

# ECS 140A Homework 5 – Problem 2

## 1 Haskell

### Step 1: Algorithm/Pseudocode

```
cmpStr (string, string) -> string
    find lcp of the two strings passed
    check if char from both strings are equal
        if equal, add onto resulting prefix and recurse on the rest of the strings
        if not, current prefix is longest between the two strings

lcp (string array) -> string
    use fold function to apply cmpStrings to and reduce string array to lcp
```

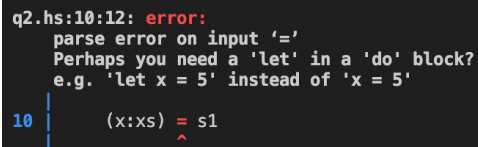
### Step 2: Actual Code

```
cmpStr :: String -> String -> String
cmpStr s1 s2 = do
    (x:xs) = s1
    (y:ys) = s2
    if x == y
        then x ++ cmpStr xs ys
        else ''

lcp :: [String] -> String
lcp arr = do
    let dummy = head arr
    foldr cmpStr dummy arr

main = do
    print (lcp ["apple", "app", "apple", "appl"])
```

### Syntax Error 1



The screenshot shows a Haskell compiler error in a file named q2.hs at line 10, column 12. The error message is: "error: parse error on input '=' Perhaps you need a 'let' in a 'do' block? e.g. 'let x = 5' instead of 'x = 5'". Below the message, the code snippet is shown: "10 | (x:xs) = s1", with a red caret pointing to the equals sign.

This error was raised because I did not properly define the variables for the pattern matching. I fixed this by using the `let` keyword.

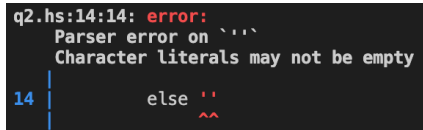
Incorrect code:

```
(x:xs) = s1
(y:ys) = s2
```

Correct code:

```
let (x:xs) = s1
let (y:ys) = s2
```

## Syntax Error 2



```
q2.hs:14:14: error:
  Parser error on `''`
  Character literals may not be empty
14 |         else ''
   |               ^
```

This error was raised because there is no such thing as an empty character. Instead, because strings are represented as a list of characters in Haskell, I used an empty list as a solution.

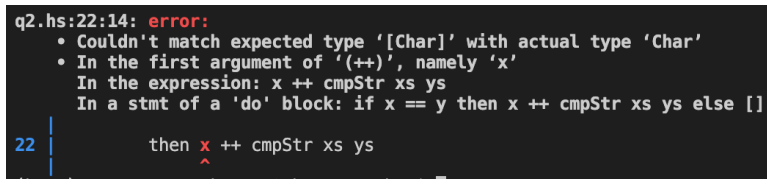
Incorrect code:

```
if x == y
  then x ++ cmpStr xs ys
  else ''
```

Correct code:

```
if x == y
  then x ++ cmpStr xs ys
  else []
```

## Syntax Error 3



```
q2.hs:22:14: error:
  • Couldn't match expected type '[Char]' with actual type 'Char'
  • In the first argument of '(++)', namely 'x'
    In the expression: x ++ cmpStr xs ys
    In a stmt of a 'do' block: if x == y then x ++ cmpStr xs ys else []
22 |         then x ++ cmpStr xs ys
   |               ^
```

This error was raised because Char type does not match String type. I wanted to keep adding characters onto the resulting string to be returned. The solution to this was to enclose the character into a list.

Incorrect code:

```
if x == y
  then [x] ++ cmpStr xs ys
  else []
```

Correct code:

```
if x == y
  then [x] ++ cmpStr xs ys
  else []
```

### Step 3: Working Code

```
cmpStr :: String -> String -> String
cmpStr s1 s2 = do
  let (x:xs) = s1
  let (y:ys) = s2
  if x == y
    then [x] ++ cmpStr xs ys
    else []

lcp :: [String] -> String
lcp arr = do
  let dummy = head arr
  foldr cmpStr dummy arr

main = do
  print (lcp ["apple", "app", "apple", "appl"])
```

### Step 4: Debug Process

#### Bug 1

```
q2: q2.hs:10:9-19: Non-exhaustive patterns in y : ys
```

I ran into this message when running the executable. This message arose due to a pattern match failure. The only way this could have occurred was with an empty string. I did not account for when all the characters in one string are used and have no more to recurse on. To fix this, I added a condition in the helper function to check if either string was empty and to handle the case accordingly.

Buggy code:

```
cmpStr :: String -> String -> String
cmpStr s1 s2 = do
  let (x:xs) = s1
  let (y:ys) = s2
  if x == y
    then [x] ++ cmpStr xs ys
    else []
```

Fixed code:

```

cmpStr :: String -> String -> String
cmpStr s1 s2 = do
  if s1 == [] || s2 == []
  then []
  else do
    let (x:xs) = s1
    let (y:ys) = s2
    if x == y
    then [x] ++ cmpStr xs ys
    else []

```

## Bug 2

**q2: Prelude.head: empty list**

This result was given when I tried testing an empty list. Because the list is empty, there is no head (first element) that we can retrieve. To fix this, I added a condition to check if the list is empty and to handle the case accordingly.

Buggy code:

```

lcp :: [String] -> String
lcp arr = do
  let dummy = head arr
  foldr cmpStr dummy arr

```

Fixed code:

```

lcp :: [String] -> String
lcp arr = do
  if arr == []
  then []
  else do
    let dummy = head arr
    foldr cmpStr dummy arr

```

After fixing the above bugs, all given and additional test cases passed and behaved as expected. Additional test cases can be found at the end of this report. Below are the output results:

```

(base) Annas-MacBook-Pro-2:hw5 annachen$ ghc q2.hs
[1 of 1] Compiling Main             ( q2.hs, q2.o )
Linking q2 ...
(base) Annas-MacBook-Pro-2:hw5 annachen$ ./q2
"ap"
""
""
"abc"
""
"zz"
"bamboo"
"bamb"

```

## Step 5: Add Documentation

```
-- Find the longest common prefix between 2 strings
cmpStr :: String -> String -> String
cmpStr s1 s2 = do
    if s1 == [] || s2 == [] -- empty string check
    then []
    else do
        -- compare first chars of both strings
        let (x:xs) = s1
        let (y:ys) = s2
        if x == y
        then [x] ++ cmpStr xs ys -- store char and recurse on rest of both strings
        else [] -- no chars to add on if mismatch

-- Uses helper function to get longest common prefix of all strings
lcp :: [String] -> String
lcp arr = do
    if arr == [] -- empty list check
    then []
    else do
        let dummy = head arr
        foldr cmpStr dummy arr -- use fold to apply helper function on the list of strings

main = do
    -- given case
    print (lcp ["apple", "app", "apple", "appl"])

    -- additional cases
    print (lcp [])
    print (lcp [""])
    print (lcp ["abc"])
    print (lcp ["abc", "xyz"])
    print (lcp ["zzzzz", "zz", "zzzz"])
    print (lcp ["bamboo", "bamboozled"])
    print (lcp ["bamboo", "bamboozled", "bambam"])
```

## Step 6: Extra Test Cases Used

- []
- [""]
- ["abc"]
- ["abc", "xyz"]
- ["zzzzz", "zz", "zzzz"]
- ["bamboo", "bamboozled"]
- ["bamboo", "bamboozled", "bambam"]