

Group 37

C++ Parser in Python

Dheeraj Kumar Singh (21114034)

Garvit Goyal (21114036)

Kulkarni Sourabh Shrinivasrao (21114053)

Problem Statement (PID = 16)

Develop a parser in Python language that accepts code in C++ and checks for syntax errors (Expected: loops, integers, if-else).

[Repository Link](#)

Description of The Code and Methods Used

The project consists of two jupyter notebook files

1. Lexer.ipynb
2. Parser.ipynb

Lexer.ipynb

This program parses the input C++ file, generating all the tokens from it. Subsequently, it stores these tokens in a list cache named "tokens".

Parser.ipynb

This file further processes the tokens generated by the lexer and simultaneously manages a symbol table to verify if any variable is undeclared within its scope. The primary loop initializes an iterator called "i," which begins at 0 and iterates through the tokens list.

In C++, a program is limited to containing variable declarations/definitions, function declarations/definitions in the global scope, as well as certain preprocessor directives and "using" statements. This structure aligns with the following BNF grammar:

Program \rightarrow function def | function decl | variable decl | variable def | preprocessor | using statement

Now, I combine function declaration/definition and variable declaration/definition into a function named:

1. **checkFuncDefOrVarDef**, which returns [index, err]:
 - This function checks for function or variable declaration or definition.

Another function checks for preprocessor directives:

2. **ignorePreprocessor**, which returns [index, err]:

- This function is called when a '#' is found as the first character in a token.
- It parses the preprocessor directives.

And one more function simply ignores the "using ..." statements:

3. **ignoreTillSemicolon**, which returns [index, err]:

- This function is called when the initial token is "using".
- It ignores all statements until encountering a semicolon.

Now, we have other functions that are called while parsing the function declaration/definition and variable declarations/definitions, namely:

4. **checkVarHeader**, which returns [index, err, dtype, ident]:

- dtype : datatype of the matched identifier
- ident : matched identifier
- This function matches the pattern <dtype> <ident> , for example, 'int a' or 'int main', etc.

5. **checkScope**, which returns [index, err]:

- This function handles everything inside curly braces '{' and '}', for example, function declaration scope, if scope, else scope, while scope, etc.
- It also takes the symbol table from the parent scope to check for all declared variables in the parent scope and updates the symbol table in its own scope.

6. **checkExpr**, which returns [index, err, dtype]:

- This function parses all expressions and checks for the datatype on which the operators work (for example, we cannot add an int and a string). It also checks if the operator is valid on the operand provided, considering the lvalue or rvalue property of the operand.

7. **checkLval**, which returns [index, err, dtype, islval]:

- This function parses the lvalue in an expression and returns if the value is an actual lvalue according to the specification of C++.

The lval and expr functions together parse the expression based on the grammar defined below:

$\text{expr} \rightarrow \text{lval operand expr} \mid \text{lval}$

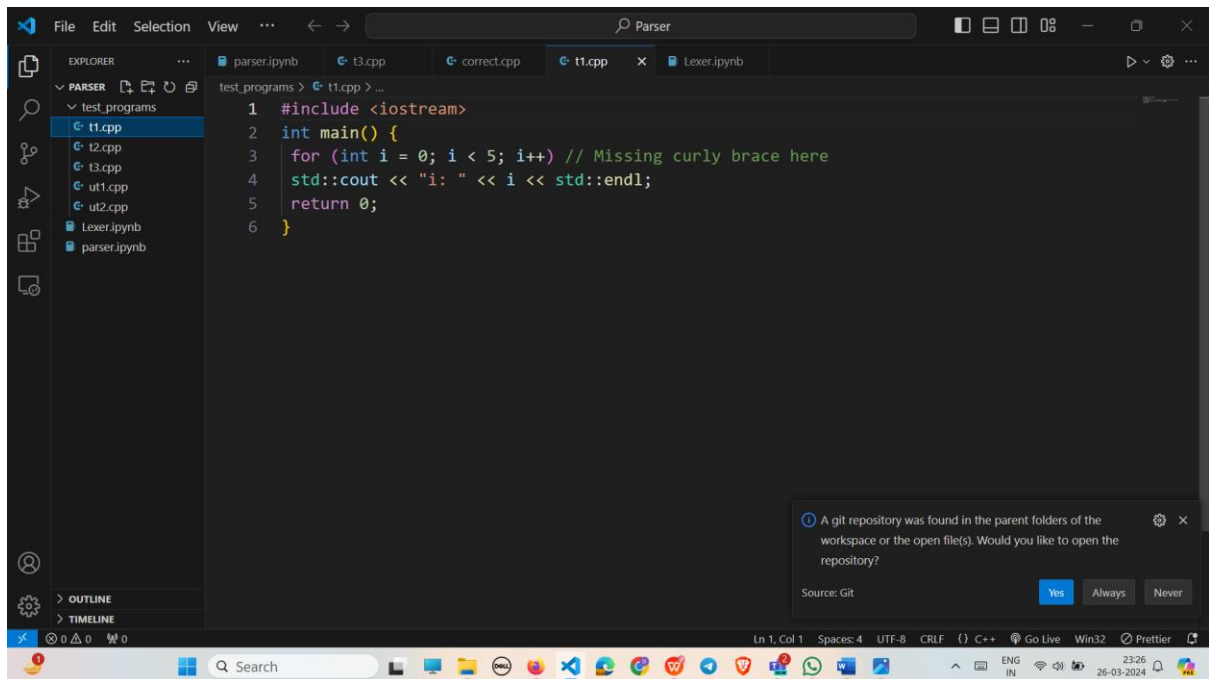
$\text{lval} \rightarrow \text{unary_op lval} \mid \text{identifier} \mid \text{identifier(param_list)} \mid \text{lval post_op} \mid \text{pre_op lval}$

8. **checkFuncDecOrValDef**, which returns [index, err, symbol_table]:

- This function parses only function declaration and variable declaration/definition, as function definition is not allowed inside any scope.

The detailed procedure for running the program is described in the readme file named “readme.md”.

Test Case 1:

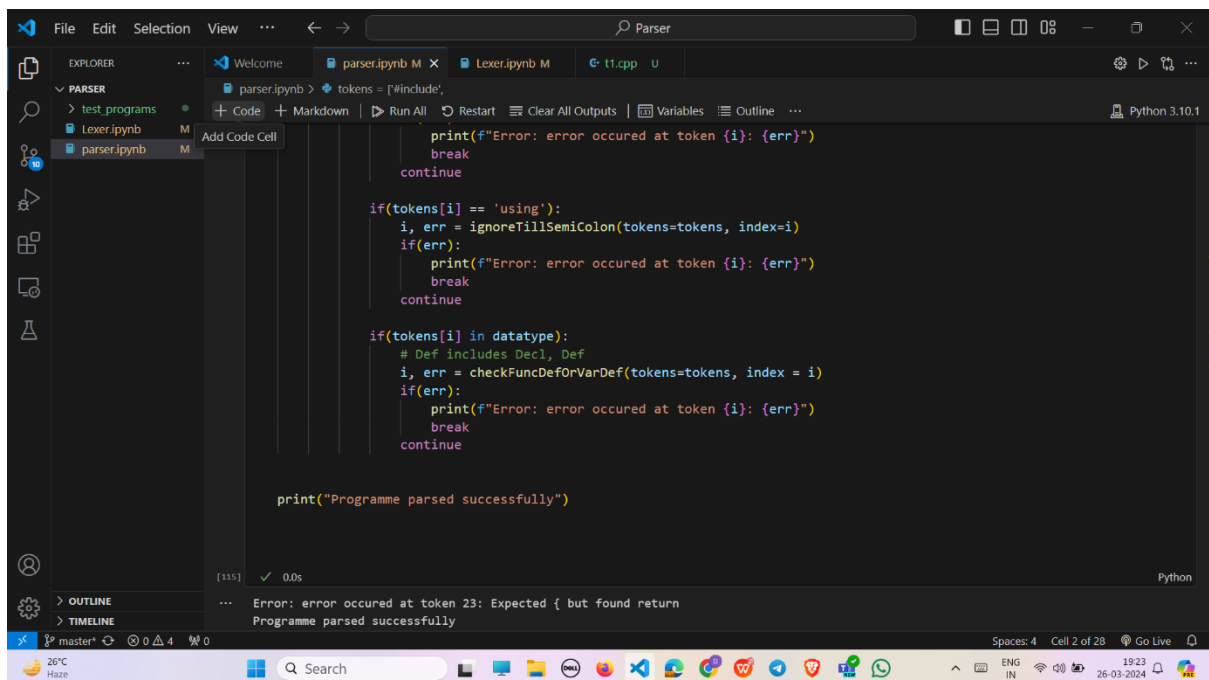


The screenshot shows the Visual Studio Code editor with a C++ file named `t1.cpp` open. The file contains the following code:

```
1 #include <iostream>
2 int main() {
3     for (int i = 0; i < 5; i++) // Missing curly brace here
4     std::cout << "i: " << i << std::endl;
5     return 0;
6 }
```

The code is syntactically incorrect due to a missing curly brace at the end of the `for` loop. A notification at the bottom right indicates that a git repository was found in the parent folders of the workspace or the open file(s).

Output:



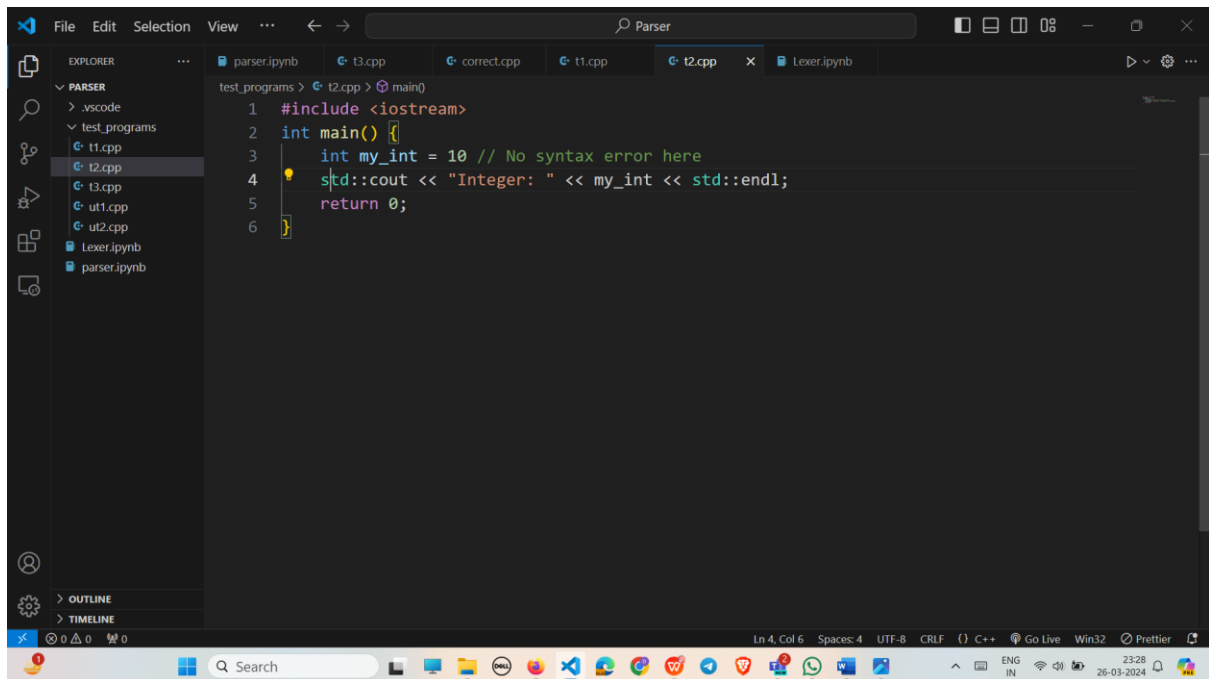
The screenshot shows the Visual Studio Code editor with a Python file named `parser.py` open. The file contains the following code:

```
1 tokens = ['include', 'using', 'return', 'std', 'cout', 'endl', 'int', 'main', 'for', 'int', 'i', '0', '5', 'i++', 'std', 'cout', 'endl', 'return', '0']
2
3 def parse(tokens):
4     i = 0
5     while i < len(tokens):
6         if(tokens[i] == 'include'):
7             i, err = ignoreTillSemicolon(tokens=tokens, index=i)
8             if(err):
9                 print(f"Error: error occurred at token {i}: {err}")
10                break
11                continue
12
13         if(tokens[i] == 'using'):
14             i, err = ignoreTillSemicolon(tokens=tokens, index=i)
15             if(err):
16                 print(f"Error: error occurred at token {i}: {err}")
17                 break
18                 continue
19
20         if(tokens[i] in datatype):
21             # Def includes Decl, Def
22             i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
23             if(err):
24                 print(f"Error: error occurred at token {i}: {err}")
25                 break
26                 continue
27
28     print("Programme parsed successfully")
29
30 if __name__ == '__main__':
31     parse(tokens)
```

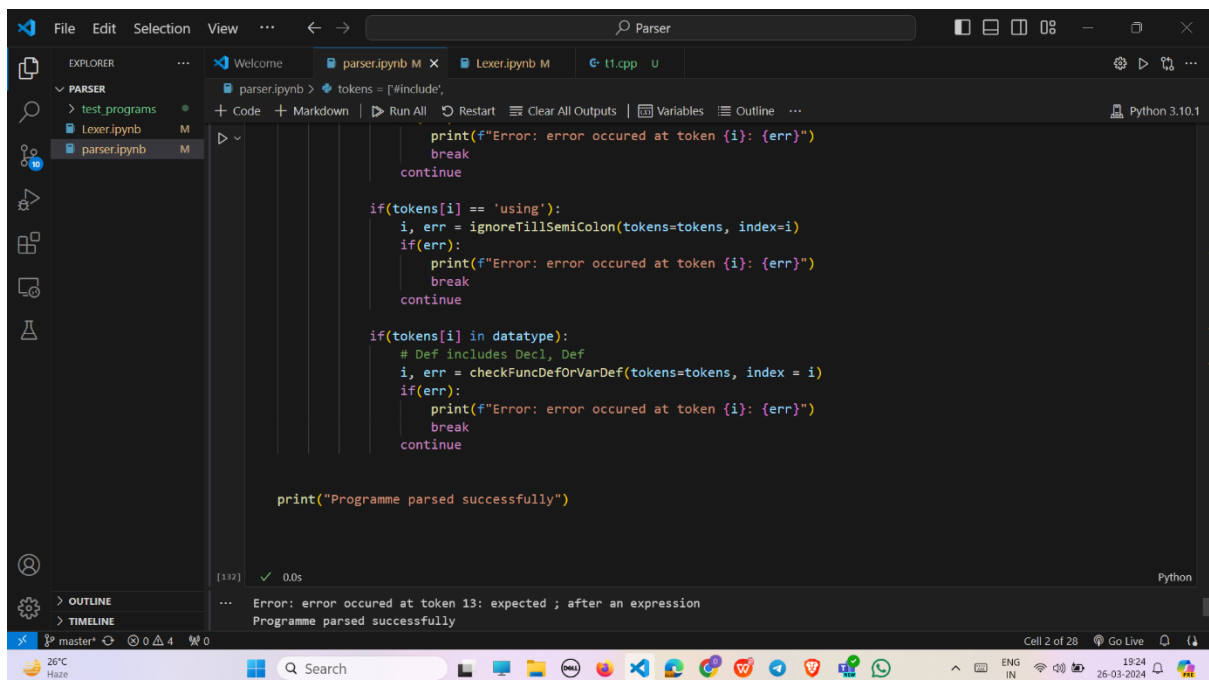
The code is syntactically correct. The output of the script is:

```
[115] ✓ 0.0s
Error: error occurred at token 23: Expected { but found return
Programme parsed successfully
```

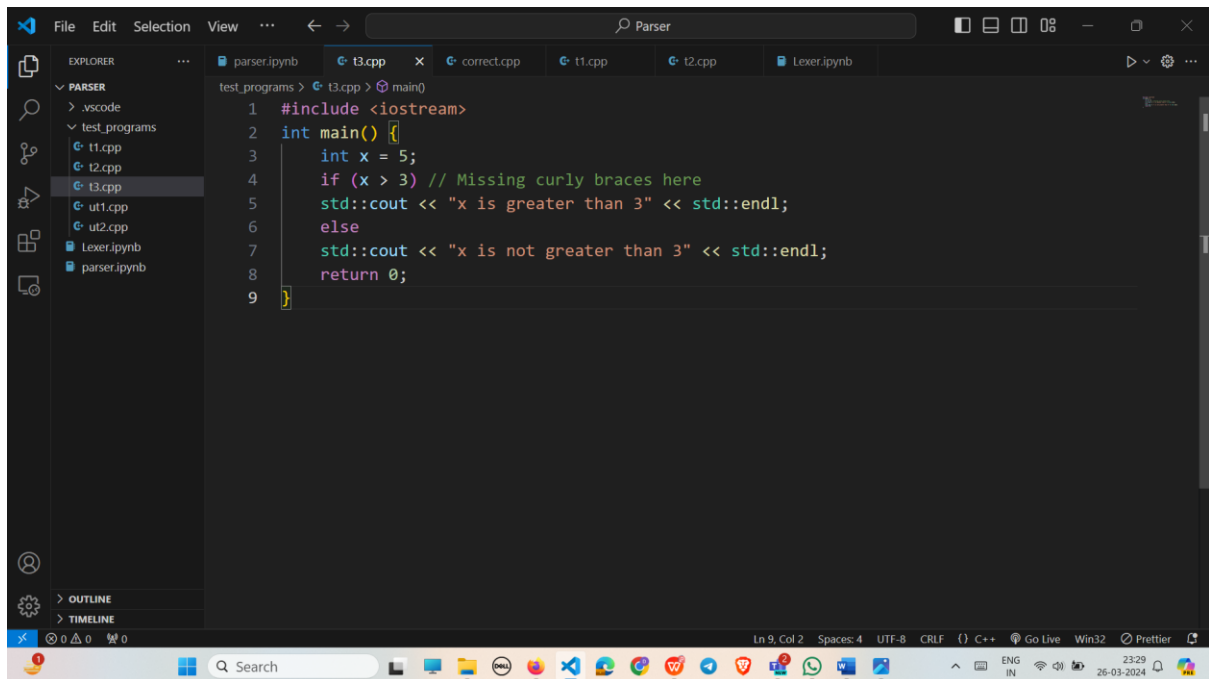
Test Case 2:



Output:



Test Case 3:

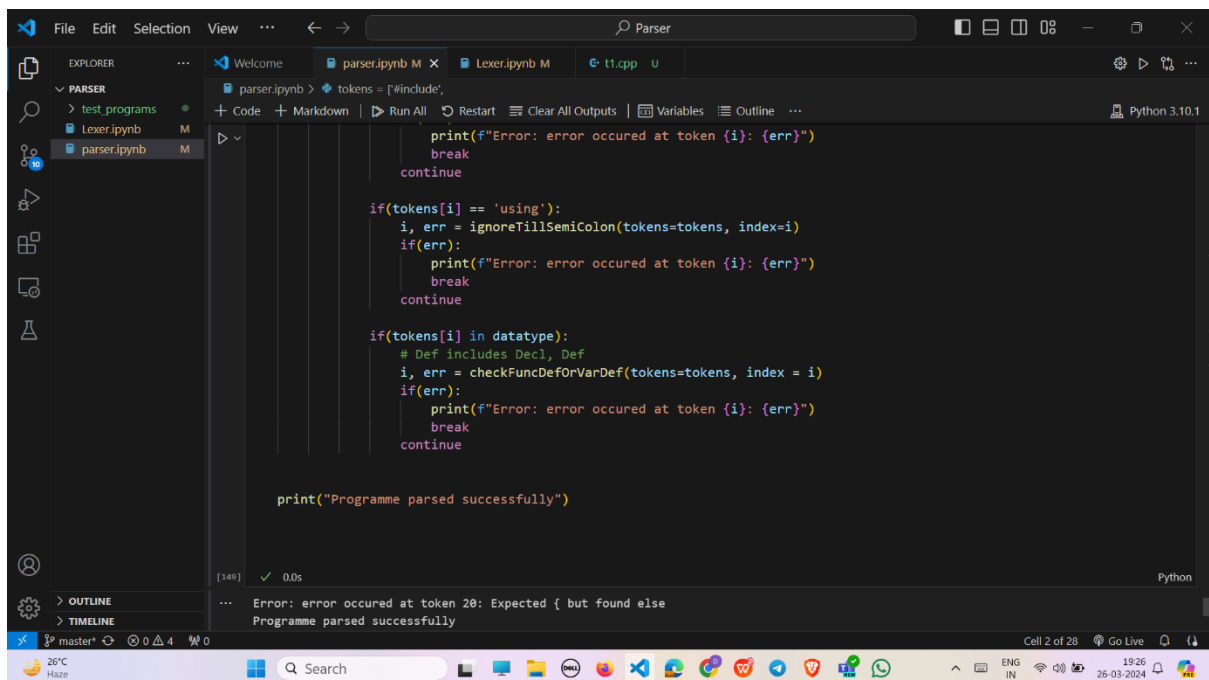


The screenshot shows the Visual Studio Code editor with a C++ file named `t3.cpp` open. The file contains the following code:

```
1 #include <iostream>
2 int main() {
3     int x = 5;
4     if (x > 3) // Missing curly braces here
5         std::cout << "x is greater than 3" << std::endl;
6     else
7         std::cout << "x is not greater than 3" << std::endl;
8     return 0;
9 }
```

The code is syntactically incorrect due to the missing closing curly brace for the `if` statement on line 4. The Explorer sidebar on the left shows the project structure, including `test_programs` and `parser.ipynb`. The status bar at the bottom indicates the current line and column as `Ln 9, Col 2`.

Output:



The screenshot shows the Visual Studio Code editor with a Python file named `parser.ipynb` open. The file contains the following code:

```
tokens = ['<include>']

print(f"Error: error occurred at token {i}: {err}")
break
continue

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

if(tokens[i] in datatype):
    # Def includes Decl, Def
    i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
    if(err):
        print(f"Error: error occurred at token {i}: {err}")
        break
        continue

print("Programme parsed successfully")
```

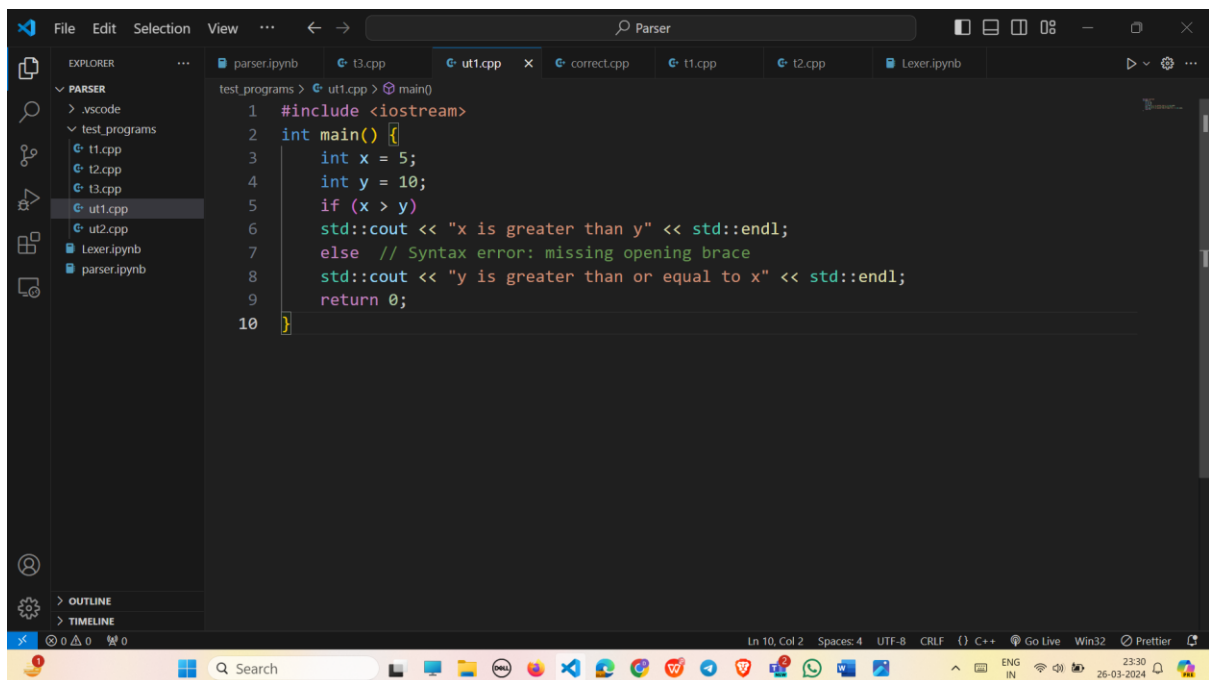
The code is a Python script that appears to be a parser for C++ tokens. It checks for various tokens and errors. The output of the script is shown in the console at the bottom:

```
[140] ✓ 0.0s
Error: error occurred at token 20: Expected { but found else
Programme parsed successfully
```

The status bar at the bottom indicates the current cell as `Cell 2 of 28` and the language as `Python`.

Test Cases for Evaluation:

Test Case 1:

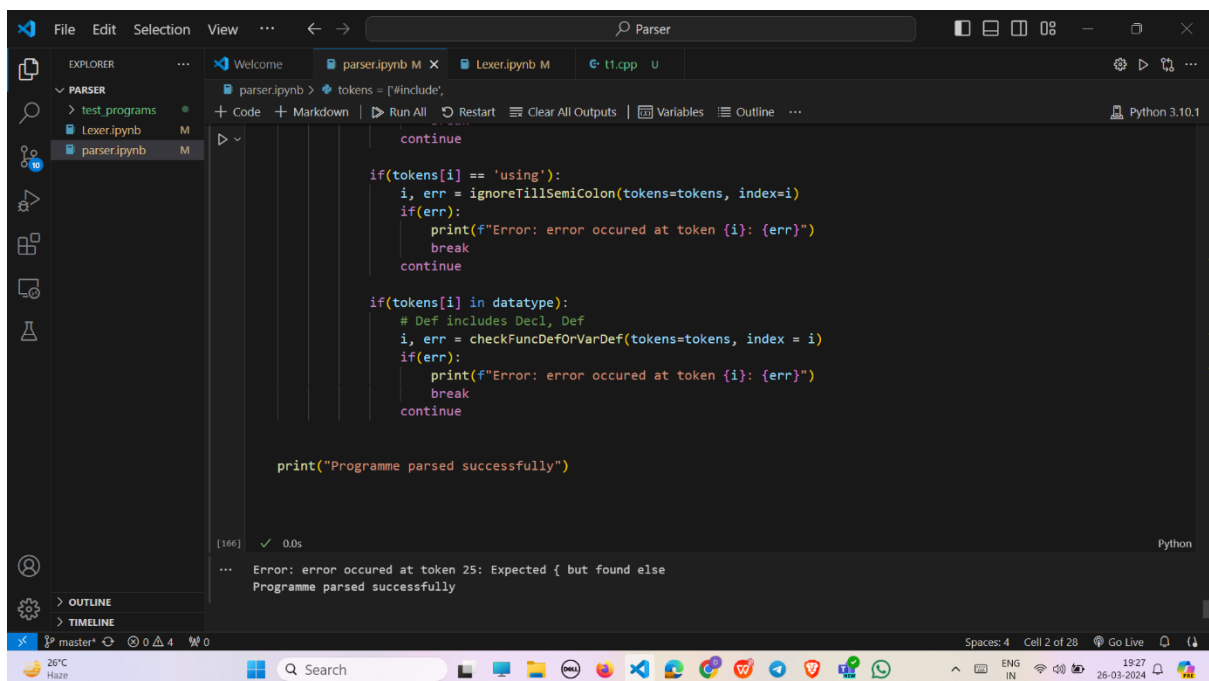


The screenshot shows a C++ code editor with the following code in `ut1.cpp`:

```
1 #include <iostream>
2 int main() {
3     int x = 5;
4     int y = 10;
5     if (x > y)
6         std::cout << "x is greater than y" << std::endl;
7     else // Syntax error: missing opening brace
8         std::cout << "y is greater than or equal to x" << std::endl;
9     return 0;
10 }
```

The code is intended to test a parser's ability to handle a syntax error (missing opening brace in the `if` statement). The program prints "x is greater than y" if the condition is true, and "y is greater than or equal to x" if the condition is false. The comment indicates a syntax error in the `else` branch.

Output:



The screenshot shows a Python code editor with the following code in `parser.py`:

```
tokens = ['#include;',
          'int', 'main()', '{',
          'int', 'x = 5;',
          'int', 'y = 10;',
          'if', '(', 'x >', 'y', ')',
          'std::cout <<', '"x is greater than y"', '<<', 'std::endl;',
          'else', // Syntax error: missing opening brace
          'std::cout <<', '"y is greater than or equal to x"', '<<', 'std::endl;',
          'return', '0;',
          '}']

def parse(tokens):
    i = 0
    while i < len(tokens):
        if tokens[i] == 'using':
            i, err = ignoreTillSemiColon(tokens=tokens, index=i)
            if err:
                print(f"Error: error occurred at token {i}: {err}")
                break
            continue

        if tokens[i] in datatype:
            # Def includes Decl, Def
            i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
            if err:
                print(f"Error: error occurred at token {i}: {err}")
                break
            continue

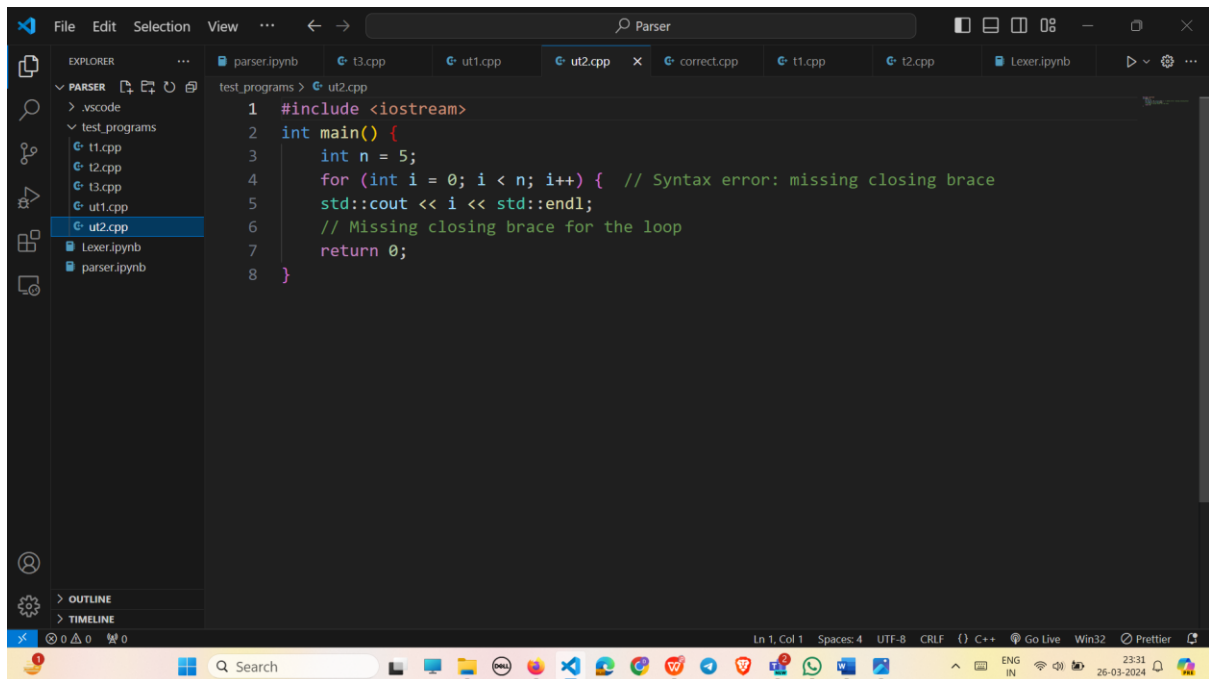
        print("Programme parsed successfully")

    print("Programme parsed successfully")

if __name__ == '__main__':
    parse(tokens)
```

The code is intended to test a parser's ability to handle a syntax error (missing opening brace in the `if` statement). The program prints "Programme parsed successfully" if the condition is true, and "Error: error occurred at token 25: Expected { but found else" if the condition is false. The comment indicates a syntax error in the `else` branch.

Test Case 2:

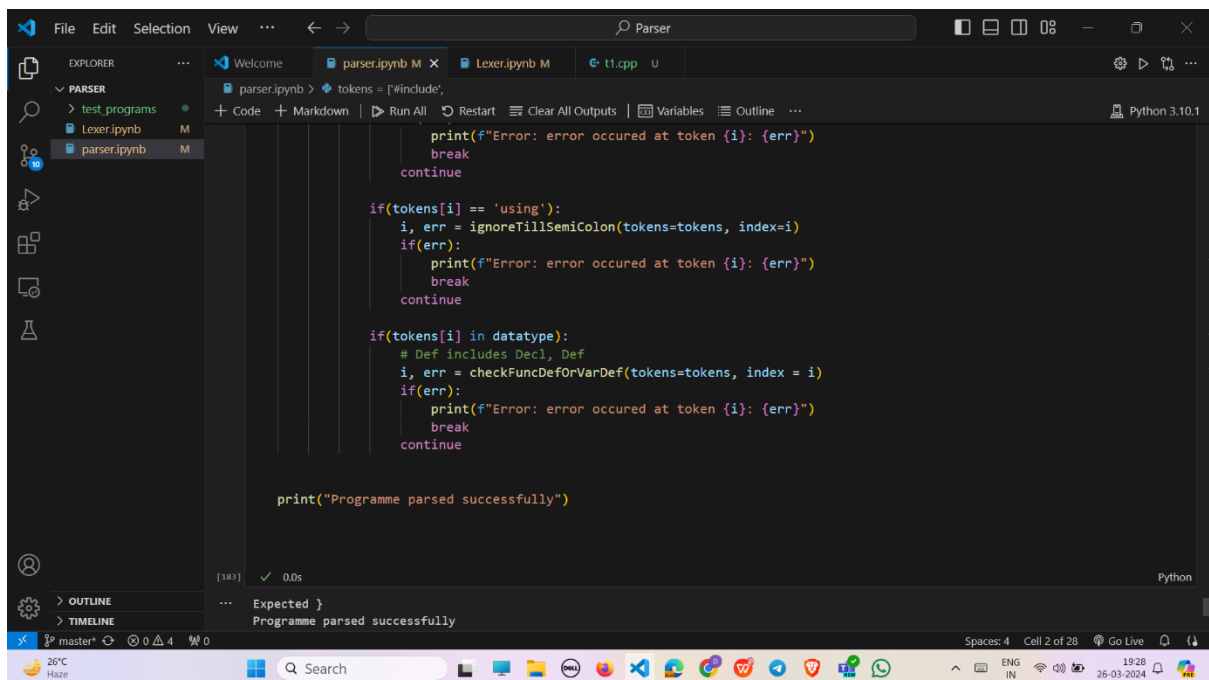


The screenshot shows a C++ code editor with a file named `ut2.cpp` open. The code is as follows:

```
1 #include <iostream>
2 int main() {
3     int n = 5;
4     for (int i = 0; i < n; i++) { // Syntax error: missing closing brace
5         std::cout << i << std::endl;
6         // Missing closing brace for the loop
7         return 0;
8     }
```

The editor highlights the missing closing brace for the `for` loop on line 4. The status bar at the bottom indicates the cursor is at line 1, column 1, with 4 spaces and UTF-8 encoding.

Output:



The screenshot shows a Python interpreter window with the following code executed:

```
tokens = ['<include>']
print(f"Error: error occurred at token {i}: {err}")
break
continue

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

if(tokens[i] in datatype):
    # Def includes Decl, Def
    i, err = checkFuncDefOrVarDef(tokens=tokens, index = i)
    if(err):
        print(f"Error: error occurred at token {i}: {err}")
        break
        continue

print("Programme parsed successfully")
```

The output shows the program was parsed successfully. The status bar at the bottom indicates the cursor is at line 183, column 4, with 0.0s execution time.