Assignment Part 4 – Lex and Yacc

# Program Testing Procedure

Run program, try various inputs and observe parsed and sorted list data. Focus on parsing behaviour; I assume std::sort() works correctly. Make sure all valid syntax is accepted, but also that all invalid syntax is rejected.

|  |  |  |
| --- | --- | --- |
| **Test name** | **Test input** | **Expected behaviour** |
| Empty list | [] | Parsed: []  Sorted: [] |
| 1 element | [42] | Parsed: [42]  Sorted: [42] |
| 2 elements | [42, 7] | Parsed: [42, 7]  Sorted: [7, 42] |
| Many elements | [5, 3, 8, 3, 2, 9] | Parsed: [5, 3, 8, 3, 2, 9]  Sorted: [2, 3, 3, 5, 8, 9] |
| Negative numbers | [-1, -4, -2, -2] | Parsed: [-1, -4, -2, -2]  Sorted: [-4, -2, -2, -1] |
| Long numbers | [1234567, 86432, -10000, 999999] | Parsed: [1234567, 86432, -10000, 999999]  Sorted: [-10000, 86432, 999999, 1234567] |
| Various whitespace | [1,2, -1, -4,6,3 , 8 ] | Parsed: [1, 2, -1, -4, 6, 3, 8]  Sorted: [-4, -1, 1, 2, 3, 6, 8] |
| Leading and trailing whitespace | [87, 53, 26, 3, 5, 3] | Parsed: [87, 53, 26, 3, 5, 3]  Sorted: [3, 3, 5, 26, 53, 87] |
| Unmatched brace | [1, 2, 3, 4 | Syntax error |
| Extra comma 1 | [1, 2, 3, 4,] | Syntax error |
| Extra comma 2 | [,] | Syntax error |
| Non-numeric values | [a, foo, bar] | Syntax error |
| Just whitespace |  | Syntax error |
| Multiple lists | [1, 2, 3] [3, 2, 1] | Syntax error |
| Leading garbage | foo[1, 2, 3] | Syntax error |
| Trailing garbage | [1, 2, 3]foo | Syntax error |
| Malformed numbers | [1, 2, 43e] | Syntax error |

# Weekly Question

I would implement a symbol table with a hash table, mapping from names to AST nodes. To allow nested scopes, multiple hash tables could be used, stored in order of nesting, in list. To retrieve a name, the list is traversed in reverse (going up the tree of scopes) from the scope of the name usage, until a match is found.  
If it is important that symbols may be used only after declaration, then name resolution may be done as soon as possible (e.g. one-pass compilers such as C), or, the ordering of name declarations is stored in the symbol table as well.

# Reflection

Lex and Bison’s file structure violates the structured program principle, reducing readability and reliability. The grammar definitions are declarative in nature, with not much notion of control flow (that is controlled by the Lex and Yacc internals). However, the flow of control is important to consider since the rule actions manipulate shared global state. It can be difficult to discern the ordering of execution and how different parts of the code interact, leading to poor readability and increased probability of bugs. I had trouble making syntax error handling work as I desired, since it was difficult to determine what state the program was in when an error occurs.

The nature of Lex and Bison as code generators / preprocessors violates the syntactic consistency principle, reducing readability. The files consist of effectively three syntaxes, all intermixed: C or C++ code (with the ability to also use the C preprocessor!), grammar definition (including a regex-like syntax), and the supporting Lex and Yacc file structure and setup. Naturally, there are many syntax conflicts between them. In addition, C, and especially C++, by themselves are not particularly syntactically consistent. The result is that the semantics of any given portion of the program is highly context-dependent and potentially difficult to understand.