Compilers Assignment - 1

Vishal Vijay Devadiga (CS21BTECH11061)

Instructions

Folder Structure

There are 2 folders:

- `T1` consists of 3 public test cases with their `seq tokens` output and `C` output.

- `P1` contains the source files:-
  - `overview.pdf` is the report for the Assignment.
  - `lex_source_program.l` is the lex source file that generate lexical analyzed tokens in `TK` subfolder and translated C program in `TC` subfolder.
  - `run.sh` is a bash script that is to be used to operate the lexer.

Running the lexer

All input files **must** be placed in the same directory as `run.sh` and `lex_source_program.l`.

Edit the `run.sh` and set the Input file name as required.

The script can now be run by either right-clicking it and clicking `Run as a program` or by running `./run.sh` in a terminal.

Note: If the bash script is not executable, then run the following command: `chmod +x ./run.sh`

Output

All output sequence token files are stored in the `TK` subfolder.

All output translated C files are stored in `TC` subfolder.

Implementation

I have just added the `return` keyword, to maintain consistency with the return type of the functions.

Some implementation details:

- Ignored Whitespace and printed it to C file as it is.
- Handling Strings with escape characters (Example: `"Hello \"World"`). Similarly handled character literals.
- Ignored all "`:`" in my implementation as it does not matter in the language
- All signs are translated to their correct equivalents in the C Code

Some definition details:

- String/Chars: Quote literal –> (Any set of Characters except for individual quotes) –> Quote Literal. Using such a definition eases out keeping boolean variables for opening and closing quotes and also handling multiple individual tokens as 1 big token.
- Variables: Operator if required –> Any sequence of alphanumeric characters.
- Brackets: "[]" are converted to "()" if it occurs when a `if` condition is begin analyzed(Done using a bool variable), else it is converted to "{}". All other brackets remain the same.
- Labels: Printed only when it occurs after a `jump to` token appears. Done using a bool variable and look forward .
- Roots: Variable –> `_` –> 2 or Number –> `_` –> 2. This definition handles root in a optimal as there is no need to store previous Number/Variable and is handled as 1 big token consisting of 3 tokens.

Some present issues:

- "`_`" operator only handles square root cases. A solution to this could be to use pow function in C to represent `a _ b` as `pow (a , 1/b)` as this would be accurate to a extent.
- "`_`" operator only handles cases where there is a variable or number before it. For example, `v _ 2` is correctly translated. But cases where the operator must be applied over multiple variables in brackets is not handled in the program, as implementing it would require maintaining many previous tokens.
- Operators must have whitespaces surrounding it, leading to cases such as `a+ b` giving an error even if there is no ambiguity in the following example, that is `+` acts as a operator here.
- If there is no `do` keyword after the condition in `in case that`, then there are mismatches in the type of brackets printed after that line.
- Labels are not being translated in the C program due to the added complexity of labels to not be present outside function definitions and trying to maintain all previous labels.