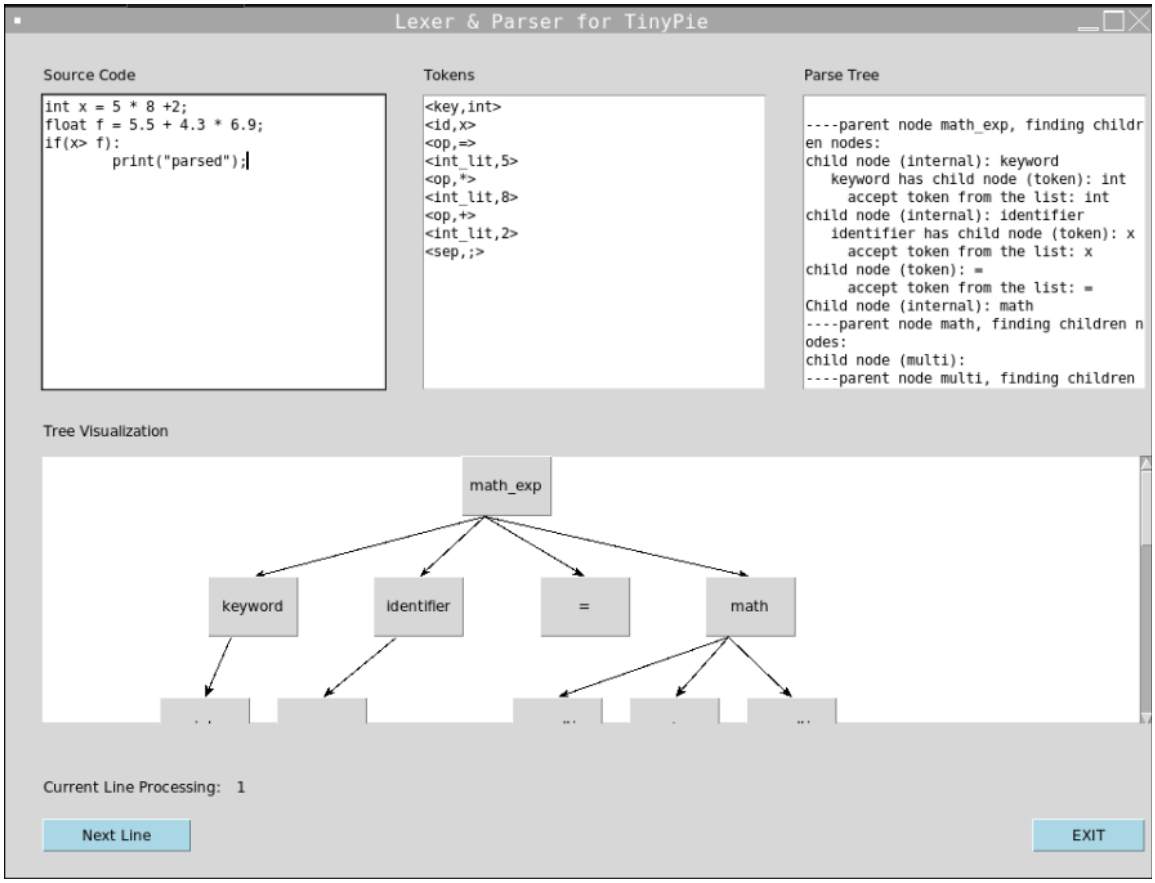
Lexer & Parser Visualizer for TinyPie

Name	Contribution	Contribution Percentage
Jacob Almon	-Implemented tree data structure using 'anytree' module for parse trees -Debugged code -Implemented Parse Tree wrapper -Question 2	40%
Hannah Gonzalez	-Made GUI presentable -**Bugged the code that Jacob debugged** -Added TreeViewer Frame and helper functions -Questions 1-3, 4, 5	35%
Natalie Pedroza	-Added features to the GUI -Created the viewer for makeWidgets function -Debugged draw levels code -Question 6	25%

The tree data structure was implemented using the Python module *anytree* which we found on Google for creating binary trees. We needed two helper classes to make the *anytree* work in a GUI. The TreeViewer class creates the canvas and all of its features to draw the tree. We also needed TreeViewer to inherit the Frame class to execute this. Lastly, we needed to implement a ParseTreeWrapper class as a helper class to implement the functions within the TreeViewer class to help GUI functionality to *anytree*. The actual implementation was done in each of the parsing functions using features from the previous helper classes. We utilized some of the code from a source our professor provided us to help us implement these helper classes, the difficult part was translating Python 2 code into Python 3, which are somewhat similar, but use features that aren't compatible with Python 3, so we had to work around that. Jacob worked to debug the code Hannah and Natalie implemented for the TreeViewer class.

The visualizer we used is called PyTree from the Tkinter GUI library and it sketches out nodes of a tree on the screen as boxes connected by arrows. This is done by "wrapping" real trees in interface objects which would come from our parser. Also referencing the link provided, ChatGPT, and StackExchange we figured out how to update the functions for Python 3. The link and Google searches were handy because they provided details with comments of what each function was doing and what the binary search tree looked like.

Here are some screenshots of our project for the math, if, and print expressions.



Lexer & Parser for TinyPie

Source Code

```
int x = 5 * 8 +2;
float f = 5.5 + 4.3 * 6.9;
if(x> f):
        print("parsed");
```

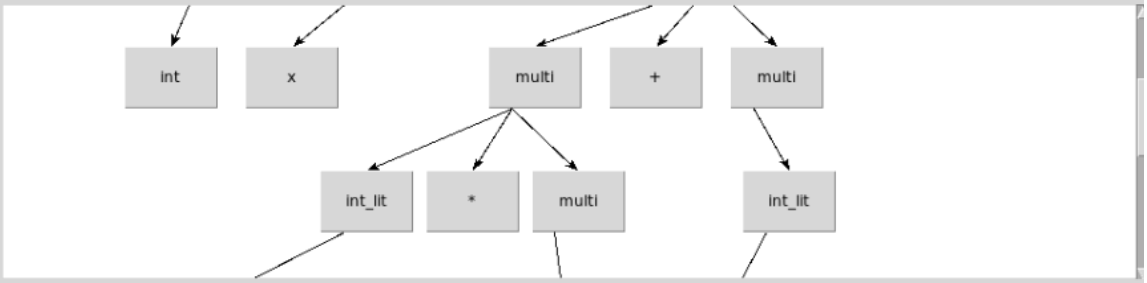
Tokens

```
<key,int>
<id,x>
<op,=>
<int_lit,5>
<op,*>
<int_lit,8>
<op,+>
<int_lit,2>
<sep,;>
```

Parse Tree

```
----parent node math_exp, finding childr
en nodes:
child node (internal): keyword
   keyword has child node (token): int
     accept token from the list: int
child node (internal): identifier
   identifier has child node (token): x
     accept token from the list: x
child node (token): =
     accept token from the list: =
Child node (internal): math
----parent node math, finding children n
odes:
child node (multi):
----parent node multi, finding children
```

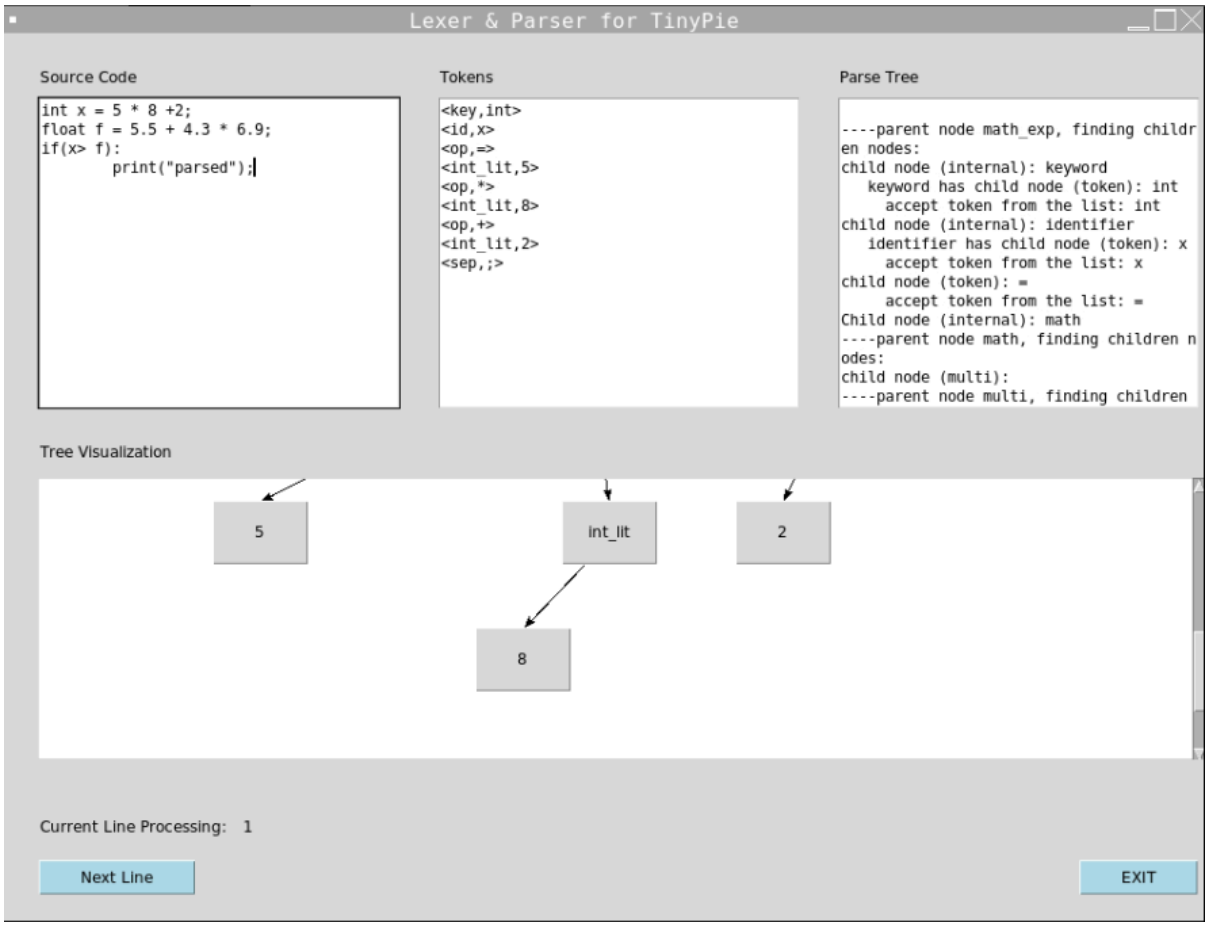
Tree Visualization

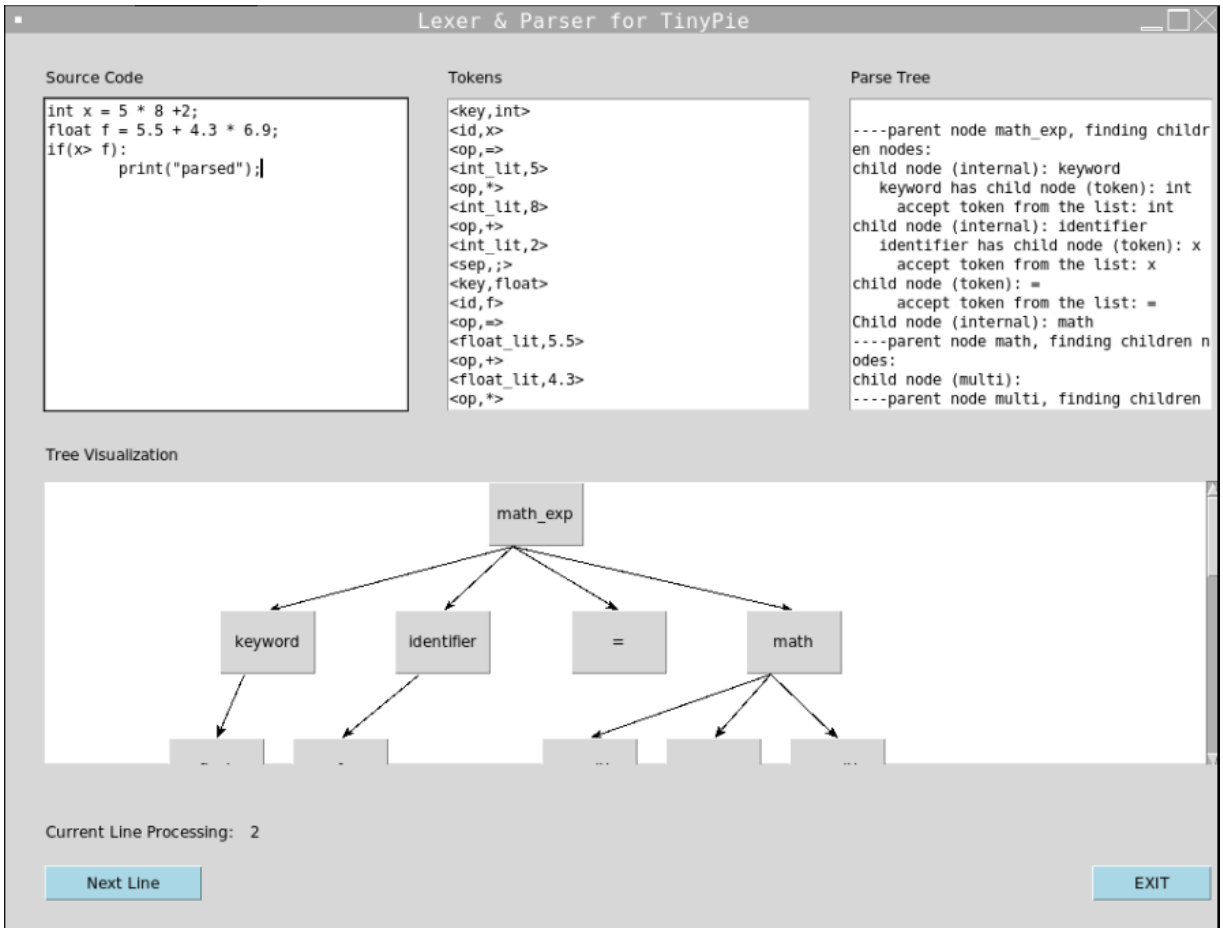


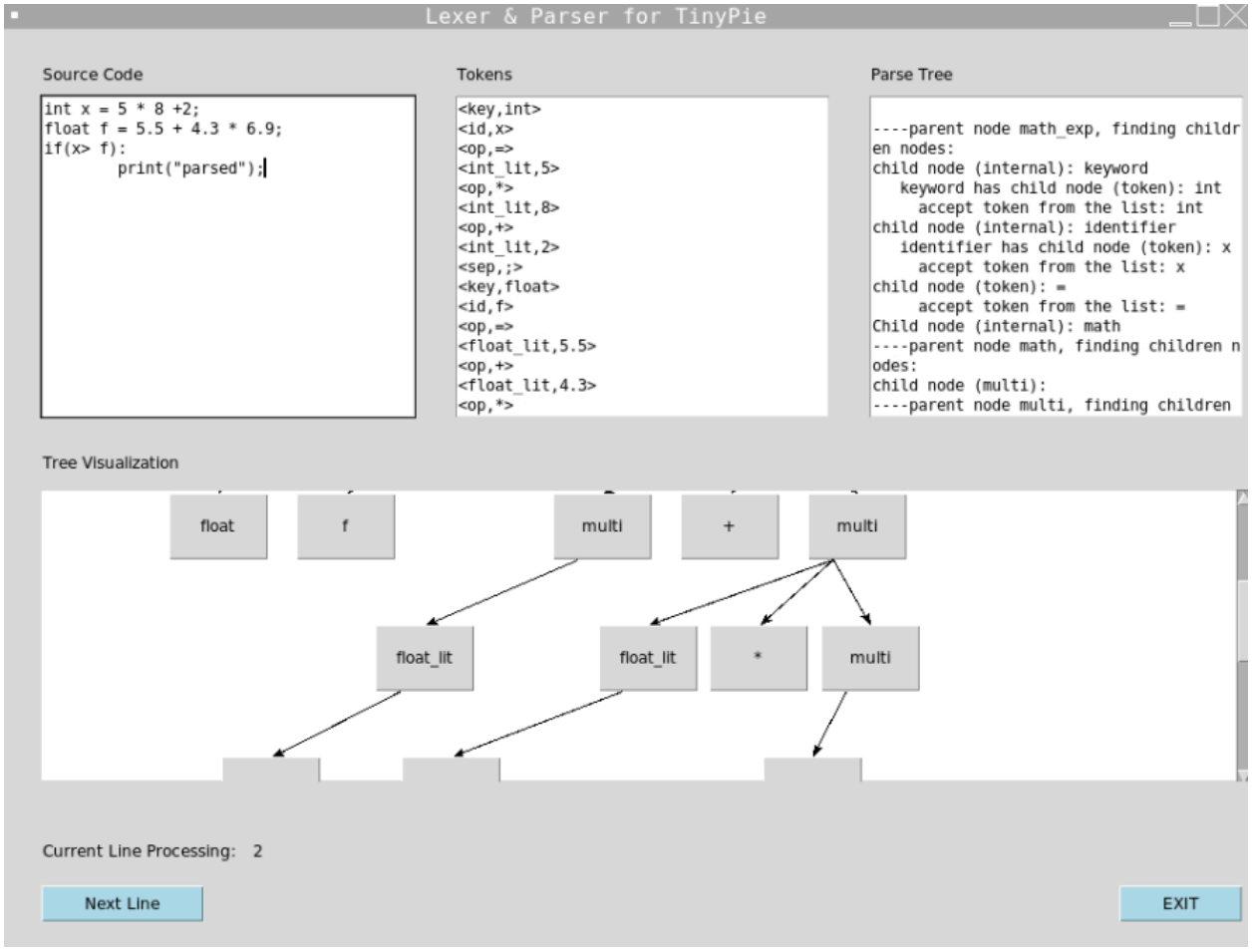
Current Line Processing: 1

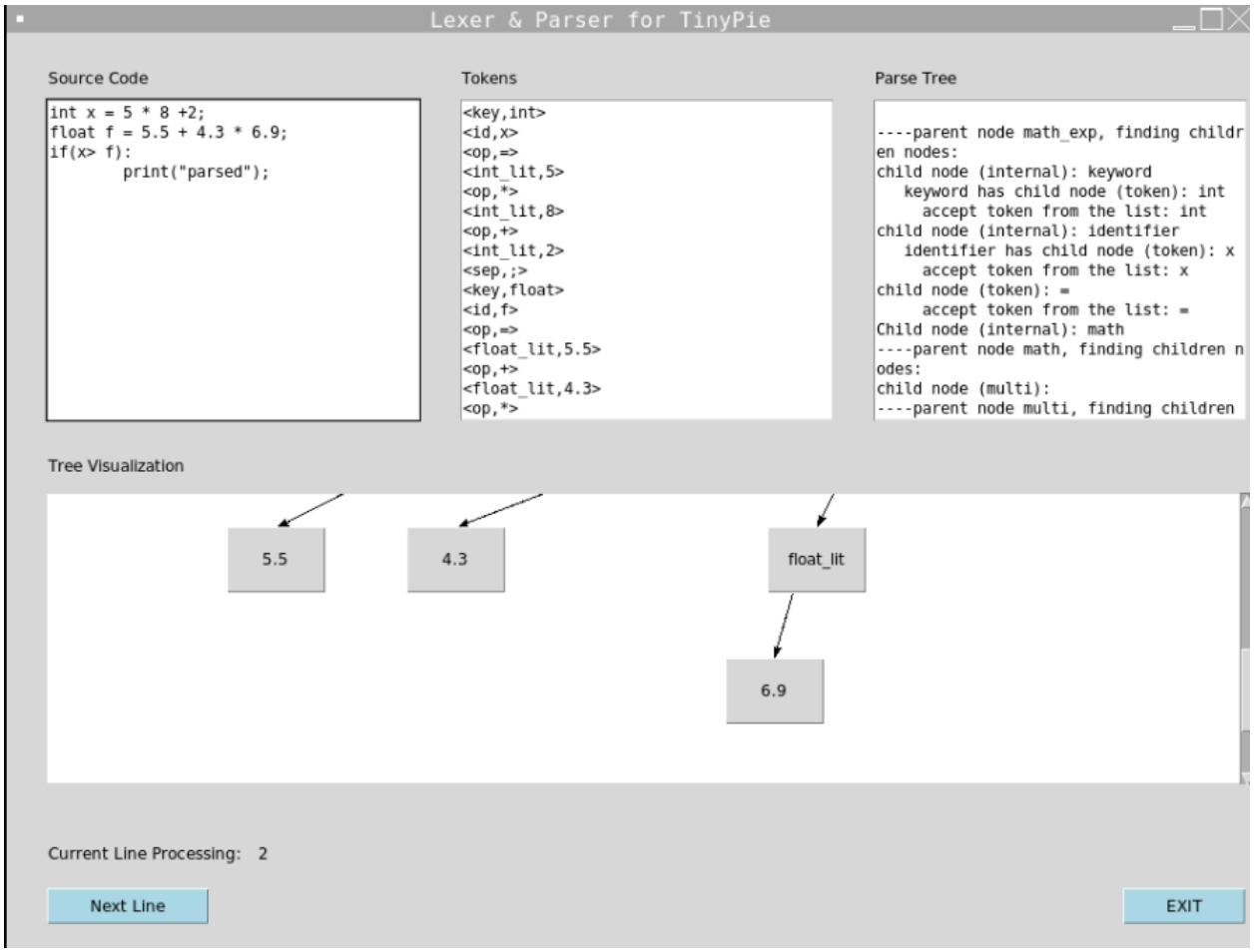
Next Line

EXIT


Lexer & Parser for TinyPie
Source Code
int x = 5 * 8 +2;
float f = 5.5 + 4.3 * 6.9;
if(x> f):
print("parsed");
Tokens
<key,int>
<id,x>
<op,=>
<int_lit,5>
<op,*>
<int_lit,8>
<op,+>
<int_lit,2>
<sep,;>
Parse Tree
----parent node math_exp, finding childr
en nodes:
child node (internal): keyword
keyword has child node (token): int
accept token from the list: int
child node (internal): identifier
identifier has child node (token): x
accept token from the list: x
child node (token): =
accept token from the list: =
Child node (internal): math
----parent node math, finding children n
odes:
child node (multi):
----parent node multi, finding children
Tree Visualization
5
int_lit
2
8
Current Line Processing: 1
Next Line
EXIT


Lexer & Parser for TinyPie
Source Code
int x = 5 * 8 +2;
float f = 5.5 + 4.3 * 6.9;
if(x> f):
print("parsed");
Tokens
<key,int>
<id,x>
<op,=>
<int_lit,5>
<op,*>
<int_lit,8>
<op,+>
<int_lit,2>
<sep,;>
<key,float>
<id,f>
<op,=>
<float_lit,5.5>
<op,+>
<float_lit,4.3>
<op,*>
Parse Tree
----parent node math_exp, finding childr
en nodes:
child node (internal): keyword
keyword has child node (token): int
accept token from the list: int
child node (internal): identifier
identifier has child node (token): x
accept token from the list: x
child node (token): =
accept token from the list: =
Child node (internal): math
----parent node math, finding children n
odes:
child node (multi):
----parent node multi, finding children
Tree Visualization
math_exp
keyword
identifier
=
math
Current Line Processing: 2
Next Line
EXIT


Lexer & Parser for TinyPie
Source Code
int x = 5 * 8 +2;
float f = 5.5 + 4.3 * 6.9;
if(x> f):
print("parsed");
Tokens
<key,int>
<id,x>
<op,=>
<int_lit,5>
<op,*>
<int_lit,8>
<op,+>
<int_lit,2>
<sep,;>
<key,float>
<id,f>
<op,=>
<float_lit,5.5>
<op,+>
<float_lit,4.3>
<op,*>
Parse Tree
----parent node math_exp, finding childr
en nodes:
child node (internal): keyword
keyword has child node (token): int
accept token from the list: int
child node (internal): identifier
identifier has child node (token): x
accept token from the list: x
child node (token): =
accept token from the list: =
Child node (internal): math
----parent node math, finding children n
odes:
child node (multi):
----parent node multi, finding children
Tree Visualization
float
f
multi
+
multi
float_lit
float_lit
*
multi
Current Line Processing: 2
Next Line
EXIT


Lexer & Parser for TinyPie
Source Code
int x = 5 * 8 +2;
float f = 5.5 + 4.3 * 6.9;
if(x> f):
print("parsed");
Tokens
<key,int>
<id,x>
<op,=>
<int_lit,5>
<op,*>
<int_lit,8>
<op,+>
<int_lit,2>
<sep,;>
<key,float>
<id,f>
<op,=>
<float_lit,5.5>
<op,+>
<float_lit,4.3>
<op,*>
Parse Tree
----parent node math_exp, finding childr
en nodes:
child node (internal): keyword
keyword has child node (token): int
accept token from the list: int
child node (internal): identifier
identifier has child node (token): x
accept token from the list: x
child node (token): =
accept token from the list: =
Child node (internal): math
----parent node math, finding children n
odes:
child node (multi):
----parent node multi, finding children
Tree Visualization
5.5
4.3
float_lit
6.9
Current Line Processing: 2
Next Line
EXIT

Lexer & Parser for TinyPie

Source Code

```
int x = 5 * 8+2;
float f = 5.5 + 4.3 * 6.9;
if(x> f):
        print("parsed");
```

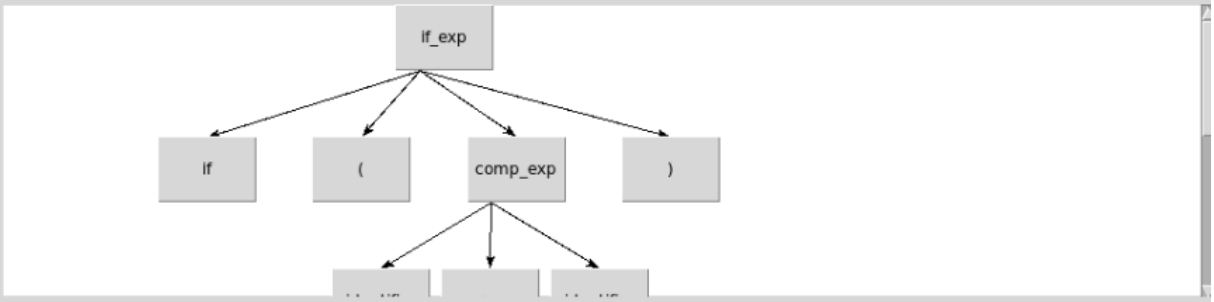
Tokens

```
<key,int>
<id,x>
<op,=>
<int_lit,5>
<op,*>
<int_lit,8>
<op,+>
<int_lit,2>
<sep,;>
<key,float>
<id,f>
<op,=>
<float_lit,5.5>
<op,+>
<float_lit,4.3>
<op,*>
```

Parse Tree

```
----parent node math_exp, finding childr
en nodes:
child node (internal): keyword
   keyword has child node (token): int
     accept token from the list: int
child node (internal): identifier
   identifier has child node (token): x
     accept token from the list: x
child node (token): =
     accept token from the list: =
Child node (internal): math
----parent node math, finding children n
odes:
child node (multi):
----parent node multi, finding children
```

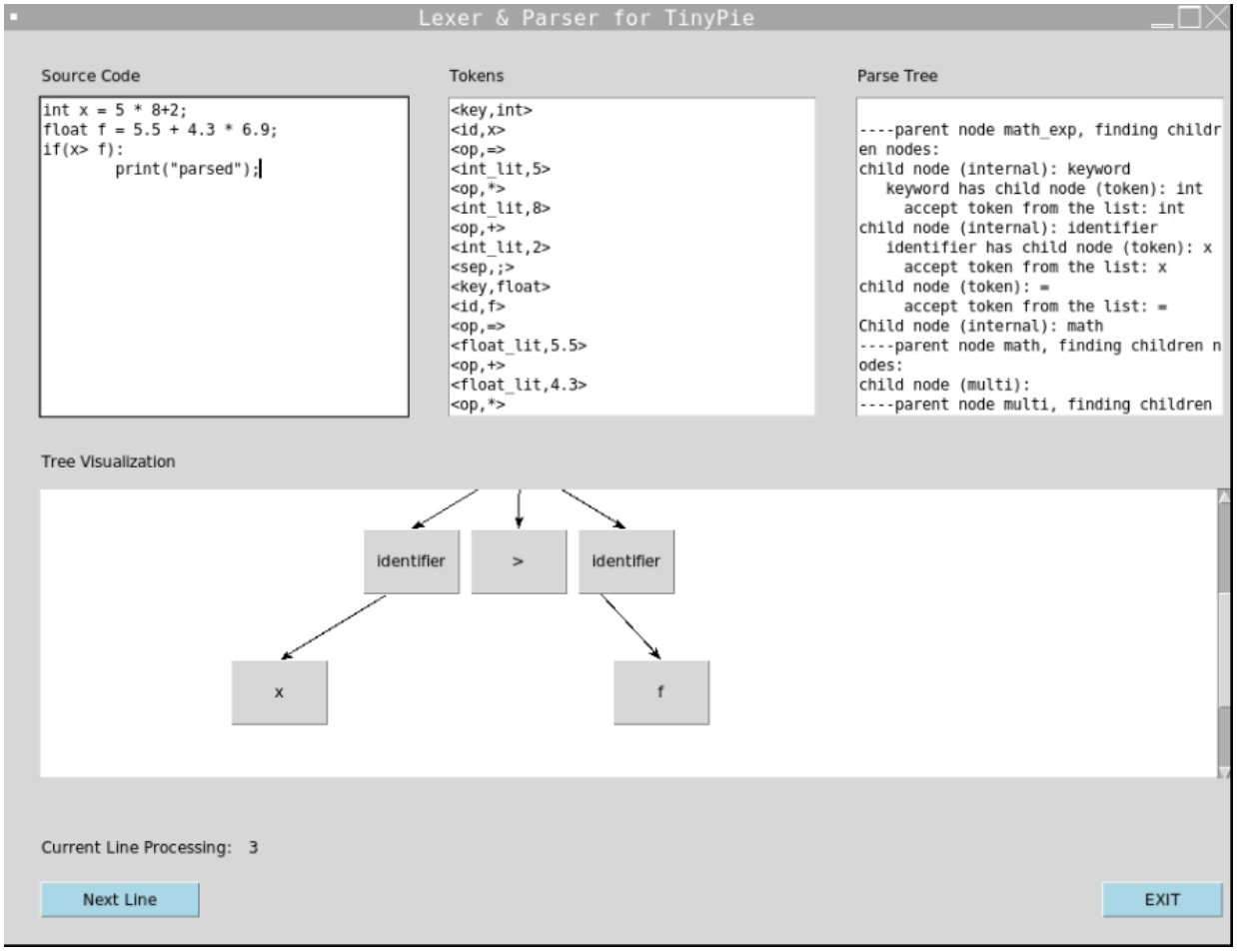
Tree Visualization

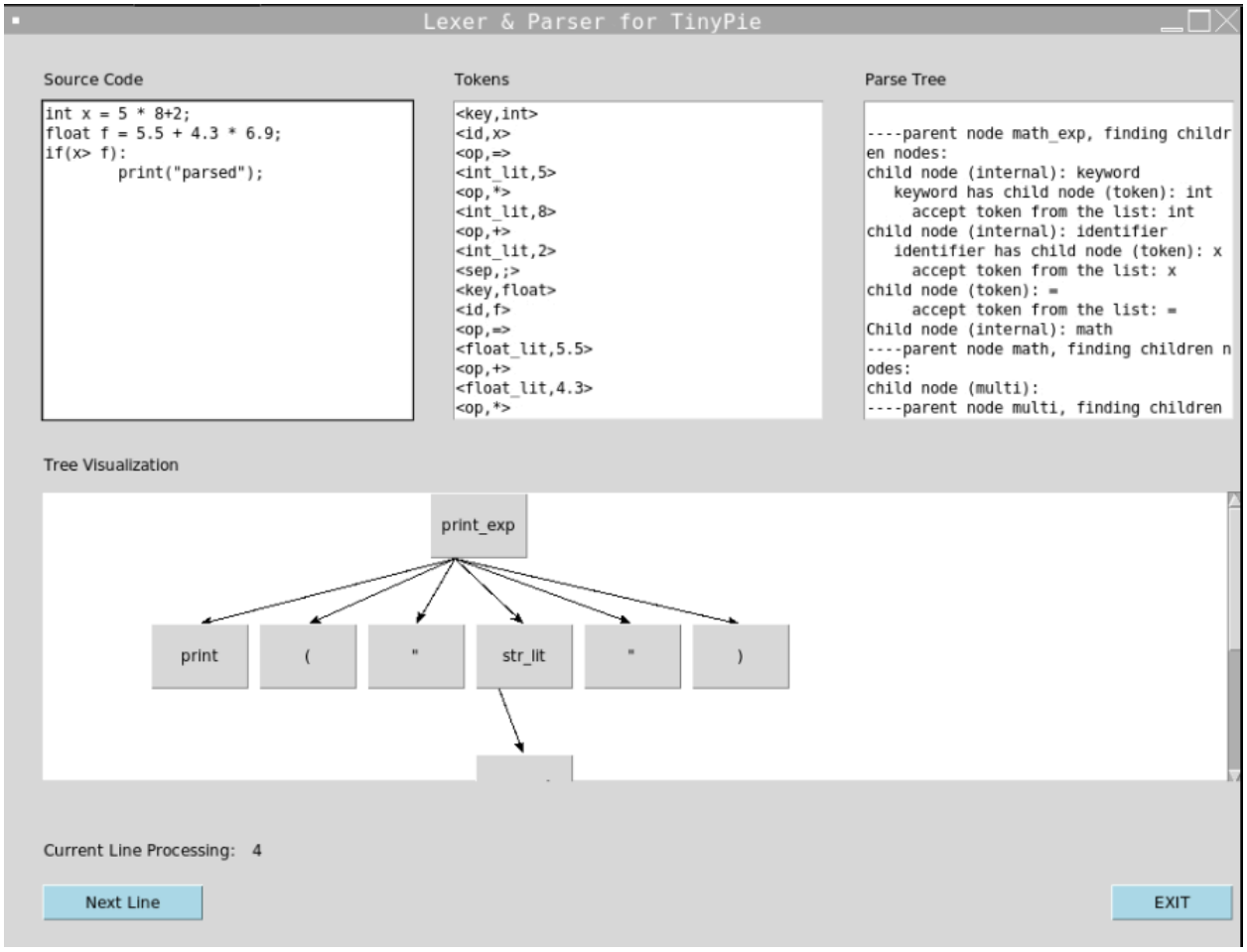


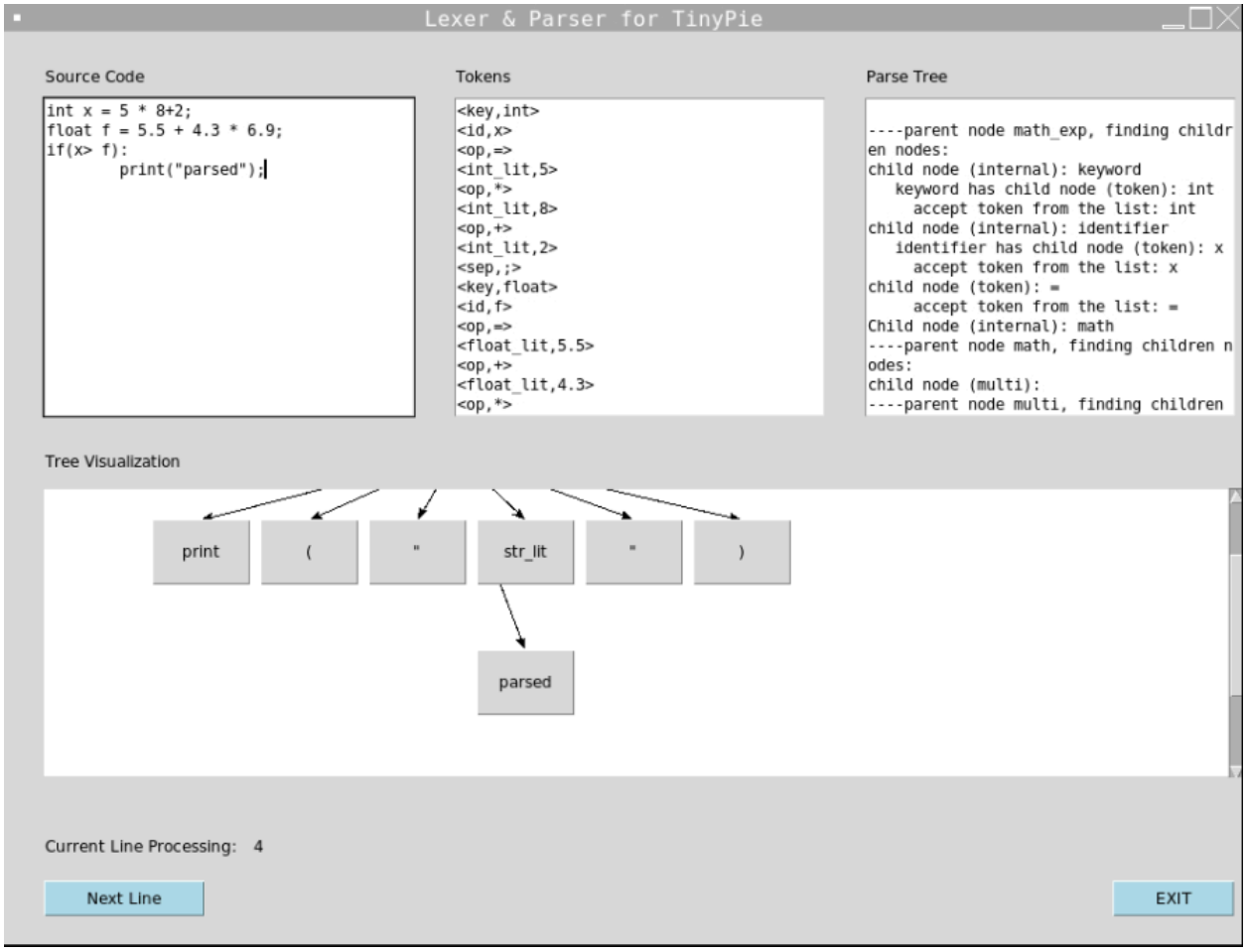
Current Line Processing: 3

Next Line

EXIT


Lexer & Parser for TinyPie
Source Code
int x = 5 * 8+2;
float f = 5.5 + 4.3 * 6.9;
if(x> f):
print("parsed");
Tokens
<key,int>
<id,x>
<op,=>
<int_lit,5>
<op,*>
<int_lit,8>
<op,+>
<int_lit,2>
<sep,;>
<key,float>
<id,f>
<op,=>
<float_lit,5.5>
<op,+>
<float_lit,4.3>
<op,*>
Parse Tree
----parent node math_exp, finding childr
en nodes:
child node (internal): keyword
keyword has child node (token): int
accept token from the list: int
child node (internal): identifier
identifier has child node (token): x
accept token from the list: x
child node (token): =
accept token from the list: =
Child node (internal): math
----parent node math, finding children n
odes:
child node (multi):
----parent node multi, finding children
Tree Visualization
identifier
>
identifier
x
f
Current Line Processing: 3
Next Line
EXIT


Lexer & Parser for TinyPie
Source Code
int x = 5 * 8+2;
float f = 5.5 + 4.3 * 6.9;
if(x> f):
print("parsed");
Tokens
<key,int>
<id,x>
<op,=>
<int_lit,5>
<op,*>
<int_lit,8>
<op,+>
<int_lit,2>
<sep,;>
<key,float>
<id,f>
<op,=>
<float_lit,5.5>
<op,+>
<float_lit,4.3>
<op,*>
Parse Tree
----parent node math_exp, finding childr
en nodes:
child node (internal): keyword
keyword has child node (token): int
accept token from the list: int
child node (internal): identifier
identifier has child node (token): x
accept token from the list: x
child node (token): =
accept token from the list: =
Child node (internal): math
----parent node math, finding children n
odes:
child node (multi):
----parent node multi, finding children
Tree Visualization
print_exp
print
(
"
str_lit
"
)
Current Line Processing: 4
Next Line
EXIT



Our team worked well with each other especially because we had consistent communication and it was clear what methods would be used since we agreed upon them ahead of time. Getting in-person time together was also very helpful so if any of us had questions we could collaborate and Jacob was able to draw out visual aids to help us understand what some functions were doing. One challenge that may have occurred was editing the report or code at the same time which would make it a bit confusing. We just communicated our availability and designated time for each member to do something then notified the others once they completed it. If there were any bugs or issues we also communicated that to each other so we could work together to debug it.

We ran our project on Replit by downloading the required libraries onto the website. To run a line of it, we type the input into the source code box; making sure to end it with a semicolon. Then we click the “next line” button. The proof that the tree data structure is working is found in the parse tree box and the tree visualizer. The parse tree box details every step the code is running through, while the tree visualizer offers a visual depiction of what the code found. *Please check out the demo for our final project, copy and paste the link into your browser and enjoy! Link: https://youtu.be/F60Vs4RM-FU*

References

Litux, litux.nl/mirror/pythonprogramming/0596000855_python2-CHP-17-SECT-10.html. Accessed 14 Dec. 2023.

"Tkinter Canvas." *Online Tutorials, Courses, and eBooks Library*, www.tutorialspoint.com/python/tk_canvas.htm. Accessed 14 Dec. 2023.

YouTube, Codemy.Com, 17 July 2020, https://youtu.be/0WafQCaok6g?si=6glZs3vOmpT6ICVP. Accessed 14 Dec. 2023.