

**American University of Sharjah**

**Department of Computer Science and Engineering**

**----------------------------------------------------------------------------------------------------**

**Course: CMP 321( Programming Languages)**

**Professor: Dr. Michel Bernard Pasquier**

**----------------------------------------------------------------------------------------------------**

### **Project Title: Prolog Parser**

### **Report Title : Project Contribution Brief**

### **---------------------------------------------------------------------------------------------------- Team Members Details**

**Section Number: 01**

**Group Number: 03**

|  |  |
| --- | --- |
| **Khondoker Labib Rahman** | **b00084215** |
| **Prem Rajendran** | **b00084833** |
| **Harshit Jiandani** | **b00082536** |
| **Sarthak Maloo** | **b00083635** |

**Description of Work**

Our project code mainly comprised of 3 sections namely:

1. Lexical Analyzer functions

2. Syntax Analyzer functions

3. Main functions (including file I/O)

We decided to allocate members based on the quantity of work. Since, the lexical analyzer required few functions, we allocated one member, namely *Prem* to handle this function. Furthermore, the syntax analyzer functions required more work so we decided to allocate a pair namely *Harshit* and *Khondoker* to simultaneously work on it. Lastly, all the testing functions and file I/O was handled by *Sarthak*.

However, all code was peer-reviewed by all team members before making commits to our GitHub repository. Furthermore, multiple in-person and online meetings were scheduled to discuss and debug our source code. During the final testing phase, the team members had a meeting testing it multiple times on all our devices, making sure to account for any runtime errors, bugs, or malfunctions. Finally, all changes made during the testing phase were pushed to the repository in one final commit which contained final versions of all files.

Now we will provide testimonials from each member describing their individual work.

**Prem Rajendran:**

My contribution mainly consisted of the initial source code in the file, which includes the global constants required to assign token codes to the characters we encounter in our parsing of a file. Furthermore, I coded a lookup table function, namely *find*, that matched a character with its corresponding unique token number. I also coded another function *get\_char*, that was required to process the file character by character and match to a certain type of character (for e.g., uppercase, lowercase, digit). Then to complete the lexical analyzer, I created a function called *lex*, that processes each sequence of tokens to determine which type of token it is. Thus, this created function wrapped up the lexical analyzer which was called in subsequent functions.

**Khondoker Labib Rahman:**

After Prems work on the lexical analyzer , I was able to implement functions for the syntax analyzer. Since the syntax analyzer required many functions(one function was needed for almost every grammar rule) , we decided to split this work between me and *Harshit.* I started with the bottom up approach where I coded the syntax analyser functions for the smaller more specific structures in our program which included the functions to check syntax/grammar of *special, alphanum, numeral, character, string, characterlist, variable,small atom and atom.* In each of these functions I called *get\_char* and *lex* functionsto get the token and do the lexical analysis, after that I used the grammar rules given to do the syntax analysis. If an error was found I incremented the number of errors(*errors\_count)* and added the description of the error to *errors\_list..*

**Harshit Jiandani:**

After *Khondoker* was done with making the syntax analyser functions for the smaller structures ,I was able to make the syntax analyser functions for the larger structures in our program which included *term,term list,structure,predicate, predicatelist,query,clause,clauselist,program*. These functions also got the tokens and did the lexical analysis using *get\_char and lex* and if any error was found its description was added to *errors\_list* and *errors\_count* was incremented . However the main difference between my work and *Khondoker’s* work was that he was working bottom up and I was working top down , so I often called the functions he created in my own functions. For example to check if something is a *term* we need to check if it's a *numeral* or *variable* or *atom* or *structure*. Since I had a top down approach I wouldn’t write the code to check if it's a numeral or variable or atom (nexttoken== not needed in this scenario) , I would rather just call the syntax analyser functions for these that were created by *Khondoker*. He worked bottom up to create the syntax analyzer functions for smaller structures and I then used these functions to create syntax analyzers for larger structures (which included the smaller structures in them).

**Sarthak Maloo:**

I mainly worked on the code for File I/O and final compilation and testing. I worked on the driver function namely *mainfunction* and *display\_errors*. The *mainfunction* is where the entire Parser is actually executed and each .txt file is tested. We access the user’s current directory to find all ‘.txt’ files that start with a numerical value. Then we check whether the list of files are sequentially numbered starting at 1, and only execute the Parser on the sequentially ordered files. After each file is analyzed, the values of each parameter is reset to work on the next file. In the *display\_errors* function, we pass the output stream file as an argument, so that the function can run for each file. This function prints the errors (if exist) to the console and writes it to the ‘parser\_output.txt’ file. The errors consist of a type, line number and index number in the line. This format allows the user to refer to the .txt files and correct their code with ease.

**Appendix**

All source code and documentation can be found on GitHub at <https://github.com/Pr3mR4jendran/CMP321Proj.git>