            # Def includes Decl, Def
            i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
            if(err and i<len(tokens)):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue
    if(char_lit):
        print(f"Missing closing single quote")
    elif(const_str):
        print(f"Missing closing double quote")
    else:
        print("fProgramme parsed successfully")
```

[136] ✓ 0.0s

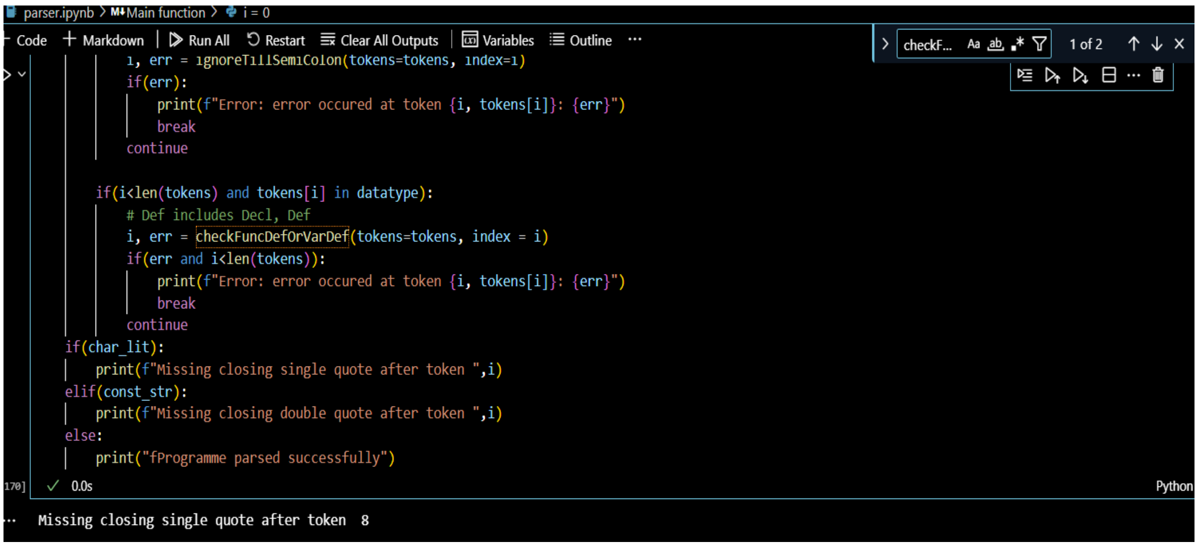
```
Error: error occured at token (15, 'cout'): expected ; after an expression
fProgramme parsed successfully
```

<u>Test Case 2</u>: C++ code with a syntax error in a character declaration.



```cpp
int main() {
 char my_char = 'A // Missing closing quote
 cout << "Character: " << my_char << endl;
 return 0;
}
```

Output:

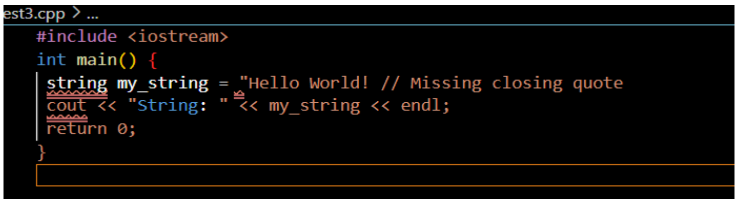


```python
            i, err = ignoreTillSemiColon(tokens=tokens, index=i)
            if(err):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue

        if(i<len(tokens) and tokens[i] in datatype):
            # Def includes Decl, Def
            i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
            if(err and i<len(tokens)):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue
    if(char_lit):
        print(f"Missing closing single quote after token ",i)
    elif(const_str):
        print(f"Missing closing double quote after token ",i)
    else:
        print("fProgramme parsed successfully")
```

170] ✓ 0.0s

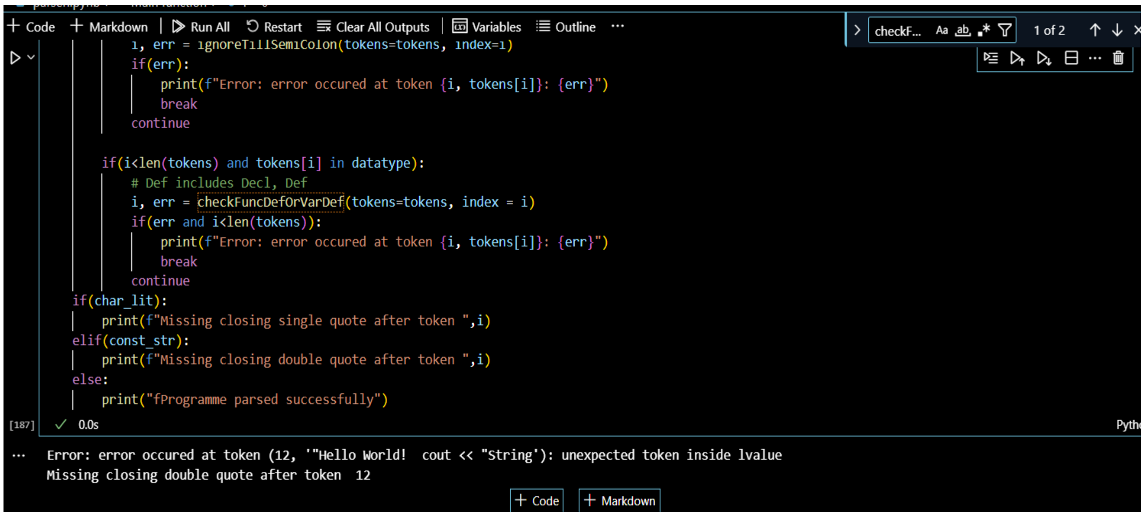
```
Missing closing single quote after token  8
```

<u>Test case 3:</u> C++ code with a syntax error in a string declaration.



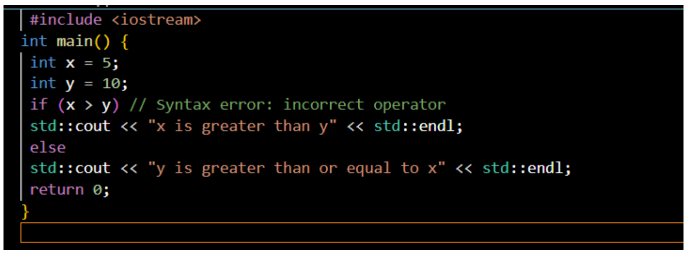
```
est3.cpp > ...
#include <iostream>
int main() {
 string my_string = "Hello World! // Missing closing quote
 cout << "String: " << my_string << endl;
 return 0;
}
```

Output:



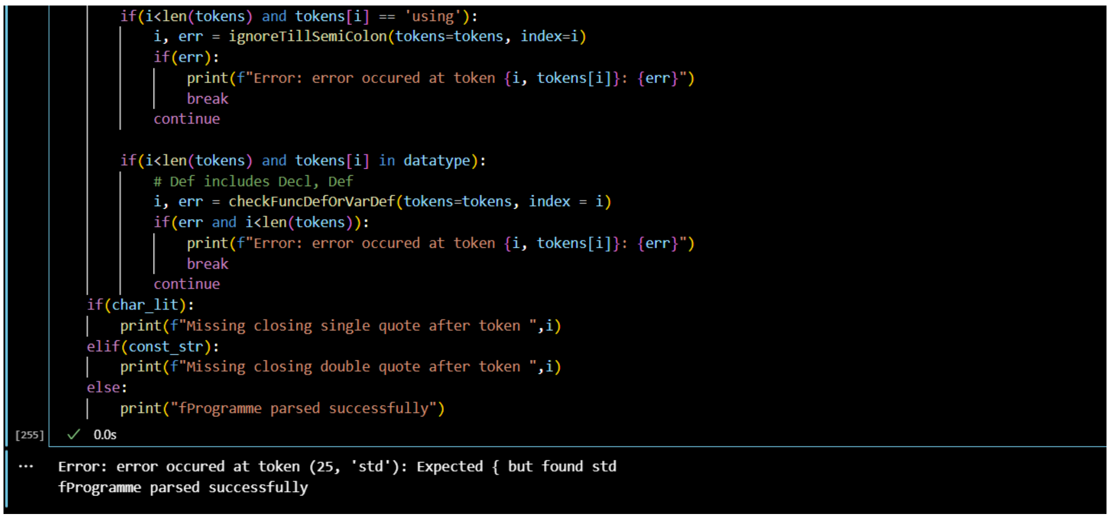
```
            i, err = ignoreTillSemiColon(tokens=tokens, index=i)
            if(err):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue

        if(i<len(tokens) and tokens[i] in datatype):
            # Def includes Decl, Def
            i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
            if(err and i<len(tokens)):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue
    if(char_lit):
        print(f"Missing closing single quote after token ",i)
    elif(const_str):
        print(f"Missing closing double quote after token ",i)
    else:
        print("fProgramme parsed successfully")
[187] ✓ 0.0s

Error: error occured at token (12, '"Hello World!  cout << "String'): unexpected token inside lvalue
Missing closing double quote after token  12
```

<u>Expected test case 1</u>: C++ code with missing opening brace.



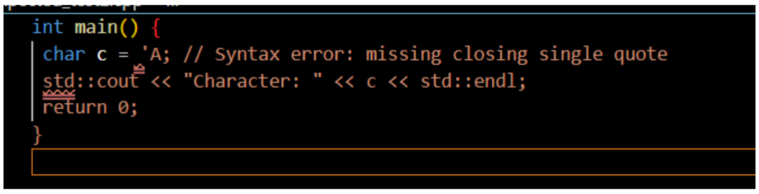
```
 #include <iostream>
int main() {
 int x = 5;
 int y = 10;
 if (x > y) // Syntax error: incorrect operator
 std::cout << "x is greater than y" << std::endl;
 else
 std::cout << "y is greater than or equal to x" << std::endl;
 return 0;
}
```

Output:



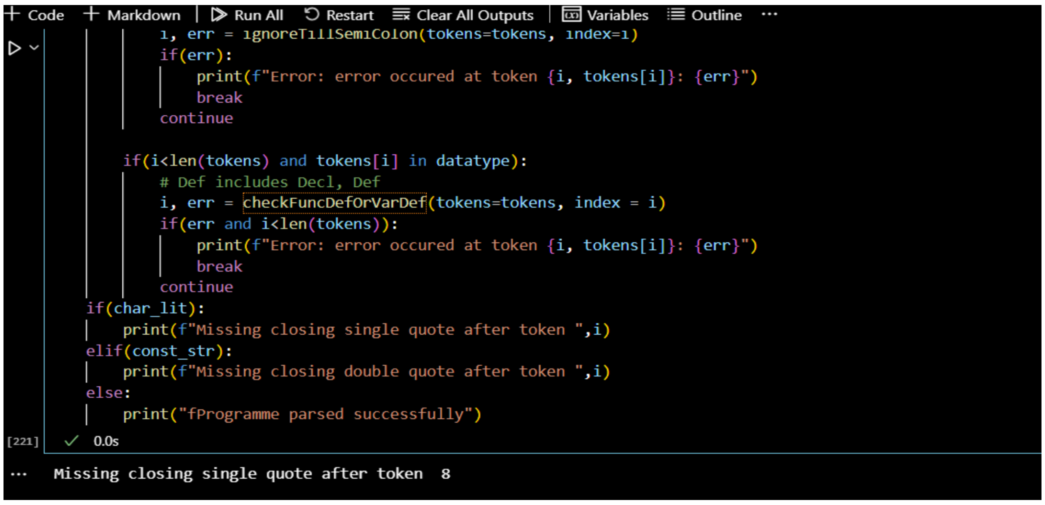
```
        if(i<len(tokens) and tokens[i] == 'using'):
            i, err = ignoreTillSemiColon(tokens=tokens, index=i)
            if(err):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue

        if(i<len(tokens) and tokens[i] in datatype):
            # Def includes Decl, Def
            i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
            if(err and i<len(tokens)):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue
    if(char_lit):
        print(f"Missing closing single quote after token ",i)
    elif(const_str):
        print(f"Missing closing double quote after token ",i)
    else:
        print("fProgramme parsed successfully")
[255] ✓ 0.0s

Error: error occured at token (25, 'std'): Expected { but found std
fProgramme parsed successfully
```

Expected test case 2: C++ code with missing closing single quote.



```cpp
int main() {
    char c = 'A; // Syntax error: missing closing single quote
    std::cout << "Character: " << c << std::endl;
    return 0;
}
```

Output:

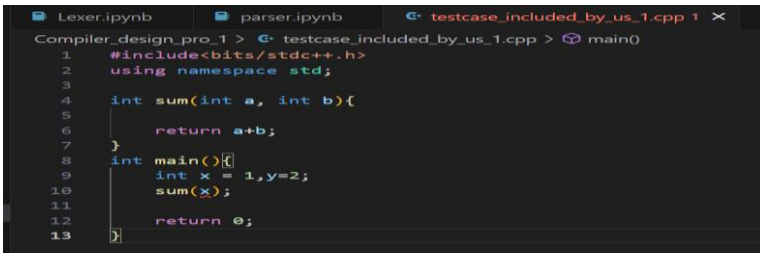


```python
        i, err = ignoreTillSemiColon(tokens=tokens, index=i)
        if(err):
            print(f"Error: error occured at token {i, tokens[i]}: {err}")
            break
        continue

    if(i<len(tokens) and tokens[i] in datatype):
        # Def includes Decl, Def
        i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
        if(err and i<len(tokens)):
            print(f"Error: error occured at token {i, tokens[i]}: {err}")
            break
        continue
if(char_lit):
    print(f"Missing closing single quote after token ",i)
elif(const_str):
    print(f"Missing closing double quote after token ",i)
else:
    print("fProgramme parsed successfully")
```

```
[221]  ✓  0.0s
```

```
Missing closing single quote after token  8
```

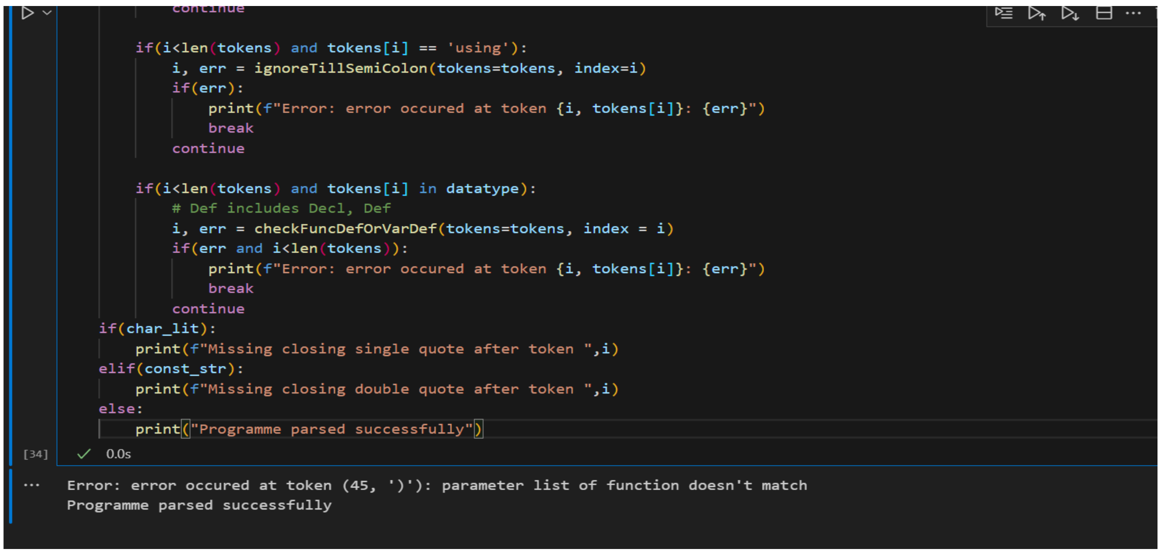
<u>**Test cases included by us 1**</u>: C++ code with incorrect parameters of function



Lexer.ipynb
parser.ipynb
testcase_included_by_us_1.cpp 1
Compiler_design_pro_1 > testcase_included_by_us_1.cpp > main()

```cpp
#include<bits/stdc++.h>
using namespace std;

int sum(int a, int b){

    return a+b;
}
int main(){
    int x = 1,y=2;
    sum(x);

    return 0;
}
```

<u>**Output:**</u>

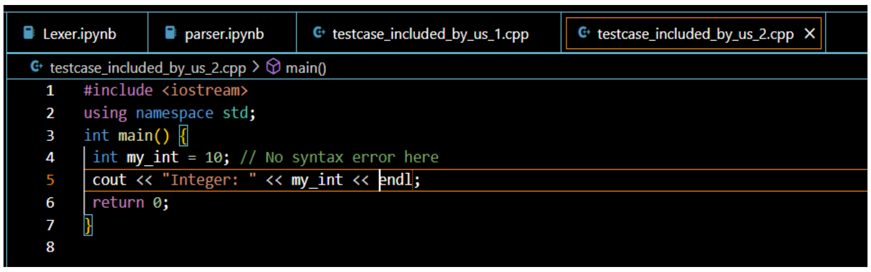


```python
        continue

    if(i<len(tokens) and tokens[i] == 'using'):
        i, err = ignoreTillSemiColon(tokens=tokens, index=i)
        if(err):
            print(f"Error: error occured at token {i, tokens[i]}: {err}")
            break
        continue

    if(i<len(tokens) and tokens[i] in datatype):
        # Def includes Decl, Def
        i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
        if(err and i<len(tokens)):
            print(f"Error: error occured at token {i, tokens[i]}: {err}")
            break
        continue
if(char_lit):
    print(f"Missing closing single quote after token ",i)
elif(const_str):
    print(f"Missing closing double quote after token ",i)
else:
    print("Programme parsed successfully")
```

```
[34]  ✓  0.0s
```

```
Error: error occured at token (45, ')'): parameter list of function doesn't match
Programme parsed successfully
```

Test cases included by us 2: C++ code with No syntax error.

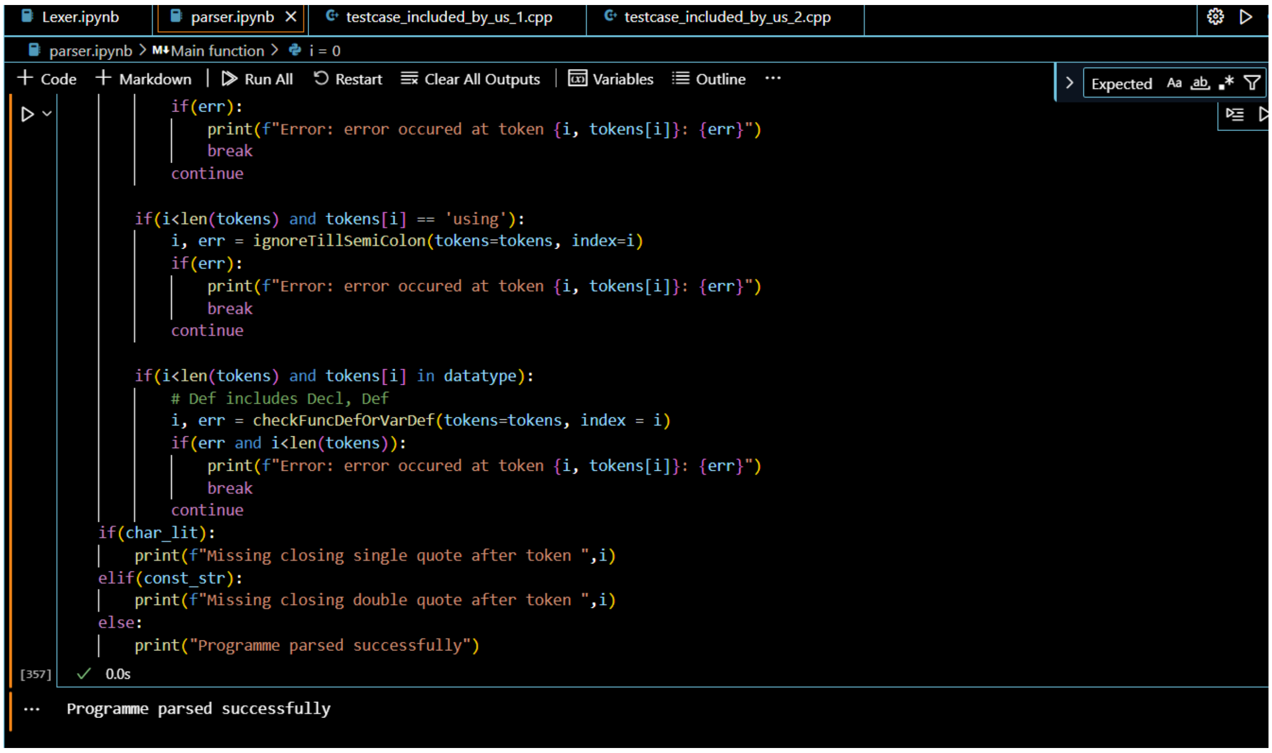


Lexer.ipynb | parser.ipynb | testcase_included_by_us_1.cpp | testcase_included_by_us_2.cpp

testcase_included_by_us_2.cpp > main()

```cpp
#include <iostream>
using namespace std;
int main() {
 int my_int = 10; // No syntax error here
 cout << "Integer: " << my_int << endl;
 return 0;
}

```

Output:



Lexer.ipynb | parser.ipynb | testcase_included_by_us_1.cpp | testcase_included_by_us_2.cpp

parser.ipynb > Main function > i = 0

```python
            if(err):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue

        if(i<len(tokens) and tokens[i] == 'using'):
            i, err = ignoreTillSemiColon(tokens=tokens, index=i)
            if(err):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue

        if(i<len(tokens) and tokens[i] in datatype):
            # Def includes Decl, Def
            i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
            if(err and i<len(tokens)):
                print(f"Error: error occured at token {i, tokens[i]}: {err}")
                break
            continue
    if(char_lit):
        print(f"Missing closing single quote after token ",i)
    elif(const_str):
        print(f"Missing closing double quote after token ",i)
    else:
        print("Programme parsed successfully")
```

[357] ✓ 0.0s

```
Programme parsed successfully
```