HW 3 Problem 2

Question: Write a simple parser for the following EBNF grammar:

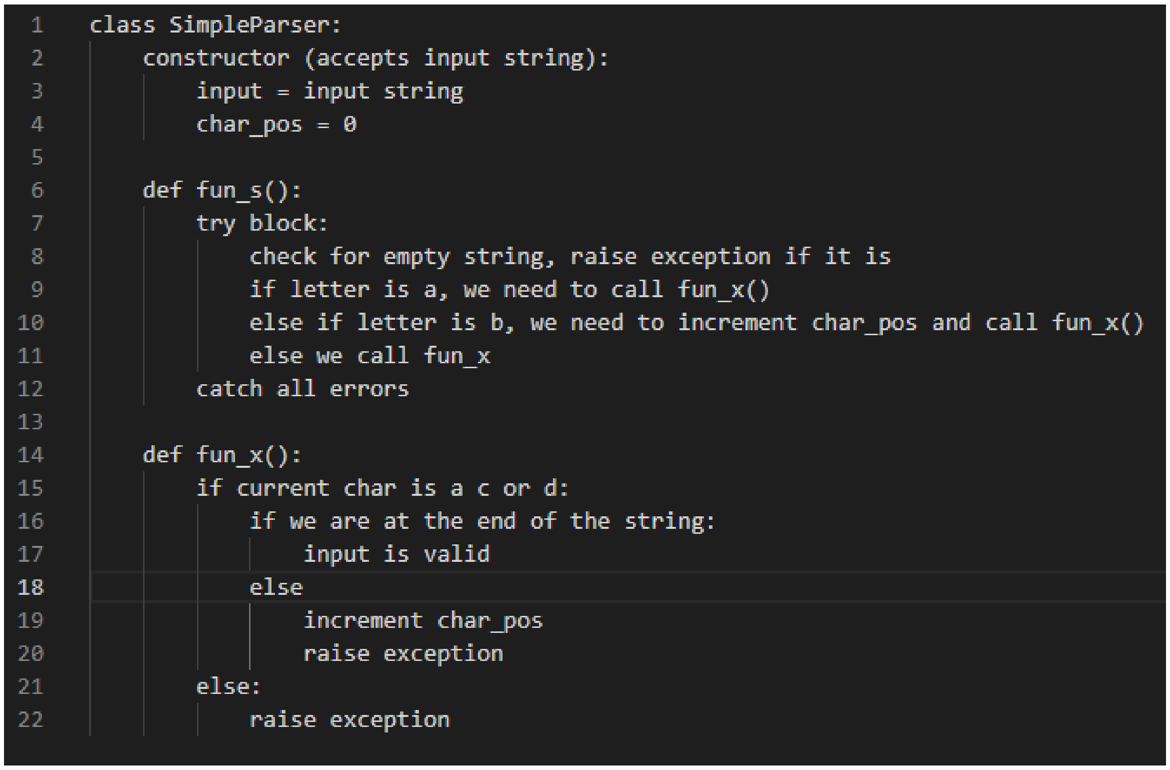
-
```
<S> ::= { a } <X> | b <X>
<X> ::= c | d
```

Provided test cases:

1. Input: "bc"
   Print: "Input is valid"
2. Input: "acd"
   Print: "Syntax error at character position 2"
3. Input: "aaad"
   Print: "Input is valid"
4. Input: "c"
   Print: "Input is valid"
5. Input: "2yz"
   Print: "Syntax error at character position 0"
6. Input: "" (empty)
   Print: "Syntax error at character position 0"

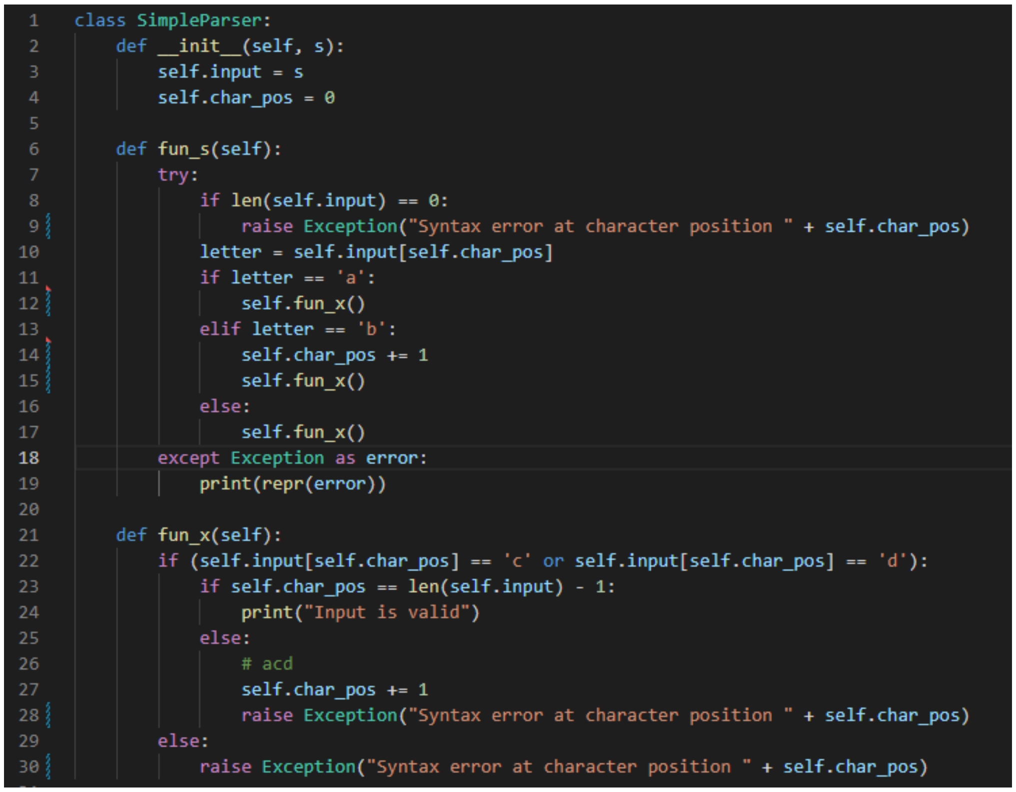
Note: First language is Python

Algorithm/Pseudo code:



```
class SimpleParser:
    constructor (accepts input string):
        input = input string
        char_pos = 0

    def fun_s():
        try block:
            check for empty string, raise exception if it is
            if letter is a, we need to call fun_x()
            else if letter is b, we need to increment char_pos and call fun_x()
            else we call fun_x
        catch all errors

    def fun_x():
        if current char is a c or d:
            if we are at the end of the string:
                input is valid
            else
                increment char_pos
                raise exception
        else:
            raise exception
```

Actual Code:

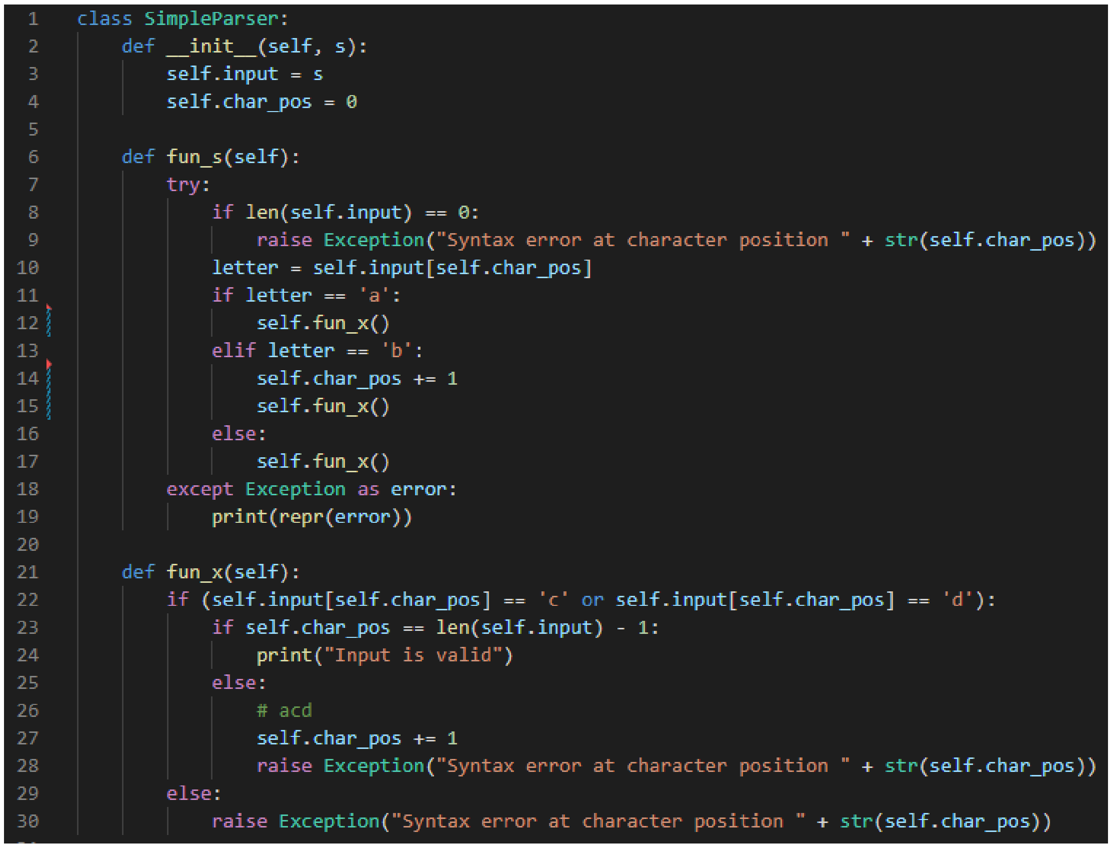


```python
class SimpleParser:
    def __init__(self, s):
        self.input = s
        self.char_pos = 0

    def fun_s(self):
        try:
            if len(self.input) == 0:
                raise Exception("Syntax error at character position " + self.char_pos)
            letter = self.input[self.char_pos]
            if letter == 'a':
                self.fun_x()
            elif letter == 'b':
                self.char_pos += 1
                self.fun_x()
            else:
                self.fun_x()
        except Exception as error:
            print(repr(error))

    def fun_x(self):
        if (self.input[self.char_pos] == 'c' or self.input[self.char_pos] == 'd'):
            if self.char_pos == len(self.input) - 1:
                print("Input is valid")
            else:
                # acd
                self.char_pos += 1
                raise Exception("Syntax error at character position " + self.char_pos)
        else:
            raise Exception("Syntax error at character position " + self.char_pos)
```

Syntax Error:

- When I raise an exception, I need to convert self.char_pos to a string when I create the error message

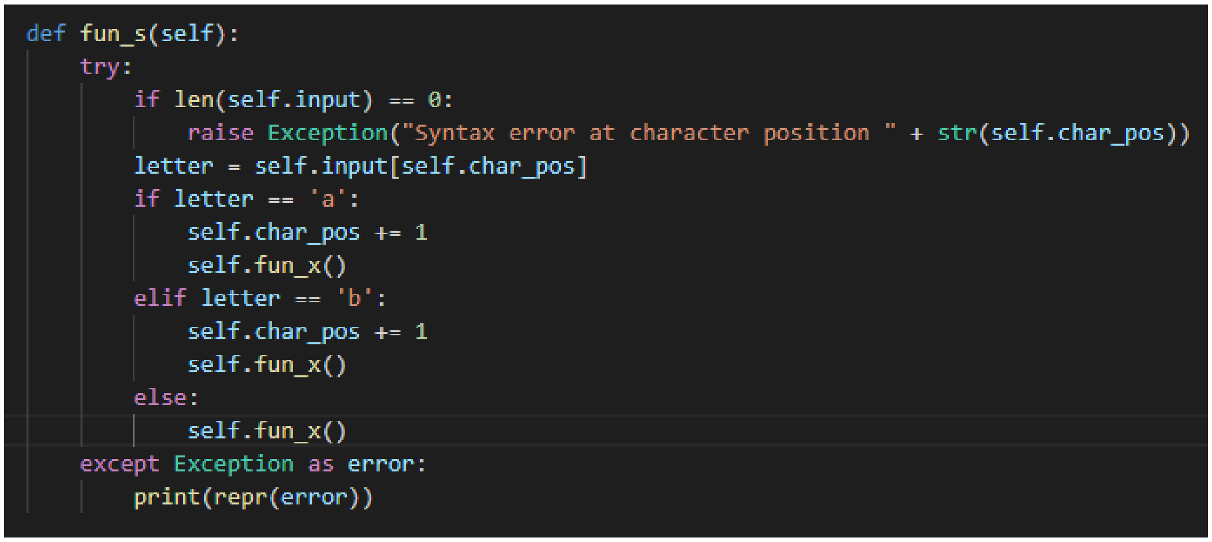
Working Code



```
1   class SimpleParser:
2       def __init__(self, s):
3           self.input = s
4           self.char_pos = 0
5
6       def fun_s(self):
7           try:
8               if len(self.input) == 0:
9                   raise Exception("Syntax error at character position " + str(self.char_pos))
10              letter = self.input[self.char_pos]
11              if letter == 'a':
12                  self.fun_x()
13              elif letter == 'b':
14                  self.char_pos += 1
15                  self.fun_x()
16              else:
17                  self.fun_x()
18          except Exception as error:
19              print(repr(error))
20
21      def fun_x(self):
22          if (self.input[self.char_pos] == 'c' or self.input[self.char_pos] == 'd'):
23              if self.char_pos == len(self.input) - 1:
24                  print("Input is valid")
25              else:
26                  # acd
27                  self.char_pos += 1
28                  raise Exception("Syntax error at character position " + str(self.char_pos))
29          else:
30              raise Exception("Syntax error at character position " + str(self.char_pos))
```

Debugging

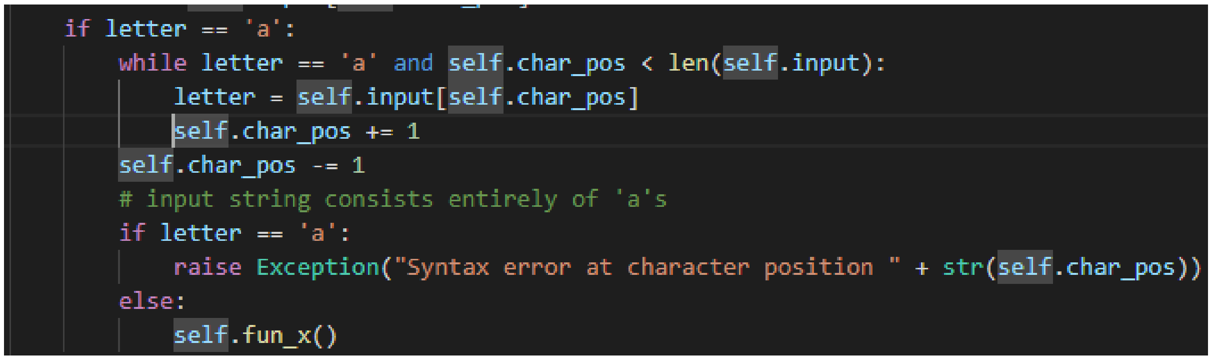
1. Did not increment char_pos before calling self.fun_x() in line 12. To fix:



```
def fun_s(self):
    try:
        if len(self.input) == 0:
            raise Exception("Syntax error at character position " + str(self.char_pos))
        letter = self.input[self.char_pos]
        if letter == 'a':
            self.char_pos += 1
            self.fun_x()
        elif letter == 'b':
            self.char_pos += 1
            self.fun_x()
        else:
            self.fun_x()
    except Exception as error:
        print(repr(error))
```

-

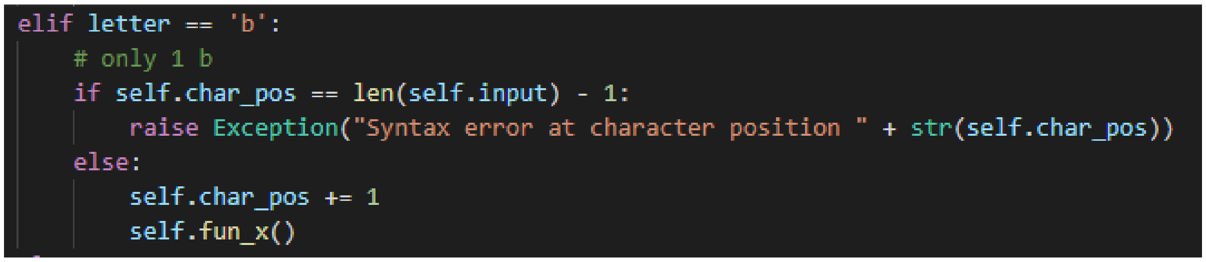
2. Did not account for the fact that we can have multiple “A”s in a row. Added a while loop to account for multiple “A”s



```
    if letter == 'a':
        while letter == 'a' and self.char_pos < len(self.input):
            letter = self.input[self.char_pos]
            self.char_pos += 1
        self.char_pos -= 1
        # input string consists entirely of 'a's
        if letter == 'a':
            raise Exception("Syntax error at character position " + str(self.char_pos))
        else:
            self.fun_x()
```

-

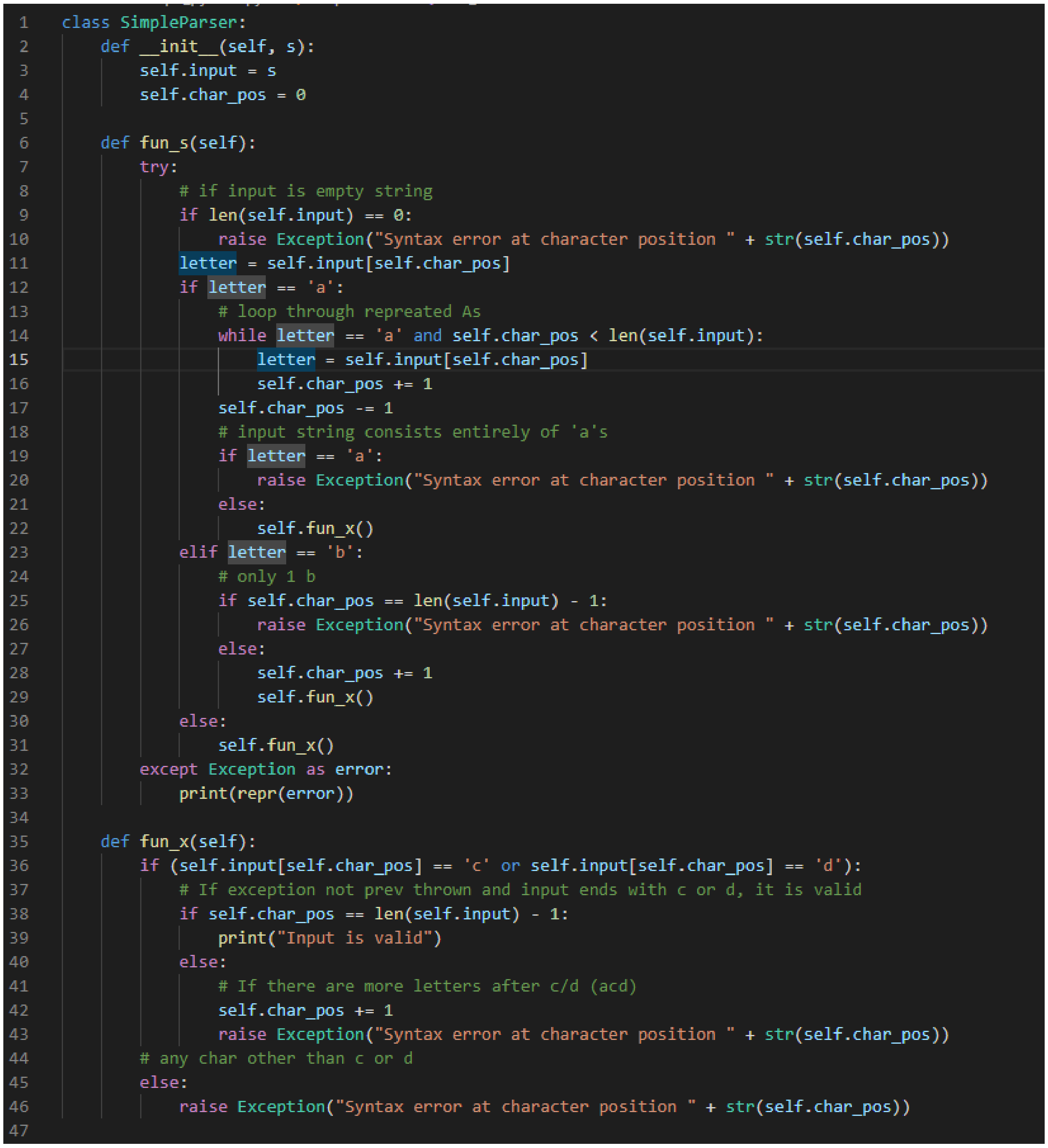
3. Need to catch error involving multiple repeating “b”s



```
elif letter == 'b':
    # only 1 b
    if self.char_pos == len(self.input) - 1:
        raise Exception("Syntax error at character position " + str(self.char_pos))
    else:
        self.char_pos += 1
        self.fun_x()
```

-

Add Documentation



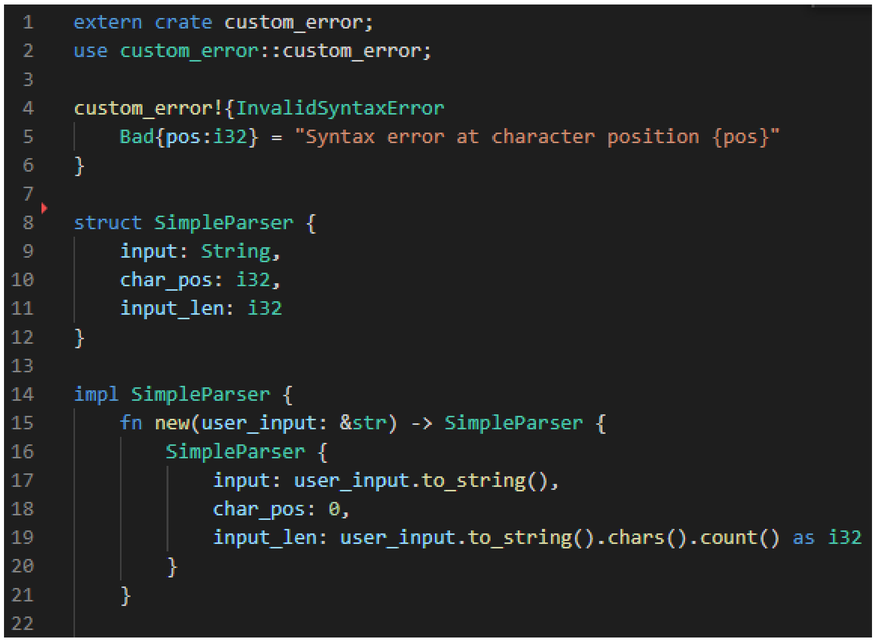
```python
class SimpleParser:
    def __init__(self, s):
        self.input = s
        self.char_pos = 0

    def fun_s(self):
        try:
            # if input is empty string
            if len(self.input) == 0:
                raise Exception("Syntax error at character position " + str(self.char_pos))
            letter = self.input[self.char_pos]
            if letter == 'a':
                # loop through prepeated As
                while letter == 'a' and self.char_pos < len(self.input):
                    letter = self.input[self.char_pos]
                    self.char_pos += 1
                self.char_pos -= 1
                # input string consists entirely of 'a's
                if letter == 'a':
                    raise Exception("Syntax error at character position " + str(self.char_pos))
                else:
                    self.fun_x()
            elif letter == 'b':
                # only 1 b
                if self.char_pos == len(self.input) - 1:
                    raise Exception("Syntax error at character position " + str(self.char_pos))
                else:
                    self.char_pos += 1
                    self.fun_x()
            else:
                self.fun_x()
        except Exception as error:
            print(repr(error))

    def fun_x(self):
        if (self.input[self.char_pos] == 'c' or self.input[self.char_pos] == 'd'):
            # If exception not prev thrown and input ends with c or d, it is valid
            if self.char_pos == len(self.input) - 1:
                print("Input is valid")
            else:
                # If there are more letters after c/d (acd)
                self.char_pos += 1
                raise Exception("Syntax error at character position " + str(self.char_pos))
        # any char other than c or d
        else:
            raise Exception("Syntax error at character position " + str(self.char_pos))
```

Extra Test Cases Used for Debugging:

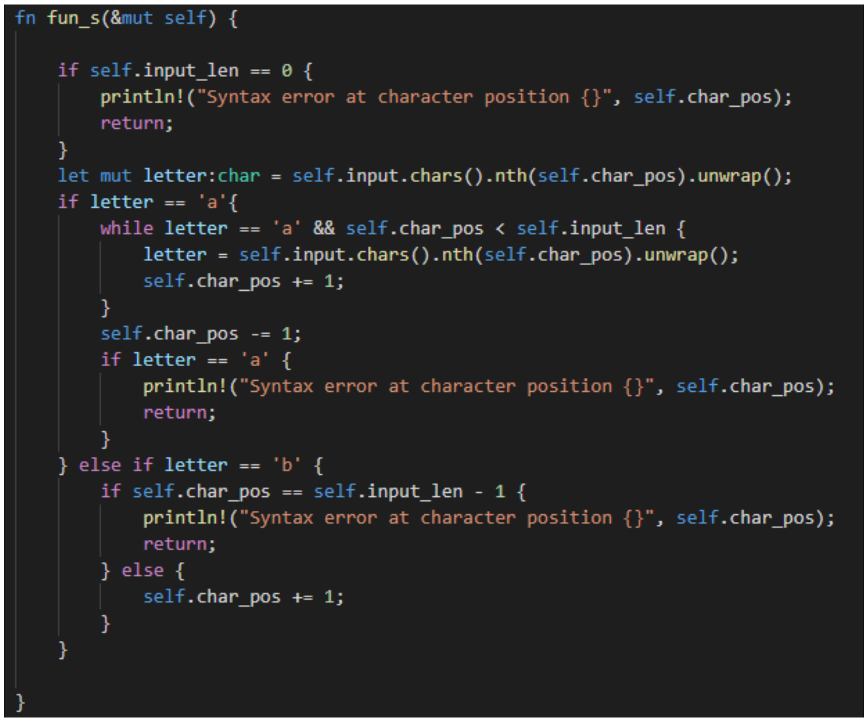
1. Input: “abd”
   Print: “Syntax error at character position 1”
2. Input: “b”
   Print: “Syntax error at character position 0”
3. Input: “aaaa”
   Print: “Syntax error at character position 3”
4. Input “bbbb”
   Print: “Syntax error at character position 1”
5. Input: “cc”
   Print: “Syntax error at character position 1”
6. Input: “3”
   Print: “Syntax error at character position 0”

Note: Second language is Rust

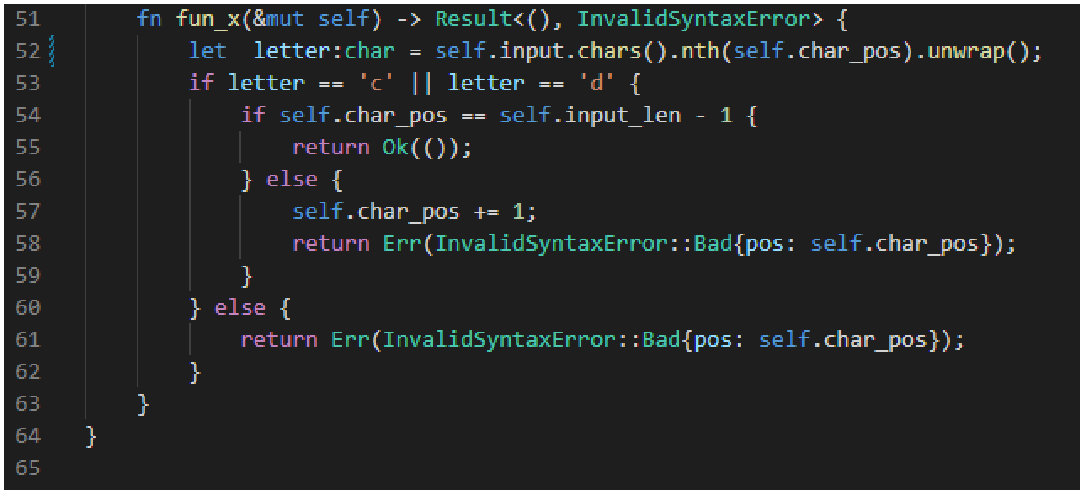
Actual Code:



```rust
extern crate custom_error;
use custom_error::custom_error;

custom_error!{InvalidSyntaxError
    Bad{pos:i32} = "Syntax error at character position {pos}"
}

struct SimpleParser {
    input: String,
    char_pos: i32,
    input_len: i32
}

impl SimpleParser {
    fn new(user_input: &str) -> SimpleParser {
        SimpleParser {
            input: user_input.to_string(),
            char_pos: 0,
            input_len: user_input.to_string().chars().count() as i32
        }
    }

```



```rust
fn fun_s(&mut self) {

    if self.input_len == 0 {
        println!("Syntax error at character position {}", self.char_pos);
        return;
    }
    let mut letter:char = self.input.chars().nth(self.char_pos).unwrap();
    if letter == 'a'{
        while letter == 'a' && self.char_pos < self.input_len {
            letter = self.input.chars().nth(self.char_pos).unwrap();
            self.char_pos += 1;
        }
        self.char_pos -= 1;
        if letter == 'a' {
            println!("Syntax error at character position {}", self.char_pos);
            return;
        }
    } else if letter == 'b' {
        if self.char_pos == self.input_len - 1 {
            println!("Syntax error at character position {}", self.char_pos);
            return;
        } else {
            self.char_pos += 1;
        }
    }

}
```

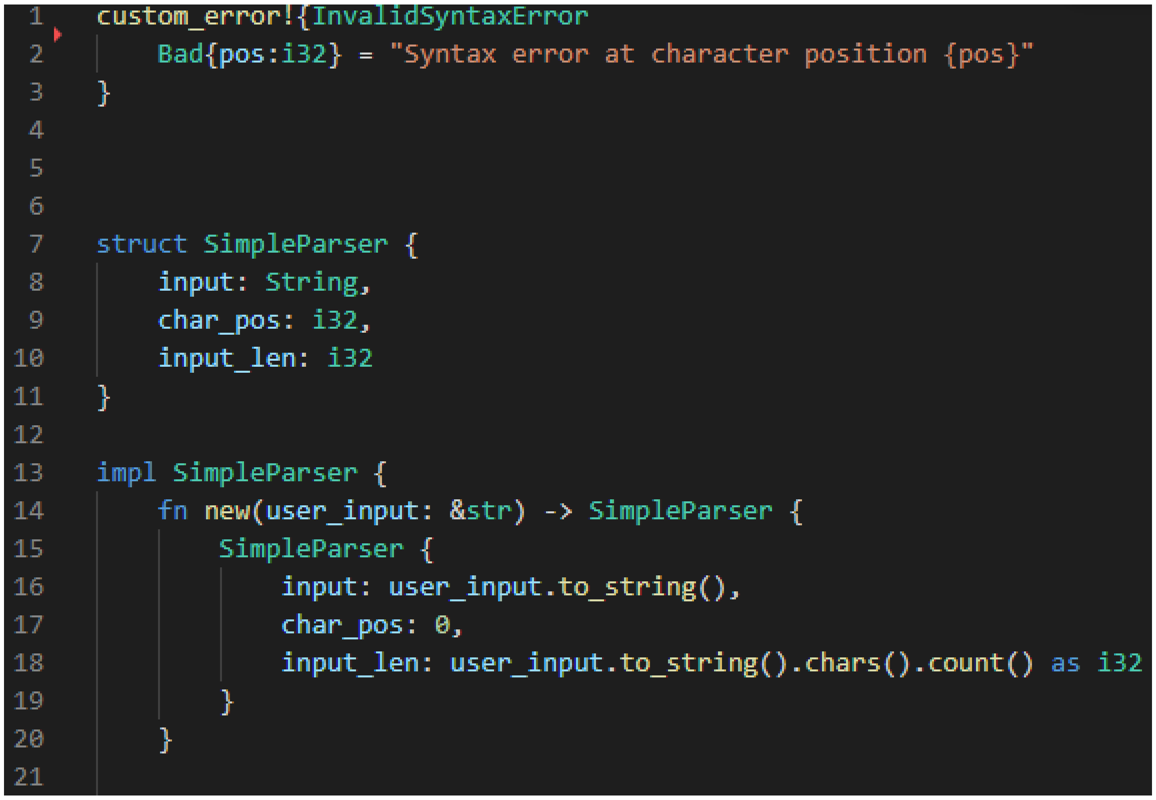


```rust
    fn fun_x(&mut self) -> Result<(), InvalidSyntaxError> {
        let  letter:char = self.input.chars().nth(self.char_pos).unwrap();
        if letter == 'c' || letter == 'd' {
            if self.char_pos == self.input_len - 1 {
                return Ok(());
            } else {
                self.char_pos += 1;
                return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
            }
        } else {
            return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
        }
    }
}

```

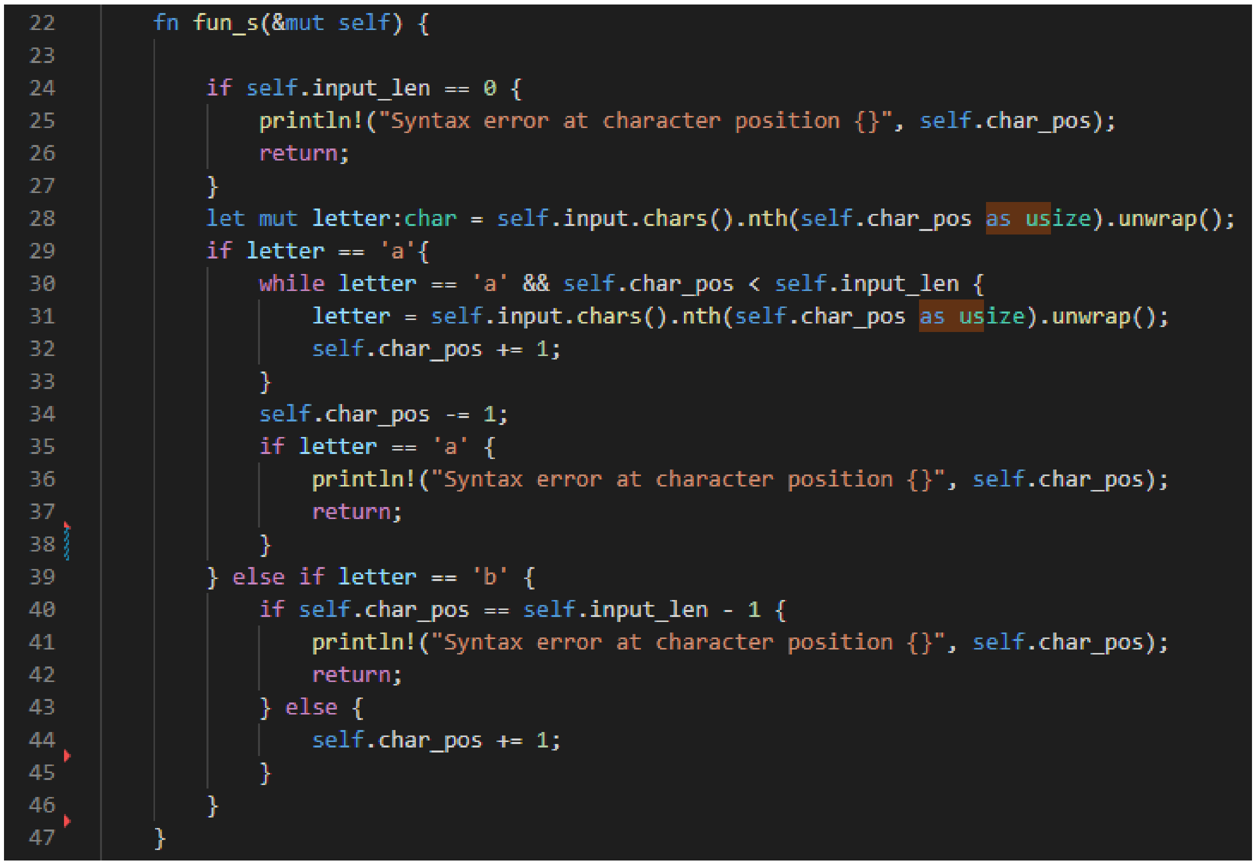
Syntax Error:

1. Whenever I am indexing the input string, I need to convert self.char_pos to usize using self.char_pos as usize

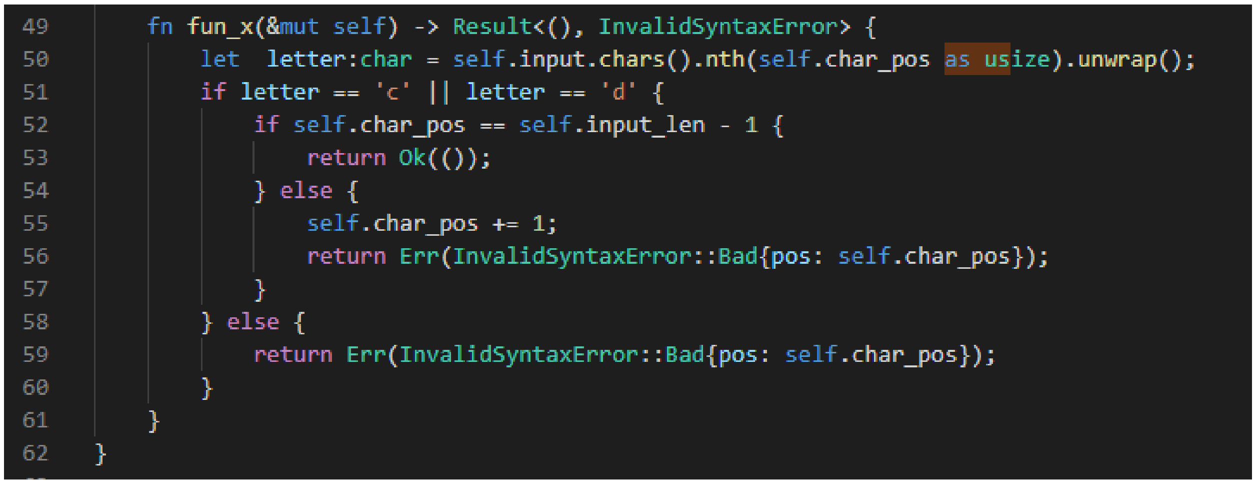
Working Code



```rust
custom_error!{InvalidSyntaxError
    Bad{pos:i32} = "Syntax error at character position {pos}"
}



struct SimpleParser {
    input: String,
    char_pos: i32,
    input_len: i32
}

impl SimpleParser {
    fn new(user_input: &str) -> SimpleParser {
        SimpleParser {
            input: user_input.to_string(),
            char_pos: 0,
            input_len: user_input.to_string().chars().count() as i32
        }
    }

```

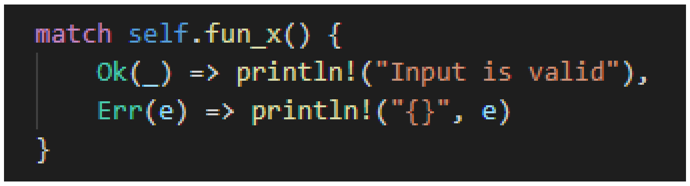


```rust
    fn fun_s(&mut self) {

        if self.input_len == 0 {
            println!("Syntax error at character position {}", self.char_pos);
            return;
        }
        let mut letter:char = self.input.chars().nth(self.char_pos as usize).unwrap();
        if letter == 'a'{
            while letter == 'a' && self.char_pos < self.input_len {
                letter = self.input.chars().nth(self.char_pos as usize).unwrap();
                self.char_pos += 1;
            }
            self.char_pos -= 1;
            if letter == 'a' {
                println!("Syntax error at character position {}", self.char_pos);
                return;
            }
        } else if letter == 'b' {
            if self.char_pos == self.input_len - 1 {
                println!("Syntax error at character position {}", self.char_pos);
                return;
            } else {
                self.char_pos += 1;
            }
        }
    }
```



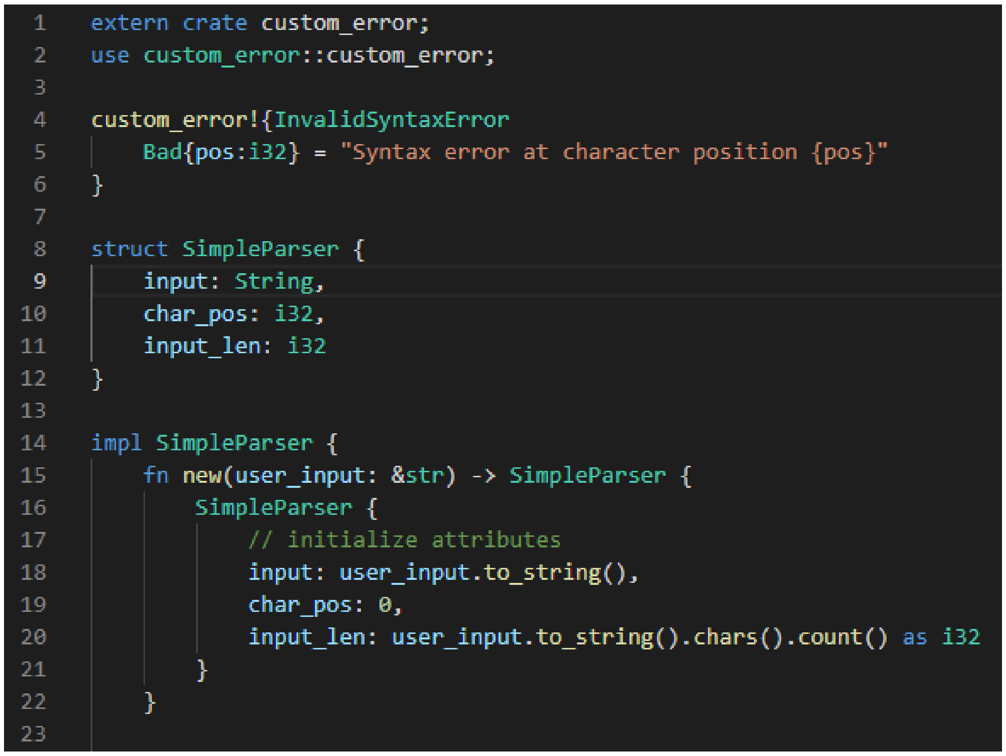
```rust
    fn fun_x(&mut self) -> Result<(), InvalidSyntaxError> {
        let  letter:char = self.input.chars().nth(self.char_pos as usize).unwrap();
        if letter == 'c' || letter == 'd' {
            if self.char_pos == self.input_len - 1 {
                return Ok(());
            } else {
                self.char_pos += 1;
                return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
            }
        } else {
            return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
        }
    }
}
```

Debugging

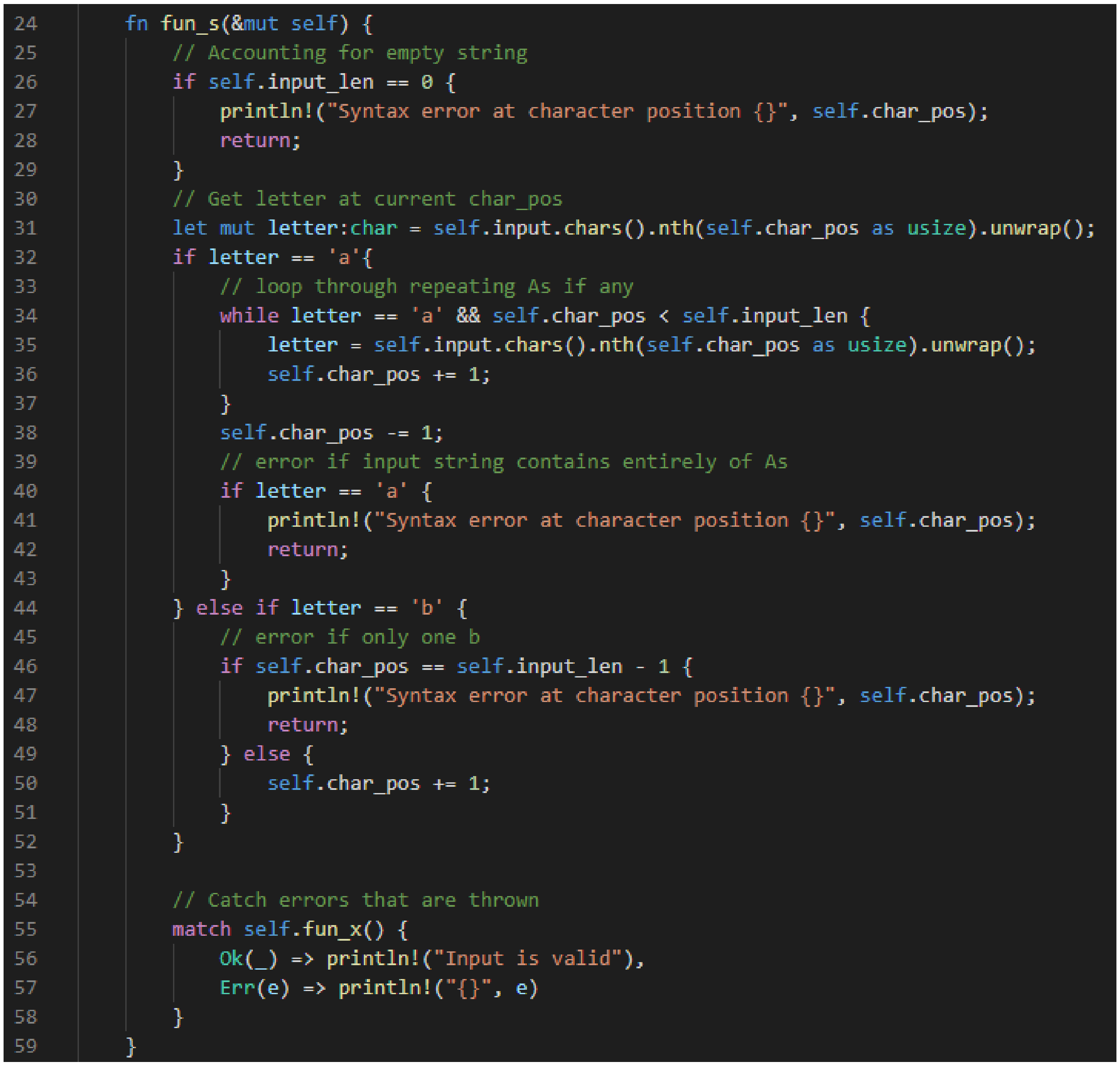
1. Threw errors using custom error, but I never caught the error. To fix, I need to add a match statement at the end of fun_s()
   - 

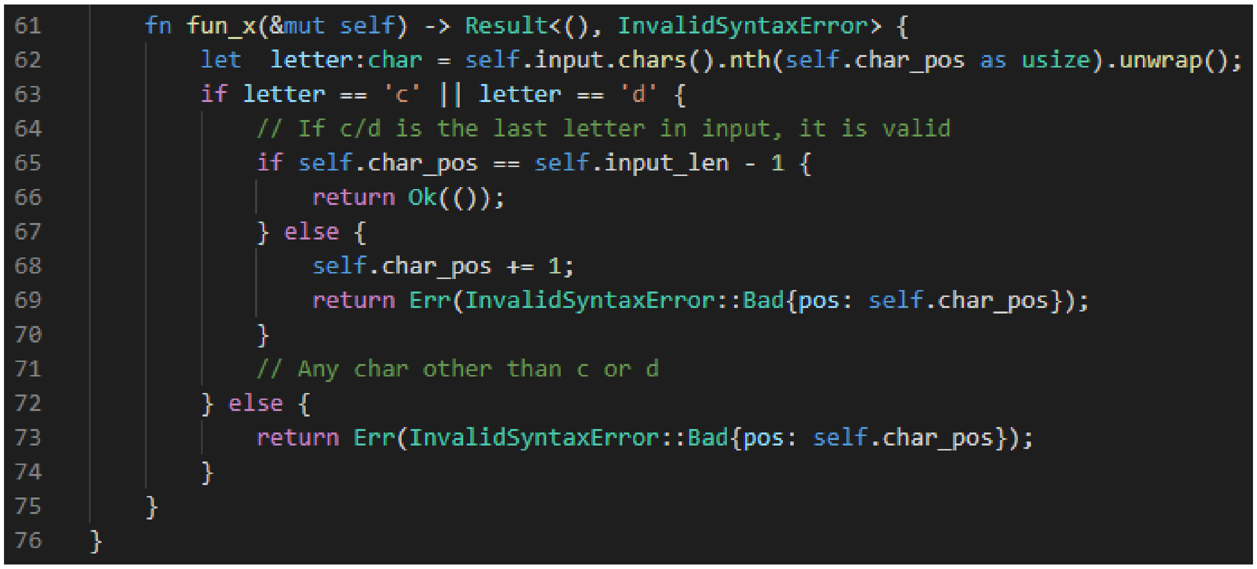
```rust
match self.fun_x() {
    Ok(_) => println!("Input is valid"),
    Err(e) => println!("{}", e)
}
```

Add Documentation



```rust
extern crate custom_error;
use custom_error::custom_error;

custom_error!{InvalidSyntaxError
    Bad{pos:i32} = "Syntax error at character position {pos}"
}

struct SimpleParser {
    input: String,
    char_pos: i32,
    input_len: i32
}

impl SimpleParser {
    fn new(user_input: &str) -> SimpleParser {
        SimpleParser {
            // initialize attributes
            input: user_input.to_string(),
            char_pos: 0,
            input_len: user_input.to_string().chars().count() as i32
        }
    }

```


```rust
    fn fun_s(&mut self) {
        // Accounting for empty string
        if self.input_len == 0 {
            println!("Syntax error at character position {}", self.char_pos);
            return;
        }
        // Get letter at current char_pos
        let mut letter:char = self.input.chars().nth(self.char_pos as usize).unwrap();
        if letter == 'a'{
            // loop through repeating As if any
            while letter == 'a' && self.char_pos < self.input_len {
                letter = self.input.chars().nth(self.char_pos as usize).unwrap();
                self.char_pos += 1;
            }
            self.char_pos -= 1;
            // error if input string contains entirely of As
            if letter == 'a' {
                println!("Syntax error at character position {}", self.char_pos);
                return;
            }
        } else if letter == 'b' {
            // error if only one b
            if self.char_pos == self.input_len - 1 {
                println!("Syntax error at character position {}", self.char_pos);
                return;
            } else {
                self.char_pos += 1;
            }
        }

        // Catch errors that are thrown
        match self.fun_x() {
            Ok(_) => println!("Input is valid"),
            Err(e) => println!("{}", e)
        }
    }
```



```rust
    fn fun_x(&mut self) -> Result<(), InvalidSyntaxError> {
        let  letter:char = self.input.chars().nth(self.char_pos as usize).unwrap();
        if letter == 'c' || letter == 'd' {
            // If c/d is the last letter in input, it is valid
            if self.char_pos == self.input_len - 1 {
                return Ok(());
            } else {
                self.char_pos += 1;
                return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
            }
            // Any char other than c or d
        } else {
            return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
        }
    }
}
```