Joseph Edwards, Cliff Sparks, Zack Mason

Dr. MacEvoy

Programming Languages

5/12/22

Group 1 - RDP Comment Implementation

For our group project, we attempted to implement comment notation into a recursive descent parser for a calculator. To achieve this, we decided to write a void function that would be called at the beginning of the scanner to handle single line and multiline comments. After passing the “got” struct into our new function, we checked for a pound sign “#” as the symbol for single line comments and curly braces “{}” for multiline comments. Once the function begins to check for a comment, it will increase the line number to the next line in a similar way that the newline part of the got function goes to the next line.

For the nested comment portion of the function, the code works by looking for an open curly brace “{“, then continuously loops until it finds a closing curly brace “}”. As it looks for the next curly brace it ignores all the characters since they are not relevant to the parser. Our original idea was to pass the ‘#’ and ‘{}’ as tokens, but this is a far more simple solution to this problem. In our opinion, it makes more sense to check for comments and process them before it even tries to parse the characters, avoiding the problem of implementing new tokens completely.

Here is the pseudocode for our concept… This version compiles and does not break the original code but we did not make any tests for comments specifically.

```
void skipComments(got c)
{
    if(eof == false)
    {
      //single line comment
      if (c.ch == '#')
      {
        //get to the end of line
        ++line;
        col = 0;
      }
      //Nested comments
      if (c.ch == '{')
      {
        while(c.ch != '}')
        {
          got c = get();
          ++col;
          if (c.ch == '\n')
          {
            ++line;
            col = 0;
          }
        }
      }
    }


}
//-------------------------------------------------
virtual Token::Ptr nextFromStream() override
{
  got g = get();
  skipComments(g);//////////////////////////////
  if (g.ch >= 'a' && g.ch <= 'z')
    return Token::identifier(std::string(1, char(g.ch)), g.line, g.col)
```