This code is a parser that reads mathematical expressions from a file and generates a parse tree for them. The expressions can be split across multiple lines and can contain white spaces between tokens.

The Node class defines a node in the parse tree. Each node has a symbol attribute that holds the symbol represented by the node, and leftChild and rightChild attributes that hold the left and right children of the node.

The merge function reads the contents of the file "d://lab3decoy.txt", removes white spaces and newlines, and writes the resulting string back to the file. The purpose of this function is to enable the parser to read expressions that are split across multiple lines.

The lexres function reads the contents of the file "d://lab3decoy.txt", removes white spaces and newlines, and appends each non-white space and non-newline character to a list. The first character in the list is assigned to the global variable next\_token, and the rest of the characters are written back to the file. The purpose of this function is to tokenize the input expression.

The unconsumed\_input function returns the remaining input that has not been consumed by the parser.

The lex function calls merge and lexres to tokenize the input expression, and sets the global variable next\_token to the first token.

The parser is implemented as a set of functions that implement the grammar rules for the arithmetic language.

The G function is the starting point for the parser. It calls lex to tokenize the input, and then calls the E function to parse the expression. If the entire input has been consumed and there were no errors during parsing, the function returns the parse tree for the expression. Otherwise, it prints an error message.

The E function calls the T function to parse the leftmost term in the expression, and then calls the R function to parse the rest of the expression, which can consist of one or more additive operators (+ or -) followed by terms.

The R function handles the additive operators and recursively calls itself to parse the remaining terms and operators. If there are no more additive operators, the function returns the parse tree for the expression.

The T function handles the multiplicative operators (\* or /) and calls the S function to parse the remaining factors and operators. If there are no more multiplicative operators, the function returns the parse tree for the expression.

The S function handles the multiplicative operators and recursively calls itself to parse the remaining factors and operators. If there are no more multiplicative operators, the function returns the parse tree for the expression.

The F function handles the atomic elements of the expression, which can be either variables (M) or integers (N) or subexpressions enclosed in parentheses.

The M function parses a single variable and returns a leaf node in the parse tree.

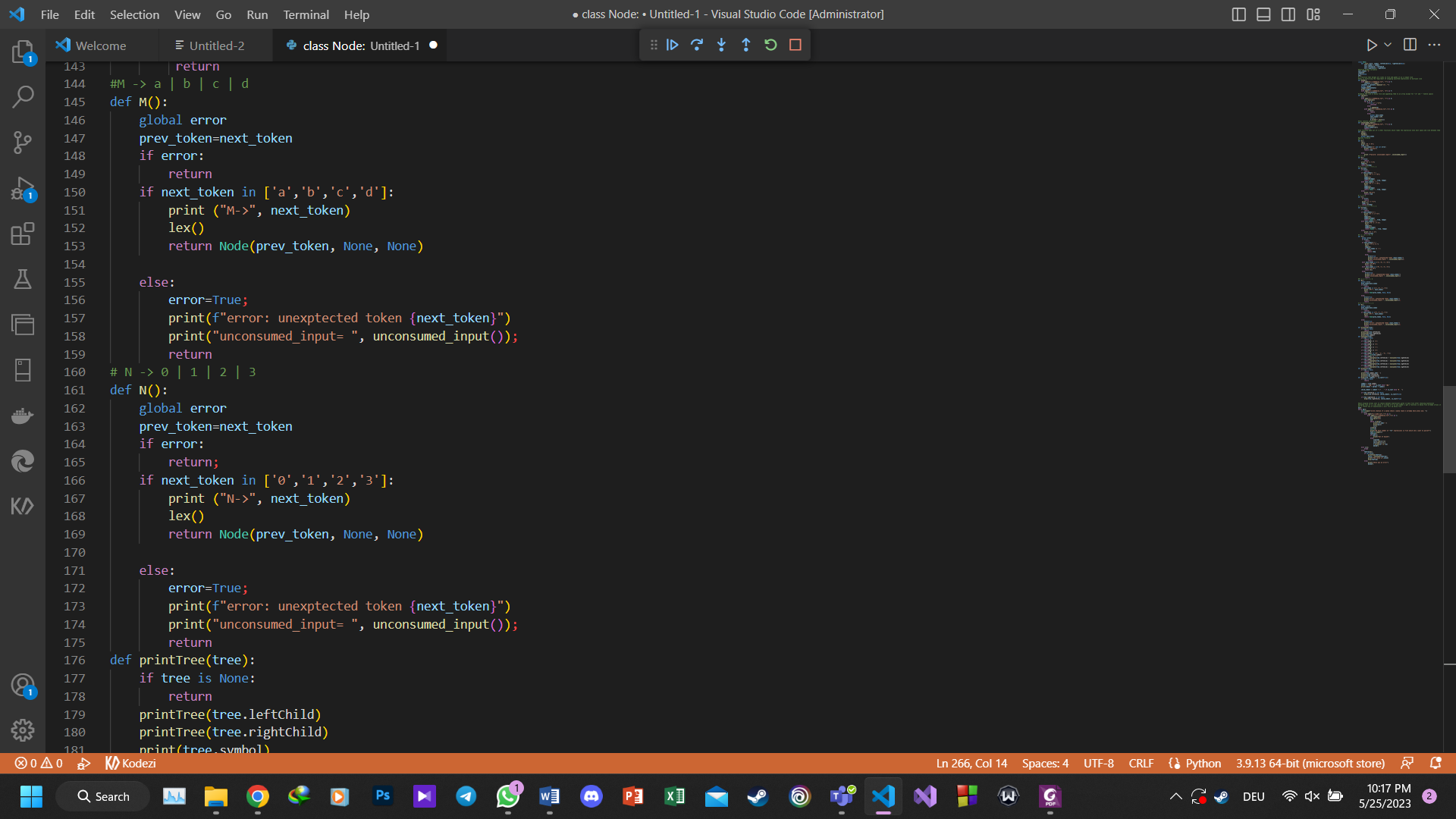
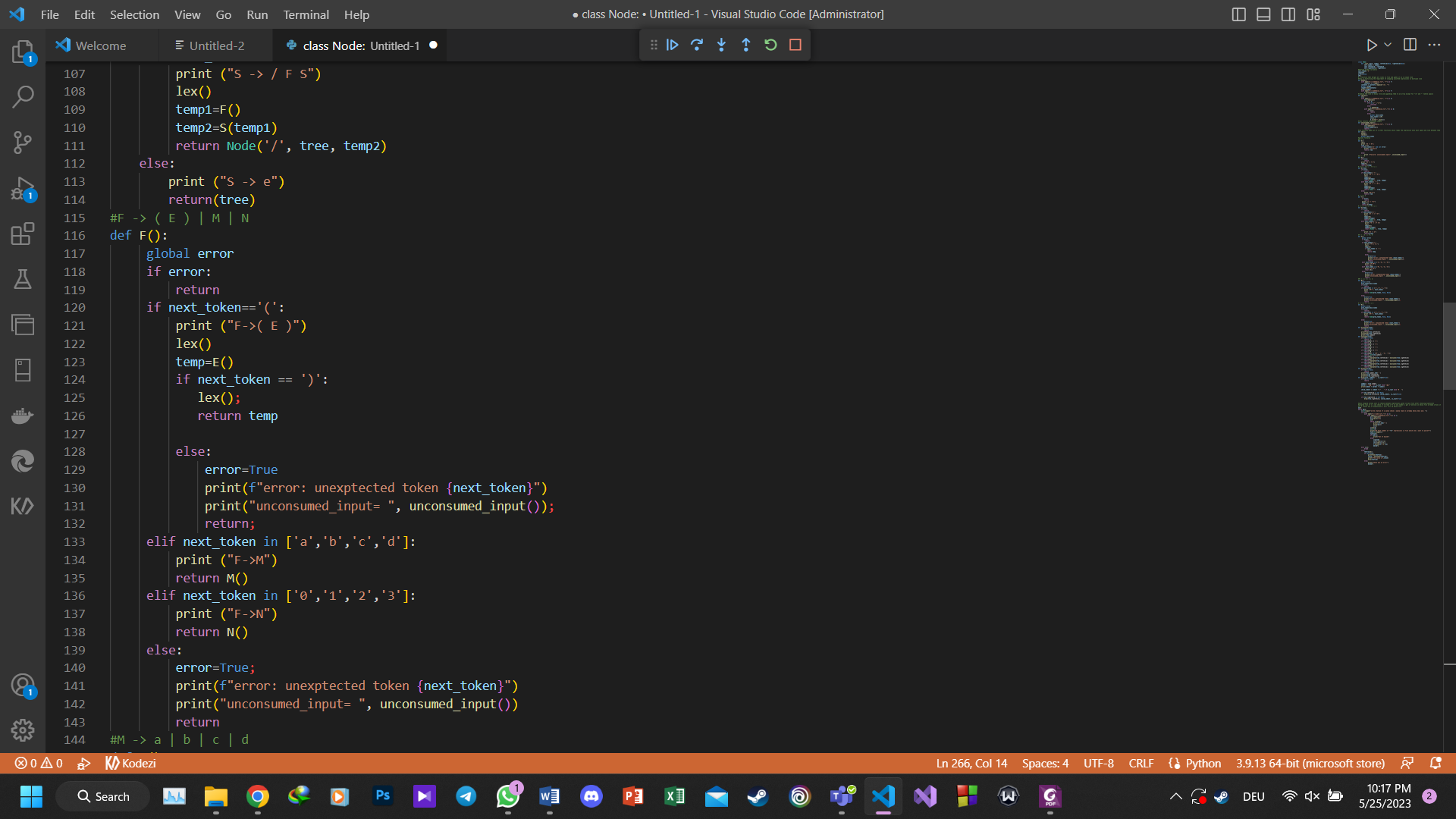
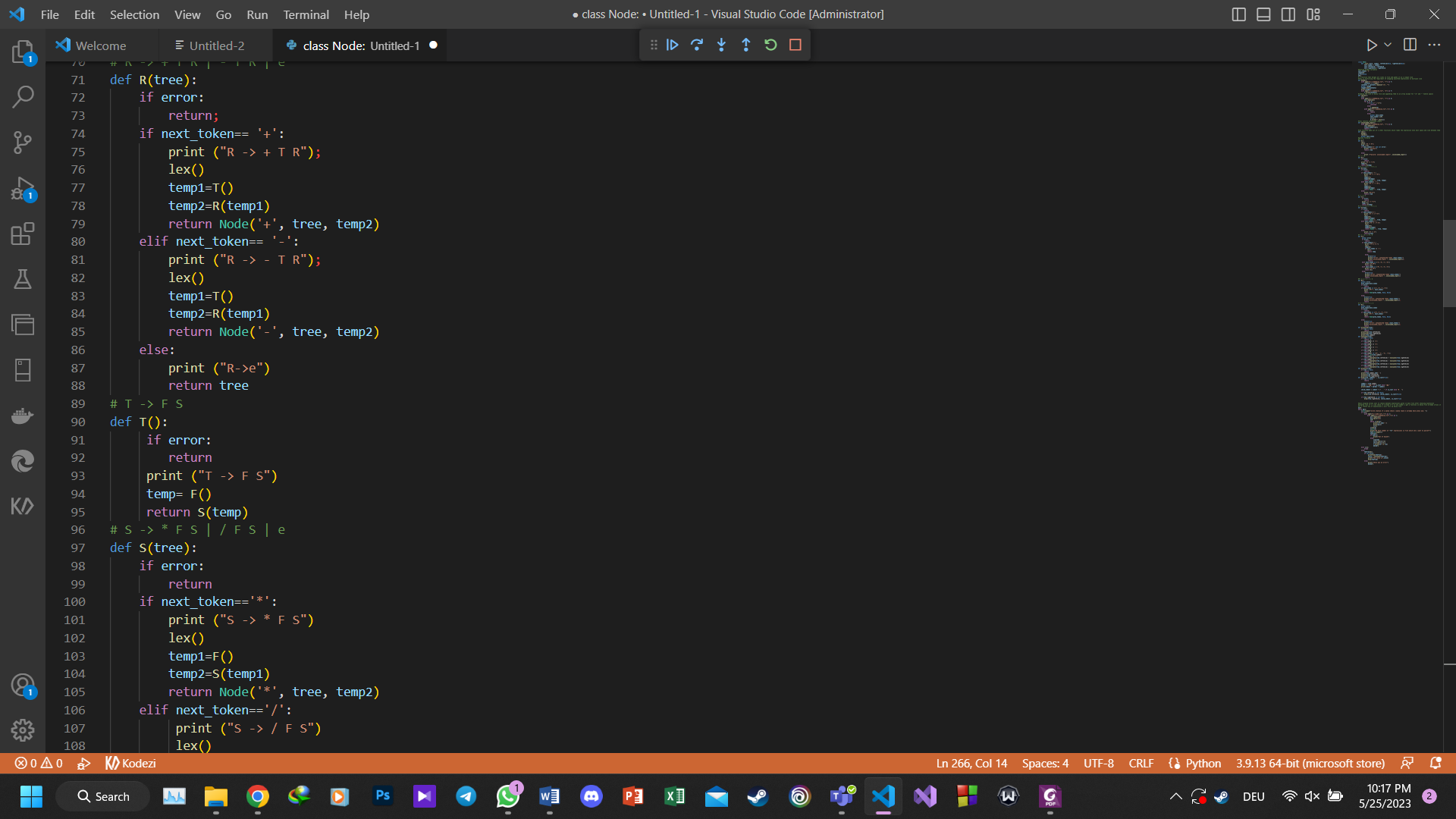
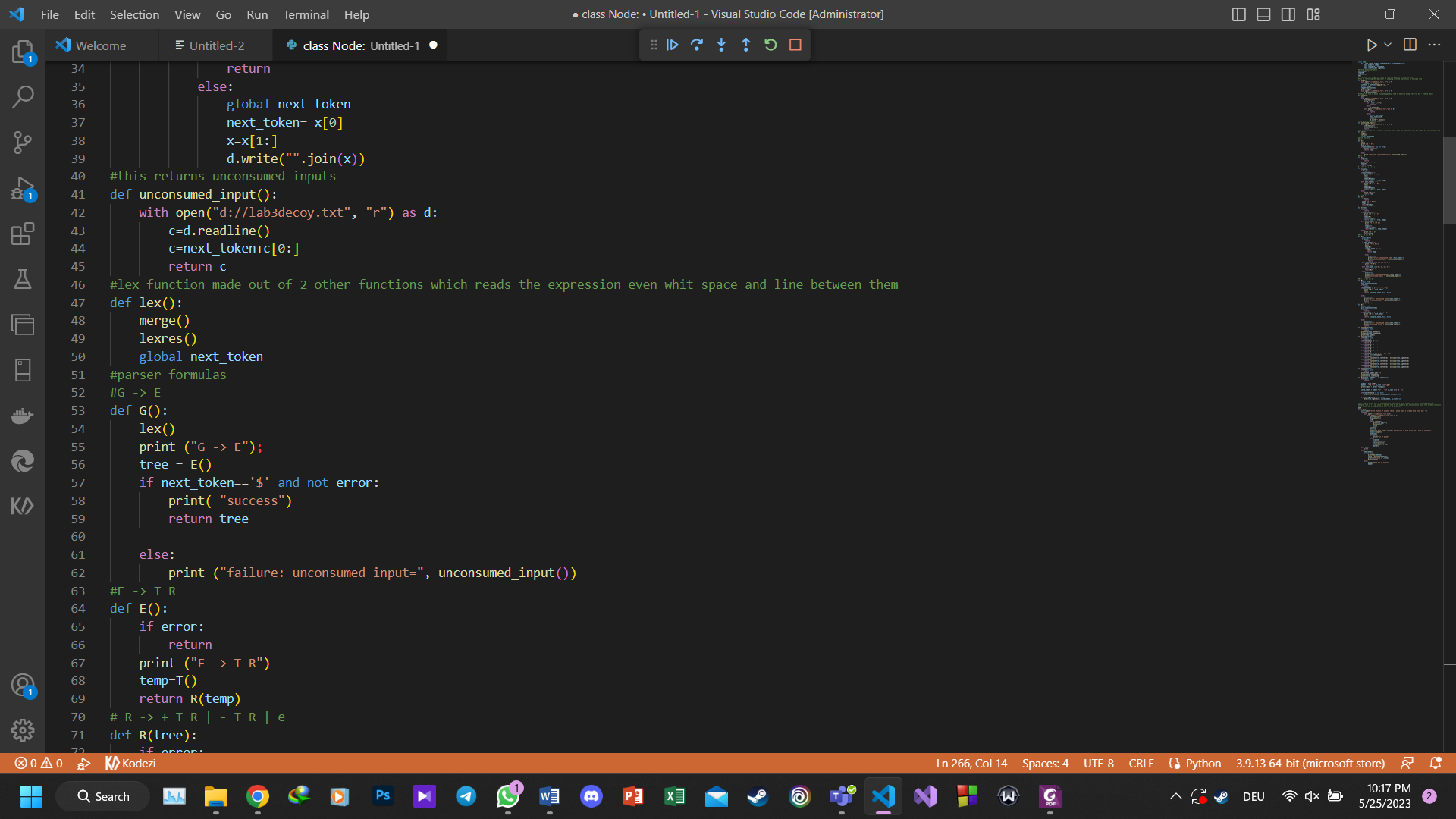
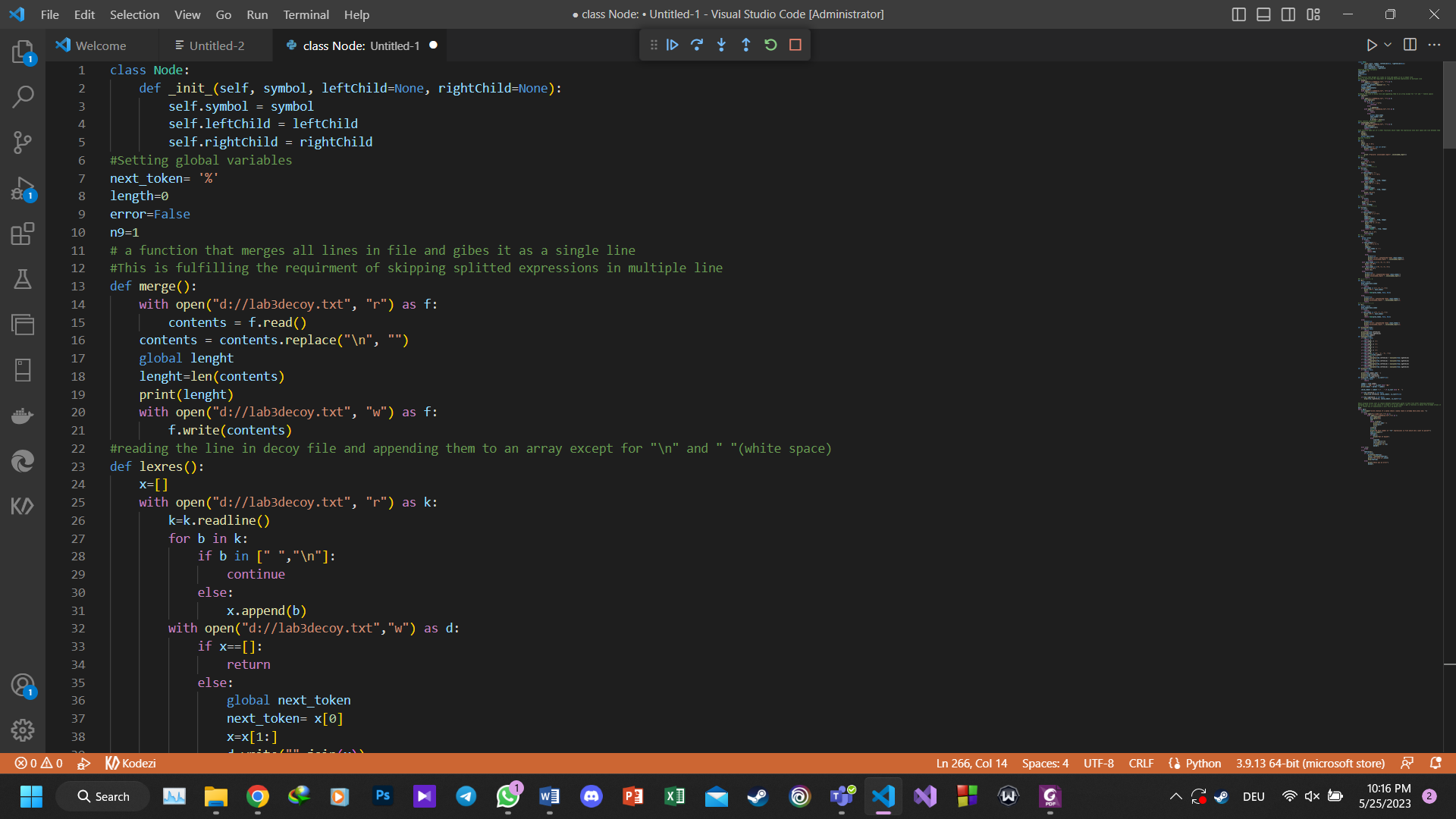
The N function parses a single digit and returns a leaf node in the parse tree.

The printTree function recursively prints the parse tree in postfix notation.

The evaluate function recursively evaluates the parse tree and returns the numerical value of the expression.

The draw function recursively prints the parse tree in a tree-like structure, using indentation to show the hierarchy of nodes.

The main program allows the user to choose an expression from a file and parse it using the G function. If the parsing is successful, the program prints the parse tree and its value. If there are errors during parsing, the program prints an error message. The program continues to prompt the user until they choose to exit.



A picture containing text, screenshot

Description automatically generated

A screenshot of a computer program

Description automatically generated with medium confidence

When running the program after choosing the expression we can see the output of HW3 parser and in addition the expression is read and printed in post order traverse, the value of expression is printed and the tree will be drawn.

A picture containing text, screenshot, software, multimedia software

Description automatically generated

A picture containing text, screenshot, multimedia software, software

Description automatically generated

**5 samples that run correctly:**

**1.(a+b)\*(c-d)/2$:**

****

**2.** **3\*a+b\*c$:**

**A picture containing multimedia software, software, graphics software, text

Description automatically generated**

**3.** **1-3/2$:A picture containing screenshot, text, software, multimedia software

Description automatically generated**

**4.** **(0\*1)/1$**

**A picture containing screenshot, text, graphics software, multimedia software

Description automatically generated**

**5.** **3/(2\*(1+1))$:** A picture containing screenshot, text, multimedia software, graphics software

Description automatically generated

**5 samples that produce errors:**

**1.** **4/(2\*(1+1))$:(4 is not defined in parser)**

**A picture containing text, screenshot, font

Description automatically generated**

**2. 2\*(a+b$:(no closing paranthesis)**

**A picture containing text, screenshot, multimedia software, graphics software

Description automatically generated**

**3. 2++3/6$:(two operands in one place)**

A black screen with white text

Description automatically generated with low confidence

**4. 34+43$:(double digits not accepted)**

A black screen with white text

Description automatically generated with low confidence

**5.** **4+3/3:(no ’$’ gives error in parser)**

A picture containing text, screenshot, font, software

Description automatically generated