CSE-4321 Software Testing Final Project - Faults and Corrections Report

Maximum Effort: Elliot Mai & Benjamin Niccum

December 2, 2024

Introduction

This report identifies the 13 faults in the original `Printtokens` program, along with their corrections and explanations. Each fault is paired with the original and corrected code to illustrate the changes made and their impact on the program.

Faults and Corrections

1. `open_character_stream`

   - **Fault:** Incorrect default for handling null input streams.

   Listing 1: Original Code

   ```
   return new BufferedReader(new StringReader(""));
   ```

   - **Correction:** Set up `System.in` as the default input stream.

   Listing 2: Corrected Code

   ```
   br = new BufferedReader(new InputStreamReader(System.in));
   ```

   - **Explanation:** This change ensures that the program accepts input from the console when no file is specified.

2. `get_token`

   - **Fault 1:** Unused variables for token processing.

Listing 3: Original Code

```
int i=0,j;
```

- **Correction 1:** Removed unused variables.

Listing 4: Corrected Code

```
// Removed unnecessary variables
```

- **Fault 2:** Flipped ID numbers for strings and comments.

Listing 5: Original Code

```
if(ch =='"')id=2;
if(ch ==59)id=1;
```

- **Correction 2:** Corrected ID numbers for proper classification.

Listing 6: Corrected Code

```
if (ch == '"')
    id = 1;
if (ch == 59)
    id = 2;
```

- **Explanation:** These changes ensure accurate classification of string and comment tokens while removing redundant variables.

3. `token_type`

- **Fault:** Misclassification of string constants due to ordering.

Listing 7: Original Code

```
if (is_keyword(tok))return (keyword);
if(is_spec_symbol(tok.charAt(0)))return(spec_symbol);
if (is_identifier(tok))return (identifier);
if (is_num_constant(tok))return (num_constant);
if (is_str_constant(tok))return (str_constant);
```

- **Correction:** Moved special symbols check after string constant check.

Listing 8: Corrected Code

```
if (is_identifier(tok))
```

```
    return (identifier);
if (is_num_constant(tok))
    return (num_constant);
if (is_str_constant(tok))
    return (str_constant);
if (is_spec_symbol(tok.charAt(0)))
    return (spec_symbol);
```

- **Explanation:** This ensures that string constants are identified before special symbols, preventing misclassification.

4. `is_char_constant`

- **Fault:** Allowed invalid character constants of incorrect length.

Listing 9: Original Code

```
if (str.length() > 2 || str.charAt(0)=='#')
```

- **Correction:** Added length validation for exactly 2 characters.

Listing 10: Corrected Code

```
if (str.length() == 2 && str.charAt(0) == '#' && Character.
   isLetter(str.charAt(1)))
```

- **Explanation:** Ensures that only valid character constants are accepted.

5. `is_num_constant`

- **Fault:** Accessed out-of-bounds indices.

Listing 11: Original Code

```
if(Character.isDigit(str.charAt(i+1)))
```

- **Correction:** Added boundary checks before accessing indices.

Listing 12: Corrected Code

```
if (Character.isDigit(str.charAt(i)))
```

- **Explanation:** Prevents runtime errors when accessing invalid indices.

6. `is_str_constant`

- **Fault:** Allowed unclosed strings to pass as valid.

Listing 13: Original Code

```
if (str.charAt(0) =='"')
```

- **Correction:** Validated that strings are properly closed.

Listing 14: Corrected Code

```
(str.length() >= 3 && str.charAt(0) == '"' && str.charAt(str.
   length() - 1) == '"')
```

- **Fault:** Did not check for intermediate quotes.

Listing 15: Original Code

```
{ while (i < str.length() && str.charAt(i)!='\0')
    { if(str.charAt(i)=='"' )
        return true;
```

- **Correction:** Added validation to disallow quotes in the middle of strings.

Listing 16: Corrected Code

```
while (i < str.length() - 1 && str.charAt(i) != '\0') {
    if (str.charAt(i) == '"')
        return false; /* meet an intermediate '"' */
}
```

- **Explanation:** Ensures strings are properly closed without internal quotes.

7. `is_identifier`

- **Fault:** Return values flipped.

Listing 17: Original Code

```
    if(Character.isLetter(str.charAt(i)) || Character.isDigit(
       str.charAt(i)))
        i++;
    else
        return false;
    }       /* end WHILE */
    return false;
```

```
    }
else
    return true;
```

- **Correction:** Corrected return values to ensure correct classification.

Listing 18: Corrected Code

```
    if(Character.isLetter(str.charAt(i)) || Character.isDigit(
       str.charAt(i)))
        i++;
    else
        return false;
    }       /* end WHILE */
    return true;
    }
else
    return false;
```

- **Explanation:** These changes ensure accurate classification of identifiers while not falsely classifying other types as identifiers.

8. `print_spec_symbol`

- **Fault:** Checked wrong paranthase.

Listing 19: Original Code

```
if (str.equals(")"))
```

- **Correction:** Corrected to check both parenthase types.

Listing 20: Corrected Code

```
if (str.equals("("))
```

- **Explanation:** Ensures correct output of both parentheses.

9. `is_spec_symbol`

- **Fault:** Checked for invalid special symbols like `/`.

Listing 21: Original Code

```
if (c == '/')
```

- **Correction:** Removed invalid checks for `/`.

Listing 22: Corrected Code

```
// Removed check for '/'
```

- **Explanation:** Aligns the method with the definition of valid special symbols.

10. `main`

- **Fault:** Used `System.exit` prematurely.

Listing 23: Original Code

```
System.exit(0);
```

- **Correction:** Replaced with a `return` statement.

Listing 24: Corrected Code

```
// Removed line
```

- **Fault:** Incorrectly terminated the program instead of continuing to accept the token stream from the command line.

Listing 25: Original Code

```
// Not handled in the original code
```

- **Correction:** Added prompt for user input via `System.in`.

Listing 26: Corrected Code

```
if (fname == null) {
    System.out.println("Please␣type␣tokens␣to␣classify␣(press␣
        Ctrl+C␣to␣end):");
}
```

- **Explanation:** Improves usability by handling null filenames and providing clear user guidance.

Conclusion

The 13 faults identified in the original `Printtokens` program were corrected to improve token classification, error handling, and program robustness. These corrections ensure the program operates reliably across all scenarios.