LAB ASSIGNMENT - 11

COMPILER DESIGN LAB – BCSE307P

NAME - ADITYARAJ SHRIVASTAVA

REG. NO. – 23BCE1968

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

typedef enum {
    TOK_c = 0,
    TOK_d,
    TOK_DOLLAR,
    TOK_COUNT
} Token;

const char *token_names[] = { "c", "d", "$" };

typedef enum {
    NT_C = 0,
    NT_S,
    NT_NCOUNT
} NonTerminal;

typedef enum { ACT_ERROR=0, ACT_SHIFT, ACT_REDUCE, ACT_ACCEPT } ActionType;

typedef struct {
    ActionType type;
```

```
    int value;
} Action;

typedef struct {
    NonTerminal lhs;
    int rhs_len;
    const char *repr;
} Production;

Production productions[] = {
    { -1, 0, NULL },
    { NT_S, 2, "S -> C C" },
    { NT_C, 2, "C -> c C" },
    { NT_C, 1, "C -> d" }
};
const int PROD_COUNT = 3;

#define STATES 6

Action ACTION[STATES][TOK_COUNT];

int GOTO_TBL[STATES][NT_NCOUNT];

typedef struct {
    const char *s;
```

```
    int pos;
} Scanner;

void scanner_init(Scanner *sc, const char *input) {
    sc->s = input;
    sc->pos = 0;
}

int next_token(Scanner *sc, char *lexeme_buf, int bufsize) {
    while (sc->s[sc->pos] && isspace((unsigned char)sc->s[sc->pos])) sc->pos++;
    char c = sc->s[sc->pos];
    if (c == '\0') {
        if (lexeme_buf) strncpy(lexeme_buf, "$", bufsize);
        return TOK_DOLLAR;
    }
    sc->pos++;
    if (lexeme_buf) {
        lexeme_buf[0] = c;
        lexeme_buf[1] = '\0';
    }

    if (c == 'c') return TOK_c;
    if (c == 'd') return TOK_d;

    return -1;
```

```
}

void set_action(int state, Token tok, ActionType t, int val) {
    ACTION[state][tok].type = t;
    ACTION[state][tok].value = val;
}

void set_goto(int state, NonTerminal nt, int tostate) {
    GOTO_TBL[state][nt] = tostate;
}

void init_parsing_table_example() {
    for (int st = 0; st < STATES; ++st) {
        for (int t = 0; t < TOK_COUNT; ++t) {
            ACTION[st][t].type = ACT_ERROR;
            ACTION[st][t].value = -1;
        }
        for (int nt = 0; nt < NT_NCOUNT; ++nt) GOTO_TBL[st][nt] = -1;
    }

    set_action(0, TOK_c, ACT_SHIFT, 2);
    set_action(0, TOK_d, ACT_SHIFT, 3);
    set_goto(0, NT_C, 1);

    set_action(1, TOK_c, ACT_SHIFT, 2);
```

```
    set_action(1, TOK_d, ACT_SHIFT, 3);
    set_goto(1, NT_C, 4);

    set_action(2, TOK_c, ACT_SHIFT, 2);
    set_action(2, TOK_d, ACT_SHIFT, 3);
    set_goto(2, NT_C, 5);

    set_action(3, TOK_c, ACT_REDUCE, 3);
    set_action(3, TOK_d, ACT_REDUCE, 3);
    set_action(3, TOK_DOLLAR, ACT_REDUCE, 3);

    set_action(4, TOK_DOLLAR, ACT_ACCEPT, 0);

    set_action(5, TOK_c, ACT_REDUCE, 2);
    set_action(5, TOK_d, ACT_REDUCE, 2);
    set_action(5, TOK_DOLLAR, ACT_REDUCE, 2);
}

#define STACK_MAX 256

int parse(Scanner *sc) {
    int stack[STACK_MAX];
    int top = 0;
    stack[top] = 0;
```

```
char lexeme[64];
int curtok = next_token(sc, lexeme, sizeof(lexeme));

while (1) {
   int state = stack[top];
   if (curtok < 0 || curtok >= TOK_COUNT) {
      printf("FAILURE: unexpected/invalid token '%s'\n", lexeme[0] ? lexeme : "?");
      return 0;
   }

   Action a = ACTION[state][curtok];

   if (a.type == ACT_SHIFT) {
      if (top + 1 >= STACK_MAX) {
         fprintf(stderr, "stack overflow\n");
         exit(1);
      }
      stack[++top] = a.value;
      curtok = next_token(sc, lexeme, sizeof(lexeme));
   }
   else if (a.type == ACT_REDUCE) {
      int prod_no = a.value;
      if (prod_no < 1 || prod_no > PROD_COUNT) {
         fprintf(stderr, "internal: bad production number %d\n", prod_no);
         return 0;
```

```
    }
    Production *p = &productions[prod_no];
    int k = p->rhs_len;
    if (k > top) {
        printf("FAILURE: stack underflow while reducing\n");
        return 0;
    }
    top -= k;
    int st_after_pop = stack[top];
    int goto_state = GOTO_TBL[st_after_pop][p->lhs];
    if (goto_state < 0) {
        printf("FAILURE: no goto from state %d on nonterminal (prod %d)\n",
               st_after_pop, prod_no);
        return 0;
    }
    stack[++top] = goto_state;
}
else if (a.type == ACT_ACCEPT) {
    printf("SUCCESS\n");
    return 1;
}
else {
    printf("FAILURE: unexpected token '%s' at position %d (state %d)\n",
           lexeme, sc->pos, state);
    return 0;
```

```
        }
    }
}

int main(int argc, char **argv) {
    char input[1024];

    printf("Enter input string (tokens 'c' and 'd', spaces allowed). End with newline.\n");
    if (!fgets(input, sizeof(input), stdin)) return 0;

    size_t L = strlen(input);
    if (L && input[L-1] == '\n') input[L-1] = '\0';

    Scanner sc;
    scanner_init(&sc, input);

    init_parsing_table_example();

    int ok = parse(&sc);
    return ok ? 0 : 1;
}
```

OUTPUT:

```
PS C:\Users\scope1\Desktop\23bce1968> gcc lab11.c
PS C:\Users\scope1\Desktop\23bce1968> ./a
Enter input string (tokens 'c' and 'd', spaces allowed). End with newline.
dd
SUCCESS
PS C:\Users\scope1\Desktop\23bce1968> ./a
Enter input string (tokens 'c' and 'd', spaces allowed). End with newline.
cd
FAILURE: unexpected token '$' at position 2 (state 1)
PS C:\Users\scope1\Desktop\23bce1968> ./a
Enter input string (tokens 'c' and 'd', spaces allowed). End with newline.
cdcd
SUCCESS
PS C:\Users\scope1\Desktop\23bce1968> ./a
Enter input string (tokens 'c' and 'd', spaces allowed). End with newline.
ddd
FAILURE: unexpected token 'd' at position 3 (state 4)
PS C:\Users\scope1\Desktop\23bce1968>
```