1. License.

2. Testdriver introduction.

Testout the lexical part of O_2. This is a quick and dirty program originally c++ but now literate? So don't critique me on its sparseness. Its header files etc are cloned from O_2's implementation. Its logic is the same pattern as O_2 program. Same notes and chords to parsing.

The driver uses the **test_components.lex** grammar to parse and test out the data files inputs. It is equivalent to O_2's **pass3.lex** token pass without the other lexical components. In **/usr/local/yacco2/qa**, there are various dat files acting as exercisers for testdriver. **/usr/local/yacco2/qa/o2testdriver_tests.bat** bash script runs the various tests.

The claim-to-fame for **test_components.lex** is how the **arbitrator-code** is placed within a rule that has direct thread calls. Rule **Rtoken** closures these other rules that directly call threads. Due to current O_2 implementation, this is a requirement to gen the arbitrator procedure. Why not have a read: **/usr/local/yacco2/docs/test_components.pdf** with its enlightened comments.

3. Append header files for testdriver environment.

⟨ accrue testdriver code 3 ⟩ ≡

```
#include "globals.h"
#include "yacco2_stbl.h"
  using namespace yacco2_stbl;
#include "test_components.h"
#include "o2_externs.h"
```

See also sections 4 and 5.

This code is used in section 6.

4. Define variables.

⟨ accrue testdriver code 3 ⟩ +≡

```
int RECURSION_INDEX__(0);
YACCO2_define_trace_variables( );
yacco2::TOKEN_GAGGLE P3_tokens;
yacco2::TOKEN_GAGGLE Error_queue;
yacco2::TOKEN_GAGGLE tc_recycle_bin;
T_sym_tbl_report_card report_card;
std::string tc_file_to_compile;
STBL_T_ITEMS_type STBL_T_ITEMS;
yacco2::CHART_SW('n');
yacco2::CHARERR_SW('n');
yacco2::CHARPRT_SW('n');
STATES_type LR1_STATES;
LR1_STATES_type LR1_COMMON_STATES;
bool LR1_HEALTH(LR1_COMPATIBLE);
int NO_LR1_STATES(0);
STATES_SET_type VISITED_MERGE_STATES_IN_LA_CALC;
CYCLIC_USE_TBL_type CYCLIC_USE_TABLE;
```

5. Mainline.

```
⟨ accrue testdriver code 3 ⟩ +≡
  int main(int argc, char *argv[])
  {
    cout ≪ "Testdriver␣start" ≪ endl;
    yacco2::lrclog ≪ "Testdriver␣start" ≪ endl;
    LOAD_YACCO2_KEYWORDS_INTO_STBL();
    GET_CMD_LINE(argc, argv, Yacco2_holding_file, Error_queue);
    if (Error_queue.empty() ≠ true) {
      DUMP_ERROR_QUEUE(Error_queue);
      return −1;
    }
    YACCO2_PARSE_CMD_LINE(T_SW, ERR_SW, PRT_SW, tc_file_to_compile, Error_queue);
    if (Error_queue.empty() ≠ true) {
      DUMP_ERROR_QUEUE(Error_queue);
      return −1;
    }
    tok_can < std::ifstream > cmd_line(tc_file_to_compile.c_str());
    if (cmd_line.file_ok() ≡ NO) {
      cout ≪ "Error␣occurred␣file␣does␣not␣exist:␣" ≪ tc_file_to_compile.c_str() ≪ endl;
      return 1;
    }     /* yacco2::YACCO2_TH__ = 1;//watch test_component.lex parse */
         /* yacco2::YACCO2_T__ = 1;//watch terminals as fetched */
    yacco2::YACCO2_AR__ = 1;      /* watch arbitration */
      /* yacco2::YACCO2_MSG__ = 1;//watch communication between threads */
    using namespace NS_test_components;
    Ctest_components tc_fsm;
    Parser tc(tc_fsm, &cmd_line, &P3_tokens, 0, &Error_queue, &tc_recycle_bin, 0);
    tc.parse();
    yacco2::TOKEN_GAGGLE::iterator i = P3_tokens.begin();
    yacco2::TOKEN_GAGGLE::iterator ie = P3_tokens.end();
    cout ≪ "Dump␣of␣P3␣tokons" ≪ endl;
    for (int yyy = 1; i ≠ ie; ++i) {
      CAbs_lr1_sym * sym = *i;
      if (sym ≡ yacco2::PTR_LR1_eog__) continue;
      cout ≪ yyy ≪ "::␣" ≪ sym→id() ≪ "␣file␣no:␣" ≪ sym→external_file_id() ≪ "␣line␣no:␣" ≪
          sym→line_no() ≪ "␣pos:␣" ≪ sym→pos_in_line() ≪ endl;
      yacco2::lrclog ≪ yyy ≪ "::␣" ≪ sym→id() ≪ "␣file␣no:␣" ≪ sym→external_file_id() ≪
          "␣line␣no:␣" ≪ sym→line_no() ≪ "␣pos:␣" ≪ sym→pos_in_line() ≪ endl;
      ++yyy;
    }
    if (Error_queue.empty() ≠ true) {
      DUMP_ERROR_QUEUE(Error_queue);
      cout ≪ "Testdriver␣exit" ≪ endl;
      yacco2::lrclog ≪ "Testdriver␣exit" ≪ endl;
      return −1;
    }     /* yacco2::Delete_tokens(Error_queue, true); */
  exit: cout ≪ "Testdriver␣exit" ≪ endl;
    yacco2::lrclog ≪ "Testdriver␣exit" ≪ endl;
    return 0;
```

}

6. Testdriver implementation.
Output program to *testdriver.cpp*.

⟨ `testdriver.cpp` 6 ⟩ ≡
⟨ accrue testdriver code 3 ⟩;

7. Index.

⟨ accrue testdriver code 3, 4, 5 ⟩ Used in section 6.
⟨ `testdriver.cpp` 6 ⟩

TESTDRIVER