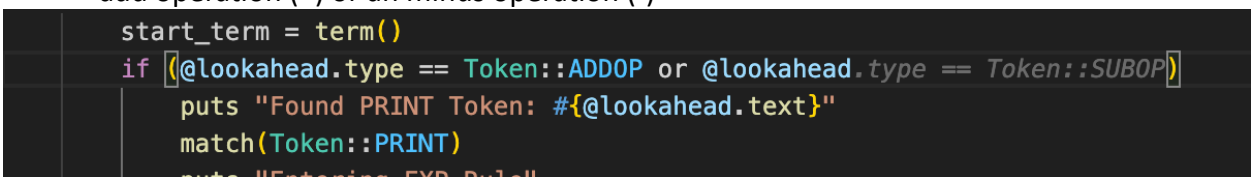
Clay Ramey – COMP 3220 – Copilot bonus

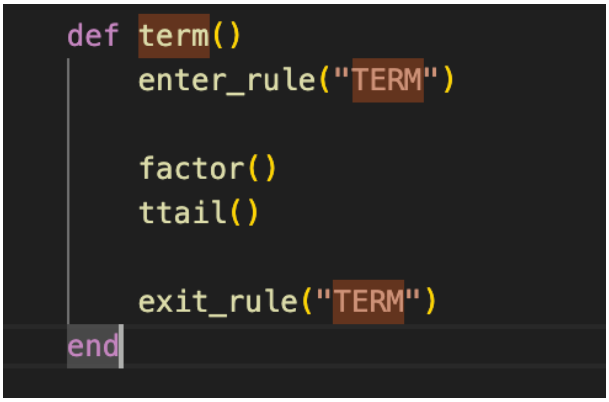
Copilot helped me by giving ideas for what a functions logic should be based off what is in my classes currently and what similar GitHub projects recommend.

1) Copilot recommended a if statement initially to utilize lookahead to see if the token is an add operation (+) or an minus operation (-)



2) Next, when creating the term function, it recommended the check for if it's a division operator or multiplication operator. I did need to change the shiftSiblingsDown function name to shiftChildDown.

I then realized this is not how the term function should be used, and changed it to:



3) I then was creating the factor function to check for the LPAREN, ID, INT, and RPAREN. The logic was close to optimal. It didn't have the found token function call for ID.

```
def factor()
    if (@lookahead.type == Token::LPAREN)
        match(Token::LPAREN)
        exp = statement()
        match(Token::RPAREN)
        return exp
    elsif (@lookahead.type == Token::INT)
        t = @lookahead
        match(Token::INT)
        return AST.new(t)
    else
        puts "Error in factor"
        @errors += 1
        return nil
    end
end
```

This caused me to add the ID token check, and factor turned out like this:

```
def factor()
    enter_rule("FACTOR")

    if (@lookahead.type == Token::LPAREN)
        found_token("LPAREN")
        match(Token::LPAREN)
        exp()
        if (@lookahead.type == Token::RPAREN)
            found_token("RPAREN")
        end
        match(Token::RPAREN)
    elsif (@lookahead.type == Token::INT)
        found_token("INT")
        match(Token::INT)
    elsif (@lookahead.type == Token::ID)
        found_token("ID")
        match(Token::ID)
    else
        match(Token::LPAREN, Token::INT, Token::ID)
    end

    exit_rule("FACTOR")
end
```

4) Next, I added the ttaill function and it gave me the following recommendations:

```
def ttail()
    if (@lookahead.type == Token::MULTOP)
        t = @lookahead
        match(Token::MULTOP)
        f = factor()
        if (@lookahead.type == Token::MULTOP or @lookahead.type == Token::DIVOP)
            tt = ttail()
            tt.addAsNextChild(f)
            tt.addAsNextChild(t)
            return tt
        else
            t.addAsNextChild(f)
            return t
        end
    elsif (@lookahead.type == Token::DIVOP)
        t = @lookahead
        match(Token::DIVOP)
        f = factor()
        if (@lookahead.type == Token::MULTOP or @lookahead.type == Token::DIVOP)
            tt = ttail()
            tt.addAsNextChild(f)
            tt.addAsNextChild(t)
            return tt
        else
            t.addAsNextChild(f)
            return t
        end
    else
        puts "Error in ttail"
        @errors += 1
        return nil
    end
end
```

The ttail function was way to overdone, after I fixed the ttaill function it looked like this:

```
def ttail()
    enter_rule("TTAIL")

    if (@lookahead.type == Token::MULTOP)
        found_token("MULTOP")
        match(Token::MULTOP)
        factor()
        ttail()
    elsif (@lookahead.type == Token::DIVOP)
        found_token("DIVOP")
        match(Token::DIVOP)
        factor()
        ttail()
    else
        choose_epsilon("MULTOP", "DIVOP")
    end

    exit_rule("TTAIL")
end
```

5) Next, I created the assign function. Copilot gave me this recommendation for my code.

```
def assign()
    if (@lookahead.type == Token::ID)
        puts "Found ID Token: #{@lookahead.text}"
        match(Token::ID)
        match(Token::ASSGN)
        puts "Entering EXP Rule"
        exp()
    else
        puts "Error in assign()"
    end
```

This is how it looked after I changed the code to fit my needs. It basically added the check for the tokens manually instead of using a match function. This worked simpler for how the architecture of my code was.

```
def assign()
    enter_rule("ASSGN")

    if (@lookahead.type == Token::ID)
        found_token("ID")
    end
    match(Token::ID)
    if (@lookahead.type == Token::ASSGN)
        found_token("ASSGN")
    end
    match(Token::ASSGN)
    exp()

    exit_rule("ASSGN")
end
```

From this coding assignment, I've realized Copilot is very useful if used right. It gives great ideas for how to design your code and can give a good idea for how to start logic in your various functions. I will definitely continue to use copilot!