# CD Lab-9 Assignment

**Name: Ketan Goud**
**Reg No: 220905260**
**Section: D D2**
**Roll No: 39**

**Q1. Develop an SLR(1) parser for the given expression grammar and demonstrate parsing actions.**
**E->E+T|T**
**T-> T*F|F**
**F-> ( E )|id**

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

char terminals[] = {'+', '*', '(', ')', 'i', '$'};
char nonterminals[] = {'S', 'E', 'T', 'F'};
typedef enum
{
   shift,
   reduce,
   accept,
   error
} action;

typedef struct
{
   char head;
   char body[10];
} prodn;

typedef struct
{
   action a;
   int value;
} entry;

typedef struct
{
   int top;
   int data[50];
} intstack;

typedef struct
{
   char top;
   char data[50];
} charstack;

entry action_table[12][6];
int goto_table[12][4];
```

```
prodn productions[7];

int termtoint(char a)
{
   for (int i = 0; i < 6; i++)
      if (a == terminals[i])
         return i;
   printf("Error: invalid terminal.");
   exit(1);
}

int nontermtoint(char a)
{
   for (int i = 0; i < 4; i++)
      if (a == nonterminals[i])
         return i;
}

void printstacks(intstack stack, charstack symbol)
{
   printf("Stack: ");
   for (int i = 0; i <= stack.top; i++)
      printf("%d ", stack.data[i]);
   printf("\n");
   printf("Symbol: ");
   for (int i = 0; i <= symbol.top; i++)
      printf("%c ", symbol.data[i]);
   printf("\n");
}

void replace_id(char *str)
{
   int len = strlen(str);
   int i, j = 0;
   char result[len + 1];

   for (i = 0; i < len; i++)
   {
      if (str[i] == 'i' && str[i + 1] == 'd')
      {
         result[j++] = 'i';
         i++;
      }
      else
      {
         result[j++] = str[i];
      }
   }
   result[j] = '\0';

   strcpy(str, result);
}
```

```
int main()
{
    for (int i = 0; i < 12; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            action_table[i][j].a = error;
            action_table[i][j].value = -1;
        }
    }
    action_table[0][2].a = shift;
    action_table[0][2].value = 4;
    action_table[0][4].a = shift;
    action_table[0][4].value = 5;
    action_table[1][0].a = shift;
    action_table[1][0].value = 6;
    action_table[1][5].a = accept;
    action_table[2][0].a = reduce;
    action_table[2][0].value = 2;
    action_table[2][1].a = shift;
    action_table[2][1].value = 7;
    action_table[2][3].a = reduce;
    action_table[2][3].value = 2;
    action_table[2][5].a = reduce;
    action_table[2][5].value = 2;
    action_table[3][0].a = reduce;
    action_table[3][0].value = 4;
    action_table[3][1].a = reduce;
    action_table[3][1].value = 4;
    action_table[3][3].a = reduce;
    action_table[3][3].value = 4;
    action_table[3][5].a = reduce;
    action_table[3][5].value = 4;
    action_table[4][2].a = shift;
    action_table[4][2].value = 4;
    action_table[4][4].a = shift;
    action_table[4][4].value = 5;
    action_table[5][0].a = reduce;
    action_table[5][0].value = 6;
    action_table[5][1].a = reduce;
    action_table[5][1].value = 6;
    action_table[5][3].a = reduce;
    action_table[5][3].value = 6;
    action_table[5][5].a = reduce;
    action_table[5][5].value = 6;
    action_table[6][2].a = shift;
    action_table[6][2].value = 4;
    action_table[6][4].a = shift;
    action_table[6][4].value = 5;
    action_table[7][2].a = shift;
    action_table[7][2].value = 4;
    action_table[7][4].a = shift;
```

```
action_table[7][4].value = 5;
action_table[8][0].a = shift;
action_table[8][0].value = 6;
action_table[8][3].a = shift;
action_table[8][3].value = 11;
action_table[9][0].a = reduce;
action_table[9][0].value = 1;
action_table[9][1].a = shift;
action_table[9][1].value = 7;
action_table[9][3].a = reduce;
action_table[9][3].value = 1;
action_table[9][5].a = reduce;
action_table[9][5].value = 1;
action_table[10][0].a = reduce;
action_table[10][0].value = 3;
action_table[10][1].a = reduce;
action_table[10][1].value = 3;
action_table[10][3].a = reduce;
action_table[10][3].value = 3;
action_table[10][5].a = reduce;
action_table[10][5].value = 3;
action_table[11][0].a = reduce;
action_table[11][0].value = 5;
action_table[11][1].a = reduce;
action_table[11][1].value = 5;
action_table[11][3].a = reduce;
action_table[11][3].value = 5;
action_table[11][5].a = reduce;
action_table[11][5].value = 5;

goto_table[0][1] = 1;
goto_table[0][2] = 2;
goto_table[0][3] = 3;
goto_table[4][1] = 8;
goto_table[4][2] = 2;
goto_table[4][3] = 3;
goto_table[6][2] = 9;
goto_table[6][3] = 3;
goto_table[7][3] = 10;

productions[0].head = 'S';
strcpy(productions[0].body, "E");
productions[1].head = 'E';
strcpy(productions[1].body, "E+T");
productions[2].head = 'E';
strcpy(productions[2].body, "T");
productions[3].head = 'T';
strcpy(productions[3].body, "T*F");
productions[4].head = 'T';
strcpy(productions[4].body, "F");
productions[5].head = 'F';
strcpy(productions[5].body, "(E)");
```

```
productions[6].head = 'F';
strcpy(productions[6].body, "i");

intstack stack;
charstack symbol;
stack.top = 0;
stack.data[stack.top] = 0;
symbol.top = -1;

char input[50];
printf("Enter the input terminated by $: ");
scanf(" %s", input);
replace_id(input);

int k = 0;
printstacks(stack, symbol);

while (1)
{
   char a = input[k];
   int s = stack.data[stack.top];
   entry e = action_table[s][termtoint(a)];

   if (e.a == shift)
   {
      stack.data[++stack.top] = e.value;
      symbol.data[++symbol.top] = a;
      printf("Shifting %c from input\n", a);
      k++;
      printstacks(stack, symbol);
   }
   else if (e.a == reduce)
   {
      prodn p = productions[e.value];
      printf("Reducing using %c -> %s\n", p.head, p.body);
      for (int i = 0; i < strlen(p.body); i++)
      {
         symbol.top--;
         stack.top--;
      }
      symbol.data[++symbol.top] = p.head;

      int jo = goto_table[stack.data[stack.top]][nontermtoint(p.head)];
      stack.data[++stack.top] = jo;
      printstacks(stack, symbol);
   }
   else if (e.a == accept)
   {
      printf("Successfully parsed.\n");
      break;
   }
   else
```

```
        {
            printf("Error: Input string could not be parsed.\n");
            exit(1);
        }
    }

    return 0;
}
```

```
cd_d2@prg:~/220905260/Lab 9$ ./a.out
Enter the input terminated by $: id+id*id$
Stack: 0
Symbol:
Shifting i from input
Stack: 0 5
Symbol: i
Reducing using F -> i
Stack: 0 3
Symbol: F
Reducing using T -> F
Stack: 0 2
Symbol: T
Reducing using E -> T
Stack: 0 1
Symbol: E
Shifting + from input
Stack: 0 1 6
Symbol: E +
Shifting i from input
Stack: 0 1 6 5
Symbol: E + i
Reducing using F -> i
Stack: 0 1 6 3
Symbol: E + F
Reducing using T -> F
Stack: 0 1 6 9
Symbol: E + T
Shifting * from input
Stack: 0 1 6 9 7
Symbol: E + T *
Shifting i from input
Stack: 0 1 6 9 7 5
Symbol: E + T * i
Reducing using F -> i
Stack: 0 1 6 9 7 10
Symbol: E + T * F
```

```
Symbol: E + T * F
Reducing using T -> T*F
Stack: 0 1 6 9
Symbol: E + T
Reducing using E -> E+T
Stack: 0 1
Symbol: E
Successfully parsed.
cd_d2@prg:~/220905260/Lab 9$
```

```
cd_d2@prg:~/220905260/Lab 9$ cc q1.c
cd_d2@prg:~/220905260/Lab 9$ ./a.out
Enter the input terminated by $: (id$
Stack: 0
Symbol:
Shifting ( from input
Stack: 0 4
Symbol: (
Shifting i from input
Stack: 0 4 5
Symbol: ( i
Reducing using F -> i
Stack: 0 4 3
Symbol: ( F
Reducing using T -> F
Stack: 0 4 2
Symbol: ( T
Reducing using E -> T
Stack: 0 4 8
Symbol: ( E
Error: Input string could not be parsed.
```