ECS 140A Homework 2 – Problem 2

1 Python

Step 1: Algorithm/Pseudocode

```
class CStream:
    init(fname):
        # set file name
        # set line_num and char_pos to -1
        # read entire file into a string, call this fileContent for now
        # set index of current place in string to -1, call this val currIndex for now

    more_available():
        # if currIndex is at the last char (strlen - 1), return false
        # else return true

    get_cur_char():
        # return character at currIndex in fileContent

    get_next_char():
        # increment currIndex and char_pos
        # get character at currIndex in fileContent
        # if newline character
            # increment line_num by 1, char_pos set to 0
            # increment again to get non-newline character

    peek_next_char():
        # call peek_ahead_char(0)

    peek_ahead_char(k):
        # k=0 returns next character
        # return character at currIndex+k+1 in fileContent
        # +1 to make k=0 next character
        # if newline character, keep incrementing until get non-newline character
```

Step 2: Actual Code

```
class CStream:
    def __init__(self, fname):
        self.fname = fname
        self.line_num = -1
        self.char_pos = -1
        self.content = ""
        with open(fname, ’r’) as fp:
            self.content = fp.read()
        self.content_pos = -1

    def more_available(self):
        if self.content_pos == len(self.content) - 1:
            return False
        return True

    def get_cur_char(self):
        return self.content[self.content_pos]

    def get_next_char(self):
        if self.line_num == -1:
            self.line_num += 1
        self.char_pos += 1
        self.content_pos += 1
        while True:
            ch = self.content[self.content_pos]
            if ch == ’\n’:
                self.char_pos = 0
                self.line_num += 1
                self.content_pos += 1
            else:
                return ch

    def peek_next_char(self):
        return self.peek_ahead_char(0)

    def peek_ahead_char(self, k):
        i = 1
        while True:
            ch = self.content[self.content_pos + k + i]
            if ch == ’\n’:
                self.char_pos = 0
                self.line_num += 1
                self.content_pos += 1
                i += 1
            else:
                return ch
```

Step 3: Working Code

There were no syntax errors, so the initial working code was the same as the previous step.

Step 4: Debug Process

In the example test in the homework 2 prompt, step 4 returned the character ‘a’ instead of ‘t’ when I ran my code. This is because the logic skips straight to k characters ahead and does not know if there are any newline characters skipped. To fix this, I peeked ahead by looking at characters one at a time until the kth character to capture any newline characters missed. I used separate counters: one to keep track of non-newline characters and one to keep track of newline characters.

Fixed code:

```
def peek_ahead_char(self, k):
    count = 0
    i = 1
    ch = ''
    while count <= k:
        ch = self.content[self.content_pos + count + i]
        if ch == '\n':
            i += 1
        else:
            count += 1
    return ch
```

Step 5: Add Documentation

```
class CStream:
    def __init__(self, fname):
        ''' Initializer '''
        self.fname = fname
        self.line_num = -1
        self.char_pos = -1
        self.content = ""
        with open(fname, 'r') as fp:
            self.content = fp.read() # read entire file into string
        self.content_pos = -1 # current index in string

    def more_available(self):
        ''' Return true if have more contents to read in file '''
        # if current position not last character
        if self.content_pos == len(self.content) - 1:
            return False
        return True

    def get_cur_char(self):
        ''' Return character at current position in file '''
        return self.content[self.content_pos] # return character at current position

    def get_next_char(self):
        ''' Get next character in file and move position '''
        if self.line_num == -1: # check initial value of line_num
            self.line_num += 1
        self.char_pos += 1
        self.content_pos += 1
        while True:
            ch = self.content[self.content_pos]
            if ch == '\n':
                self.char_pos = 0 # reset char position to 0 when on new line
                self.line_num += 1
                self.content_pos += 1
            else:
                return ch

    def peek_next_char(self):
        ''' Get next character in file without changing position in file '''
        # peek_ahead_char(0) is same as peek_next_char()
        return self.peek_ahead_char(0)
```

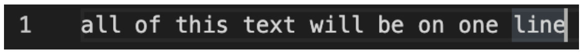
```
def peek_ahead_char(self, k):
    ''' Get kth character ahead in file without changing position '''
    count = 0 # keep track of non-newline char
    i = 1 # keep track of newline char; default is 1 to make k=0 next char
    ch = ''
    while count <= k:
        ch = self.content[self.content_pos + count + i] # check character ahead
        if ch == '\n':
            i += 1
        else:
            count += 1
    return ch
```

Step 6: Extra Test Cases Used

Note: Test 1 is considered the given test in the homework prompt.

Test 2

File contents:

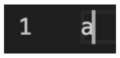


```
all of this text will be on one line
```

1. f.peek_next_char()
2. f.peek_ahead_char(7)
3. f.get_next_char()
4. f.peek_ahead_char(6)
5. f.get_cur_char()
6. f.more_available()

Test 3

File contents:

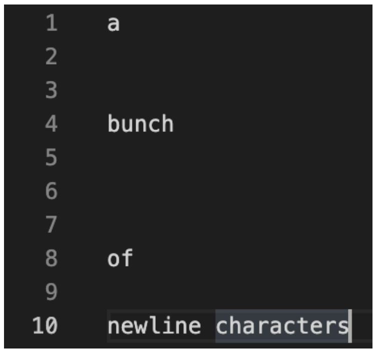


```
a
```

1. f.more_available()
2. f.get_next_char()
3. f.more_available()

Test 4

File contents:



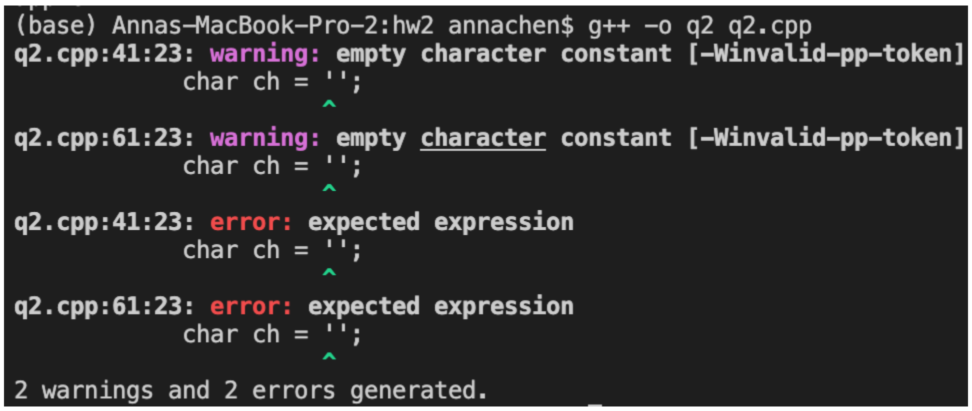
```
a


bunch



of

newline characters
```

1. f.get_next_char()
2. f.peek_ahead_char(0)
3. f.peek_ahead_char(4)
4. f.get_next_char()
5. f.get_next_char()
6. f.get_next_char()
7. f.get_next_char()
8. f.get_next_char()
9. f.peek_next_char()
10. f.peek_ahead_char(3)
11. f.more_available()

2 C++

Step 2: Actual Code

```
class CStream {
    public:
        std::string fname;
        int line_num;
        int char_pos;
        std::string content;
        int content_pos;

        CStream(std::string file_name) {
            fname = file_name;
            line_num = -1;
            char_pos = -1;
            std::ifstream t(fname);
            std::stringstream buffer;
            buffer << t.rdbuf();
            content = buffer.str();
            std::cout << content << std::endl;
            content_pos = -1;
        }

        bool more_available() {
            if (content_pos == content.length())
                return false;
            return true;
        }

        char get_cur_char() {
            return content[content_pos];
        }

        char get_next_char() {
            if (line_num == -1) {
                line_num++;
            }
            char_pos++;
            content_pos++;
            char ch = '';
            while (1) {
                ch = content[content_pos];
                if (ch == '\n') {
                    char_pos = 0;
                    line_num++;
                    content_pos++;
                } else {
                    return ch;
                }
            }
        }
        char peek_next_char() {
            return peek_ahead_char(0);
        }

        char peek_ahead_char(int k) {
            int count = 0;
            int i = 1;
            char ch = '';
            while (count <= k) {
                ch = content[content_pos + count + i];
                if (ch == '\n') {
                    i++;
                } else {
                    count++;
                }
            }
            return ch;
        }
};
```



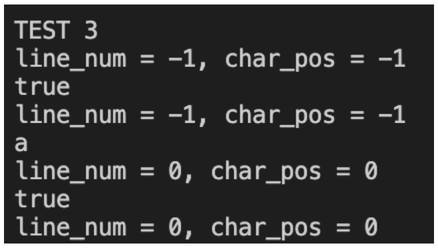
```
(base) Annas-MacBook-Pro-2:hw2 annachen$ g++ -o q2 q2.cpp
q2.cpp:41:23: warning: empty character constant [-Winvalid-pp-token]
            char ch = '';
                      ^
q2.cpp:61:23: warning: empty character constant [-Winvalid-pp-token]
            char ch = '';
                      ^
q2.cpp:41:23: error: expected expression
            char ch = '';
                      ^
q2.cpp:61:23: error: expected expression
            char ch = '';
                      ^
2 warnings and 2 errors generated.
```

This error was raised because there is no empty character. To fix this, I initialized `ch` to be a space.

Step 3: Working Code

```
class CStream {
    public:
        std::string fname;
        int line_num;
        int char_pos;
        std::string content;
        int content_pos;

        CStream(std::string file_name) {
            fname = file_name;
            line_num = -1;
            char_pos = -1;
            std::ifstream t(fname);
            std::stringstream buffer;
            buffer << t.rdbuf();
            content = buffer.str();
            content_pos = -1;
        }

        bool more_available() {
            if (content_pos == content.length())
                return false;
            return true;
        }

        char get_cur_char() {
            return content[content_pos];
        }

        char get_next_char() {
            if (line_num == -1) {
                line_num++;
            }
            char_pos++;
            content_pos++;
            char ch = ' ';
            while (1) {
                ch = content[content_pos];
                if (ch == '\n') {
                    char_pos = 0;
                    line_num++;
                    content_pos++;
                } else {
                    return ch;
                }
            }
        }
        char peek_next_char() {
            return peek_ahead_char(0);
        }

        char peek_ahead_char(int k) {
            int count = 0;
            int i = 1;
            char ch = '';
            while (count <= k) {
                ch = content[content_pos + count + i];
                if (ch == '\n') {
                    i++;
                } else {
                    count++;
                }
            }
            return ch;
        }
};
```

Step 4: Debug Process



```
TEST 3
line_num = -1, char_pos = -1
true
line_num = -1, char_pos = -1
a
line_num = 0, char_pos = 0
true
line_num = 0, char_pos = 0
```

In this test case where there is only one character in the file, the boolean value printed out should be false instead. This is because I forgot to subtract 1 from the string length.

Fixed code:

```
        bool more_available() {
            if (content_pos == content.length() - 1)
                return false;
            return true;
        }
```

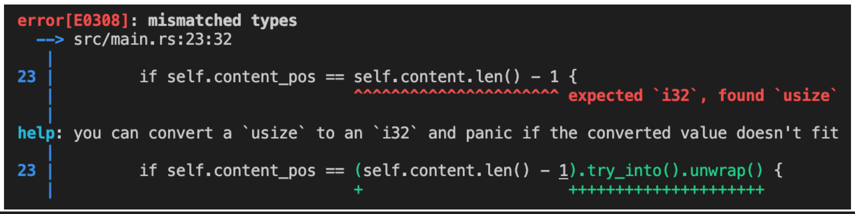
Step 5: Add Documentation

```
class CStream {
    public:
        std::string fname;
        int line_num;
        int char_pos;
        std::string content;
        int content_pos;

        // Constructor
        CStream(std::string file_name) {
            fname = file_name;
            line_num = -1;
            char_pos = -1;

            // Source:
            // https://stackoverflow.com/questions/2602013/read-whole-ascii-file-into-c-stdstring
            std::ifstream t(fname);
            std::stringstream buffer;
            buffer << t.rdbuf();
            content = buffer.str();
            content_pos = -1;
        }

        // Return true if have more contents to read in file
        bool more_available() {
            // if current position not last character
            if (content_pos == content.length() - 1)
                return false;
            return true;
        }

        // Return character at current position in file
        char get_cur_char() {
            return content[content_pos];
        }

        // Get next character in file and move position
        char get_next_char() {
            if (line_num == -1) { // check initial value of line_num
                line_num++;
            }
            char_pos++;
            content_pos++;
            char ch = ' ';
            while (1) {
                ch = content[content_pos];
                if (ch == '\n') {
                    char_pos = 0; // reset char position to 0 when on new line
                    line_num++;
                    content_pos++;
                } else {
                    return ch;
                }
            }
        }
        // Get next character in file without changing position in file
        char peek_next_char() {
            // peek_ahead_char(0) is same as peek_next_char()
            return peek_ahead_char(0);
        }

        // Get kth character ahead in file without changing position
        char peek_ahead_char(int k) {
            int count = 0; // track non-newline chars
            int i = 1; // track newline chars; default is 1 to make k=0 next char
            char ch = ' ';
            while (count <= k) {
                ch = content[content_pos + count + i]; // check character ahead
                if (ch == '\n') {
                    i++;
                } else {
                    count++;
                }
            }
            return ch;
        }
};
```

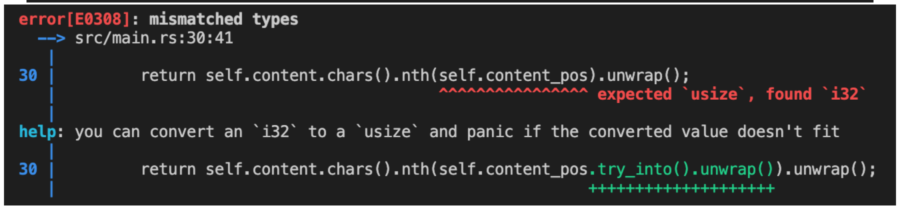
3 Rust

Step 2: Actual Code

```
struct CStream {
    fname: String,
    line_num: i32,
    char_pos: i32,
    content: String,
    content_pos: i32
}

impl CStream {
    fn new(file_name: String) -> CStream {
        return CStream {
            fname: file_name,
            line_num: -1,
            char_pos: -1,
            content: fs::read_to_string(file_name).expect("Unable to read file"),
            content_pos: -1
        };
    }

    fn more_available(&self) -> bool {
        if self.content_pos == self.content.len() - 1 {
            return false;
        }
        return true;
    }

    fn get_cur_char(&self) -> char {
        return self.content.chars().nth(self.content_pos).unwrap();
    }

    fn get_next_char(&self) -> char {
        if self.line_num == -1 {
            self.line_num += 1;
        }
        self.char_pos += 1;
        self.content_pos += 1;
        let mut ch = ' ';
        while true {
            ch = self.content.chars().nth(self.content_pos).unwrap();
            if ch == '\n' {
                self.char_pos = 0;
                self.line_num += 1;
                self.content_pos += 1
            } else {
                return ch;
            }
        }
    }
    fn peek_next_char(&self) -> char {
        return self.peek_ahead_char(0);
    }

    fn peek_ahead_char(&self, k: i32) -> char {
        let mut count = 0;
        let mut i = 1;
        let mut ch = ' ';
        while count <= k {
            ch = self.content.chars().nth(self.content_pos + count + i).unwrap();
            if ch == '\n' {
                i += 1;
            } else {
                count += 1;
            }
        }
        return ch;
    }
}
```

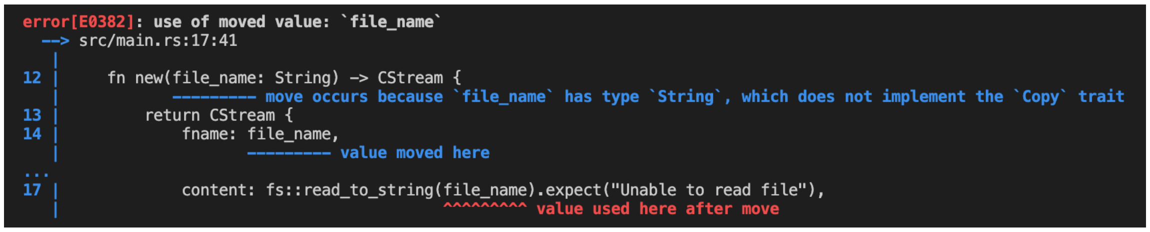


```
error[E0308]: mismatched types
  --> src/main.rs:23:32
   |
23 |         if self.content_pos == self.content.len() - 1 {
   |                                ^^^^^^^^^^^^^^^^^^^^^^ expected `i32`, found `usize`
   |
help: you can convert a `usize` to an `i32` and panic if the converted value doesn't fit
   |
23 |         if self.content_pos == (self.content.len() - 1).try_into().unwrap() {
   |                                +                      +++++++++++++++++++++
```



```
error[E0308]: mismatched types
  --> src/main.rs:30:41
   |
30 |         return self.content.chars().nth(self.content_pos).unwrap();
   |                                         ^^^^^^^^^^^^^^^^ expected `usize`, found `i32`
   |
help: you can convert an `i32` to a `usize` and panic if the converted value doesn't fit
   |
30 |         return self.content.chars().nth(self.content_pos.try_into().unwrap()).unwrap();
   |                                                         ++++++++++++++++++++
```

These errors were raised because if a mismatch between `i32` and usize types. To fix, I used the `as` for casting types.



```
error[E0382]: use of moved value: `file_name`
  --> src/main.rs:17:41
   |
12 |     fn new(file_name: String) -> CStream {
   |            --------- move occurs because `file_name` has type `String`, which does not implement the `Copy` trait
13 |         return CStream {
14 |             fname: file_name,
   |                    --------- value moved here
...
17 |             content: fs::read_to_string(file_name).expect("Unable to read file"),
   |                                         ^^^^^^^^^ value used here after move
```

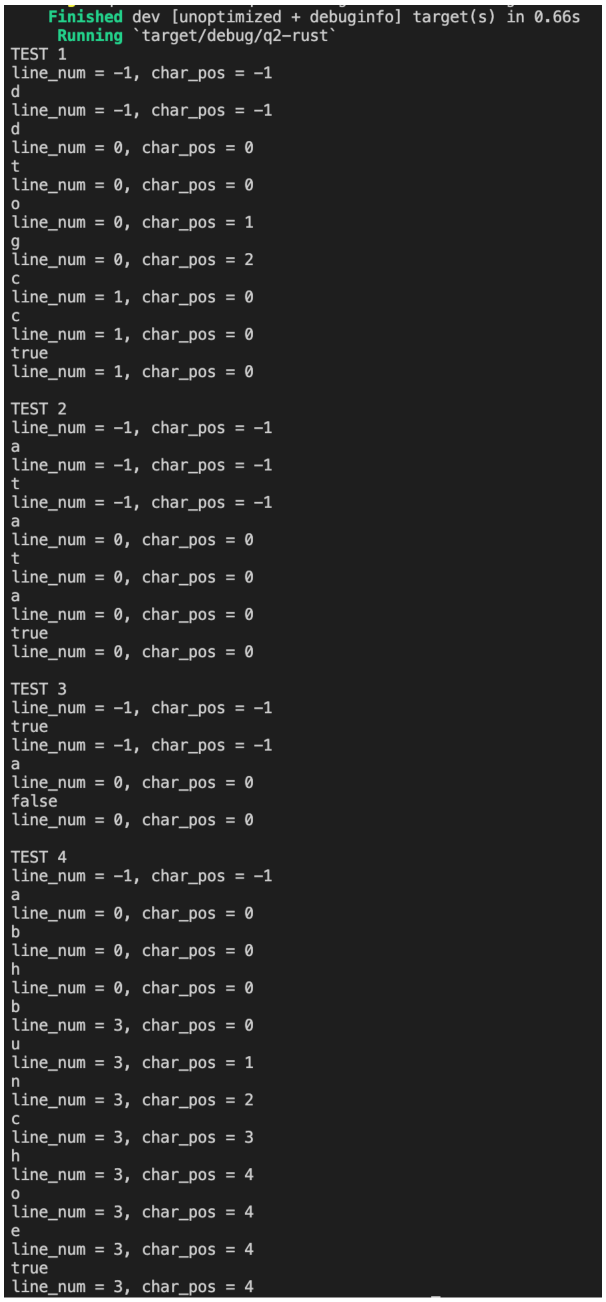
Due to `file_name` potentially being moved in memory, I cloned `file_name` to fix this error.

Step 3: Working Code

```
struct CStream {
    fname: String,
    line_num: i32,
    char_pos: i32,
    content: String,
    content_pos: i32
}

impl CStream {
    fn new(file_name: String) -> CStream {
        return CStream {
            fname: file_name.clone(),
            line_num: -1,
            char_pos: -1,
            content: fs::read_to_string(file_name.clone()).expect("Unable to read file"),
            content_pos: -1
        };
    }

    fn more_available(&self) -> bool {
        if self.content_pos == (self.content.len() - 1) as i32 {
            return false;
        }
        return true;
    }

    fn get_cur_char(&self) -> char {
        return self.content.chars().nth(self.content_pos as usize).unwrap();
    }

    fn get_next_char(&mut self) -> char {
        if self.line_num == -1 {
            self.line_num += 1;
        }
        self.char_pos += 1;
        self.content_pos += 1;
        let mut ch = ' ';
        while true {
            ch = self.content.chars().nth(self.content_pos as usize).unwrap();
            if ch == '\n' {
                self.char_pos = 0;
                self.line_num += 1;
                self.content_pos += 1
            } else {
                return ch;
            }
        }
        return ch;
    }
    fn peek_next_char(&mut self) -> char {
        return self.peek_ahead_char(0);
    }

    fn peek_ahead_char(&mut self, k: i32) -> char {
        let mut count = 0;
        let mut i = 1;
        let mut ch = ' ';
        while count <= k {
            ch = self.content.chars().nth((self.content_pos + count + i) as usize).unwrap();
            if ch == '\n' {
                i += 1;
            } else {
                count += 1;
            }
        }
        return ch;
    }
}
```

Step 4: Debug Process

All test cases passed and behaved as expected.



```
    Finished dev [unoptimized + debuginfo] target(s) in 0.66s
     Running `target/debug/q2-rust`
TEST 1
line_num = -1, char_pos = -1
d
line_num = -1, char_pos = -1
d
line_num = 0, char_pos = 0
t
line_num = 0, char_pos = 0
o
line_num = 0, char_pos = 1
g
line_num = 0, char_pos = 2
c
line_num = 1, char_pos = 0
c
line_num = 1, char_pos = 0
true
line_num = 1, char_pos = 0

TEST 2
line_num = -1, char_pos = -1
a
line_num = -1, char_pos = -1
t
line_num = -1, char_pos = -1
a
line_num = 0, char_pos = 0
t
line_num = 0, char_pos = 0
a
line_num = 0, char_pos = 0
true
line_num = 0, char_pos = 0

TEST 3
line_num = -1, char_pos = -1
true
line_num = -1, char_pos = -1
a
line_num = 0, char_pos = 0
false
line_num = 0, char_pos = 0

TEST 4
line_num = -1, char_pos = -1
a
line_num = 0, char_pos = 0
b
line_num = 0, char_pos = 0
h
line_num = 0, char_pos = 0
b
line_num = 3, char_pos = 0
u
line_num = 3, char_pos = 1
n
line_num = 3, char_pos = 2
c
line_num = 3, char_pos = 3
h
line_num = 3, char_pos = 4
o
line_num = 3, char_pos = 4
e
line_num = 3, char_pos = 4
true
line_num = 3, char_pos = 4
```

Step 5: Add Documentation

```
struct CStream {
    fname: String,
    line_num: i32,
    char_pos: i32,
    content: String,
    content_pos: i32
}

impl CStream {
    // Initializer
    // Source for reading file into a string:
    // https://stackoverflow.com/questions/31192956/whats-the-de-facto-way-of-reading-
    // and-writing-files-in-rust-1-x
    fn new(file_name: String) -> CStream {
        return CStream {
            fname: file_name.clone(),
            line_num: -1,
            char_pos: -1,
            content: fs::read_to_string(file_name.clone()).expect("Unable to read file"),
            content_pos: -1
        };
    }

    // Return true if have more contents to read in file
    fn more_available(&self) -> bool {
        // if current position not last character
        if self.content_pos == (self.content.len() - 1) as i32 {
            return false;
        }
        return true;
    }

    // Return character at current position in file
    fn get_cur_char(&self) -> char {
        return self.content.chars().nth(self.content_pos as usize).unwrap();
    }

    // Get next character in file and move position
    fn get_next_char(&mut self) -> char {
        if self.line_num == -1 { // check initial value of line_num
            self.line_num += 1;
        }
        self.char_pos += 1;
        self.content_pos += 1;
        let mut ch = ' ';
        loop { // warning message in compiler suggested using loop for infinite loop
            ch = self.content.chars().nth(self.content_pos as usize).unwrap();
            if ch == '\n' {
                self.char_pos = 0; // reset char position to 0 when on new line
                self.line_num += 1;
                self.content_pos += 1
            } else {
                return ch;
            }
        }
        return ch;
    }

    // Get next character in file without changing position in file
    fn peek_next_char(&mut self) -> char {
        // peek_ahead_char(0) same as peek_next_char()
        return self.peek_ahead_char(0);
    }

    // Get kth character ahead in file without changing position
    fn peek_ahead_char(&mut self, k: i32) -> char {
        let mut count = 0; // track non-newline chars
        let mut i = 1; // track newline chars
        let mut ch = ' ';
        while count <= k {
            // check character ahead
            ch = self.content.chars().nth((self.content_pos + count + i) as usize).unwrap();
            if ch == '\n' {
                i += 1;
            } else {
                count += 1;
            }
        }
        return ch;
    }
}
```