Assignment II - Syntax Analyzer

Name: Janga Tushita Sharva

Roll Number: CS21BTECH11022

September 22, 2023

1 Steps for Execution

While in the `CS21BTECH11022` directory, run the following commands in the terminal. The following command generates y.tab.c and y.tab.h files.

```
$ yacc -d src.y
```

In case the generation of `y.output` file (which contains the details of automata) is desired, the following command is to be used.

```
$ yacc -d -v src.y
```

The following command generates the lex.yy.c file.

```
$ lex src.l
```

The following command is used to create an executable src.

```
$ gcc -g lex.yy.c y.tab.c -o src
```

The following command is to be executed. **The number here refers to the number of file name present in the test case file. By default I have assumed the file name to be "1.clike" and to be present at the file location `CS21BTECH11022/P2/TPP`. If that is not the case, please edit the line number 212 of src.y file accordingly.**

```
$ ./src number
```

Now, the corresponding `.txt` and `.parsed` files will be found in the location `CS21BTECH11022/P2/TP2`.

2 Mistakes in Public Test Cases

- In the third test case I submitted, the predicate in the loop has `lteq` instead of `leq`.
- The third test case I submitted did not have the `do` keyword in the while loop.
- The second test case I submitted had expression statements in class body and outside function body.

They have been rectified for this submission.

3 Known Issues with my implementation

3.1 Printing to parsed file (2)

1. When printing to the .parsed file, we were instructed to print the statements without any spaces at the start of the file. But with the implementation I came up with, I am unable to remove the tab spaces at the start. The possible reason I guess is that my lex program is unable to distinguish tab spaces and singular spaces.
2. When printing to a the .parsed file if the class has no arguments, ideally it should be printed just beside the line line class was defined, but my implementation prints it after the following curly brace:

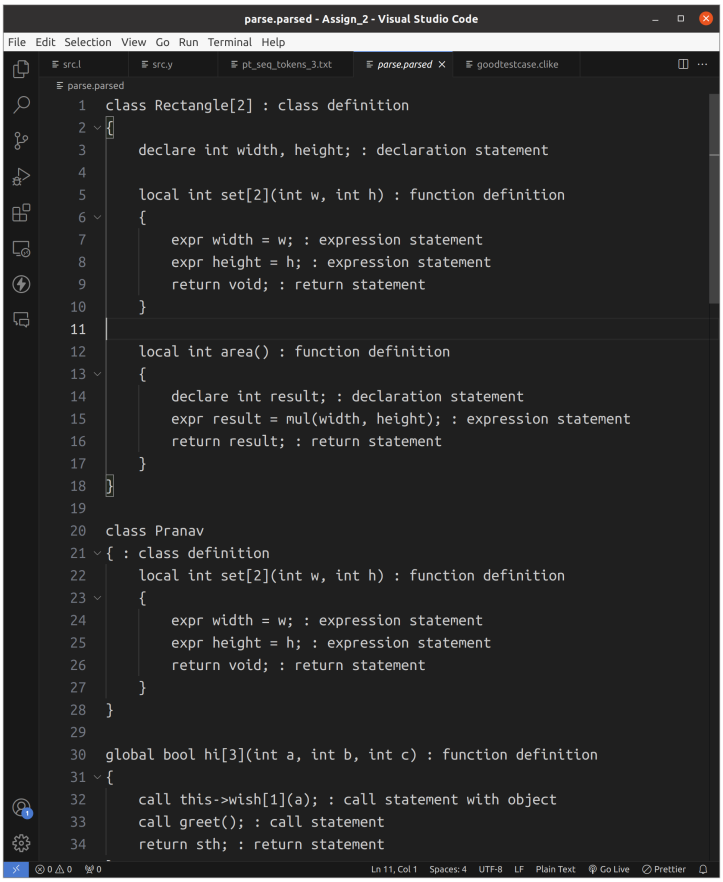

Figure 1: Error while printing in .parsed file

```
class IHaveNoArguements
{ :  class definition
local int set[2](int w, int h) :  function definition
{
expr width = w; :  expression statement
expr height = h; :  expression statement
return void; :  return statement
}
}
```

3.2 Predicates : In Loops or Conditional Statements

- A single predicate cannot be inside the brackets. However they can be there along with logical operators. For example,

  `in case that((i lt j))` gives an error
  `in case that((i lt j) and (i gt j))` **does not** give error.
  `in case that(i lt j and i gt j)` **does not** give error. (Has left precedence)

- This throws an error

```
in case that((call remainder[2](num1,i) eq 0) and (call remainder[2](num2,i) eq 0))
```

3.3 Token Type of Arrow

I didn't know if the arrow comes under `punctuation` or `operator`. I went with punctuation while implementing.

4 Class Instances handling

Initially, In my implementation, I defined the grammar for `declaration` to be:

```
declaration:  DECLARE declarebody SEMICOLON {fprintf(yyout, " :  declaration statement");};
declarebody:  datatype variable_list;
variable_list:  ID| ID COMMA variable_list;
```

Instead of resorting to allow grammar to allow rules of type `DECLARE ID ID`, I had defined a new token for **user defined datatypes**, called the token, `UDATATYPE`. This code at present restricts the number of classes to 2, but more of them can be allowed by changing the preprocessor `MAX` in line 4 of src.l to be the number of instances desired.

Implementation: Whenever the reserved word `class` is encountered, it turns on the boolean for class on. The corresponding identifier is then stored in a global array. Whenever an identifier is seen, it is first seen whether it is a user defined data type or an identifier (by referring to that array) and then returns the corresponding token.

5 Learning in this Assignment

5.1 Language Parsing and Grammar Specification:

I have a reasonable understanding of parsing techniques. I learned how to define a context-free grammar for a programming language using yacc tool.

5.2 Integration with Other Tools:

I have understood how to integrate lexer and parser into a larger tool chain for a programming language.

5.3 Error Handling:

I implemented error handling in parser, generating meaningful error messages when syntax errors occur.

5.4 Debugging and Output:

I used the y.output automata (especially for conflicts) and fprintf statements for debugging to track issues in my parser or lexer.