EECS 337

Compiler Design

2009 Fall Semester

Homework 2

Revision 3: September 22, 2009

Due in class, Tuesday, September 29, 2009

Objectives: The objective of the second assignment is to develop a lexical analyzer. In future assignments, you will reuse the functions that you will develop in this assignment.

Lexical Analyzer: In this assignment, the instructions assume the use of lex for consistency with the book. You are encouraged to transition to flex and bison (http://www.gnu.org/software/bison/).

Source Language: The source language is defined by the grammar in Appendix A.1.

Regular Expressions: Write regular expressions for the following tokens:

- **id** (see Figure 3.23 for inspiration)
- **basic** (a basic token could be either "int" or "float")
- **num** (see Section 2.6.3 for inspiration)
- **real** (see Figure 3.23 for inspiration)

Hand in the regular expressions for these tokens. You are to hand in ordinary regular expressions, not lex syntax for them.

Symbol Table: Refactor homework 1 to account for the comment that the TA gave you (if any). Then, modify the original list_insert from homework 1 to return a pointer to the newly inserted element.

```
    list_entry_t *list_insert(list_head_t *, void *key, void *value);
```

The return value is a pointer to the newly inserted list entry if all went well, or NULL in case of error. Hand in: the source code, and the code and output for the test cases defined in homework 1:

- Test your code on a list whose keys and values are doubles. Your test code should execute the following operations and print out the item returned by list_search (if list_search returns NULL, your code should print out a warning message): insert(1,2), insert(3,4), insert(5,6), search(1), search(3), search(5), insert(3,7), search(8), search(3).
- Test your code on a list whose keys are strings and values are doubles. Your test code should execute the following operations and print out the item returned by list_search (if list_search returns NULL, your code should print out a warning message): insert("one",2), insert("two", 4), insert("three",6), search("one"), search("three"), search("five"), insert("three",7), search("eight"), search("three").

Lexical Analyzer: Write a lexical analyzer with lex (or flex-bison) for the language in Appendix A.1. The lexical analyzer should have the following effect:

- *symbol table*: all id's should be installed in the symbol table. The symbol table is implemented with a linked list. The key is a string that represents the identifier, copied from yytext. The value is NULL. A symbol should be inserted only once in the symbol table.
- *output*: the analyzer should print out the name of the token whenever it recognizes one. For example, the program
  ```
  int i;
  while (true)
          i=i+1;
  ```
  should result in the output: basic id; while (true) id = id + num;
  Additionally, the output should contain a full print out of the contents of the symbol table.
- *return value*: The return values are a code for the token. See Figure 3.23 for inspiration. The return value is unused in Homework 2, but it will be critical in Homework 3.
- *yylval*: Define yylval as a void * in the preamble. Then, set yylval as:
  - **basic**: define two integer variables called int_var and float_var, and set yylval to be a pointer to the appropriate variable (cast to void *)
  - **id**: a pointer to the corresponding `list_entry_t` (cast to void *)
  - **num, real**: allocate an integer (double) variable with the numerical value of the num (real), and use a pointer to it as yylval (cast to void *)
  - **bool:** define two integer variables called true_var and false_var, and set yylval to be a pointer to the appropriate variable (cast to void *)

  The yylval is NULL for all other tokens. The yylval is unused in Homework 2, but it will be critical in Homework 3.
- *yywrap*: At the end of the input file, lex invokes the yywrap function (`int yywrap()`). Define yywrap to return 1.

To test your analyzer, write a main in the "auxiliary functions" section. The main should minimally call the lexical analyzer by invoking yylex. It may also call various other functions, for example, to initialize the linked list, to print out the symbol table at the end, and so on.

Hand in: source code and its output on the program at the bottom of page 986; other required submission elements as described in the Web site.

Source control: You will implement the following method for source control:

1. If you do not have an account on volatile.case.edu, request one to `help@eecs.case.edu`
2. Sign up for a (free) account on a non-open source server (e.g., unfuddle.com)
3. Check in homework 1 and 2 in the repository (*hint*: use TortoiseSVN, esvn, or the svn command).
4. Check out a copy of the repository to your account on volatile (*hint*: use svn checkout or esvn).
5. Verify that you can edit, compile, and run homework 1 from its copy on volatile

Notes:

- Do *not* hand in anything for this section. However, you are going to be responsible for your code throughout the semester even if your primary development platform fails.
- Keep your password secret: you are going to share responsibility if your homework is copied. Do not use open-source SVN repositories (e.g., Google).
- If you would like to use an alternative strategy for source control (e.g., carbonite, rsync, other), hand in a description of your strategy, and any implementation notes that you may want to provide. Your plan must enable you to continue working on your assignment (e.g., on volatile) with practically no downtime in case of failure of your primary development platform.