Saivya Singh
220905730
CSE D
44

# *Lab 9 : Bottom Parser for Simple Grammar*

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*

## Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX_STACK 100
#define MAX_TOKENS 100

typedef struct {
    char lhs;
    int rhs_len;
} Production;

Production productions[] = {
    {'\0', 0},
    {'E', 3},  // E -> E + T
    {'E', 1},  // E -> T
    {'T', 3},  // T -> T * F
    {'T', 1},  // T -> F
    {'F', 3},  // F -> ( E )
    {'F', 1}   // F -> id
};

char* table[12][9] = {
    {"s5",   "",    "",    "s4",   "",    "",    "1", "2", "3"},
    {"",     "s6",  "",    "",     "",    "acc", "",  "",  ""},
    {"",     "r2",  "s7",  "",     "r2",  "r2",  "",  "",  ""},
    {"",     "r4",  "r4",  "",     "r4",  "r4",  "",  "",  ""},
    {"s5",   "",    "",    "s4",   "",    "",    "8", "2", "3"},
    {"",     "r6",  "r6",  "",     "r6",  "r6",  "",  "",  ""},
    {"s5",   "",    "",    "s4",   "",    "",    "",  "9", "3"},
    {"s5",   "",    "",    "s4",   "",    "",    "",  "",  "10"},
    {"",     "s6",  "",    "",     "s11","",    "",  "",  ""},
    {"",     "r1",  "s7",  "",     "r1",  "r1",  "",  "",  ""},
    {"",     "r3",  "r3",  "",     "r3",  "r3",  "",  "",  ""},
    {"",     "r5",  "r5",  "",     "r5",  "r5",  "",  "",  ""}
};
```

```
int getColumn(char *token) {
    if (strcmp(token, "id") == 0) return 0;
    if (strcmp(token, "+") == 0) return 1;
    if (strcmp(token, "*") == 0) return 2;
    if (strcmp(token, "(") == 0) return 3;
    if (strcmp(token, ")") == 0) return 4;
    if (strcmp(token, "$") == 0) return 5;
    return -1;
}

int getGotoColumn(char nt) {
    if (nt == 'E') return 6;
    if (nt == 'T') return 7;
    if (nt == 'F') return 8;
    return -1;
}

int stateStack[MAX_STACK];
int top = -1;
void pushState(int state) { stateStack[++top] = state; }
void popState(int n) { top -= n; }
void printStateStack() {
    printf("State Stack: ");
    for (int i = 0; i <= top; i++)
        printf("%d ", stateStack[i]);
    printf("\n");
}

char* procSymbols[MAX_STACK];
int procCount = 0;
void pushSymbol(const char *sym) { procSymbols[procCount++] =
strdup(sym); }
void popSymbol(int n) { procCount -= n; }
void printSententialForm() {
    printf("Symbol Table: ");
    for (int i = 0; i < procCount; i++)
        printf("%s ", procSymbols[i]);
    printf("\n");
}

void printInput(char *tokens[], int ip, int tokenCount) {
    printf("Input: ");
    for (int i = ip; i < tokenCount; i++)
        printf("%s ", tokens[i]);
    printf("\n");
}

int main() {
    char inputLine[256];
    printf("Enter input tokens separated by space (end with $):\n");
    fgets(inputLine, sizeof(inputLine), stdin);
```

```
char *tokens[MAX_TOKENS];
int tokenCount = 0;
char *tok = strtok(inputLine, " \n\t");
while (tok != NULL) {
    tokens[tokenCount++] = tok;
    tok = strtok(NULL, " \n\t");
}
pushState(0);
int ip = 0;
printf("\nParsing Actions:\n");
while (1) {
    printStateStack();
    printSententialForm();
    printInput(tokens, ip, tokenCount);

    int state = stateStack[top];
    int col = getColumn(tokens[ip]);
    if (col == -1) {
        printf("Error: Unknown token %s\n", tokens[ip]);
        exit(1);
    }
    char *act = table[state][col];
    if (strcmp(act, "") == 0) {
        printf("Error: No action for state %d and token %s\n", state, tokens[ip]);
        exit(1);
    }
    if (strcmp(act, "acc") == 0) {
        printf("ACCEPT\n");
        break;
    } else if (act[0] == 's') {
        int nextState = atoi(act + 1);
        printf("Action: Shift %s, push state %d\n\n", tokens[ip], nextState);
        pushState(nextState);
        pushSymbol(tokens[ip]);
        ip++;
    } else if (act[0] == 'r') {
        int prodNum = atoi(act + 1);
        Production prod = productions[prodNum];
        printf("Action: Reduce by production %d: %c -> ", prodNum, prod.lhs);
        if (prodNum == 1) printf("E + T");
        else if (prodNum == 2) printf("T");
        else if (prodNum == 3) printf("T * F");
        else if (prodNum == 4) printf("F");
        else if (prodNum == 5) printf("( E )");
        else if (prodNum == 6) printf("id");
        printf("\n");
        popState(prod.rhs_len);
        popSymbol(prod.rhs_len);
        int curState = stateStack[top];
        int gotoCol = getGotoColumn(prod.lhs);
        char *gotoVal = table[curState][gotoCol];
        int newState = atoi(gotoVal);
```

```
            char lhsStr[2];
            lhsStr[0] = prod.lhs; lhsStr[1] = '\0';
            printf("Goto: push state %d\n\n", newState);
            pushState(newState);
            pushSymbol(lhsStr);
        } else {
            printf("Unknown action: %s\n", act);
            exit(1);
        }
    }
    return 0;
}
```

**Input :**

id * id + id $

**Output :**

```
Enter input tokens separated by space (end with $):
id * id + id $

Parsing Actions:

---------------------------------------------------------------------------
Stack    Symbol Table     Input          Action
---------------------------------------------------------------------------
   0              -              id * id + id $   Shift id, push state 5
   0 5            id              * id + id $     Reduce by F → id, push state 3
   0 3            F               * id + id $     Reduce by T → F, push state 2
   0 2            T               * id + id $     Shift *, push state 7
   0 2 7          T *              id + id $       Shift id, push state 5
   0 2 7 5         T * id           + id $          Reduce by F → id, push state 10
   0 2 7 10        T * F            + id $          Reduce by T → T * F, push state 2
   0 2            T               + id $          Reduce by E → T, push state 1
   0 1            E               + id $          Shift +, push state 6
   0 1 6          E +              id $            Shift id, push state 5
   0 1 6 5         E + id           $               Reduce by F → id, push state 3
   0 1 6 3         E + F            $               Reduce by T → F, push state 9
   0 1 6 9         E + T            $               Reduce by E → E + T, push state 1
   0 1            E               $              ACCEPT
---------------------------------------------------------------------------
```

**Input :**

( id + id ) * id $

**Output :**

```
Enter input tokens separated by space (end with $):
( id + id ) * id $

Parsing Actions:
------------------------------------------------------------------------
Stack  Symbol Table      Input             Action
------------------------------------------------------------------------
  0             -              ( id + id ) * id $   Shift (, push state 4
  0 4            (              id + id ) * id $    Shift id, push state 5
  0 4 5          ( id            + id ) * id $      Reduce by F → id, push state 3
  0 4 3          ( F             + id ) * id $      Reduce by T → F, push state 2
  0 4 2          ( T             + id ) * id $      Reduce by E → T, push state 8
  0 4 8          ( E             + id ) * id $      Shift +, push state 6
  0 4 8 6         ( E +            id ) * id $        Shift id, push state 5
  0 4 8 6 5       ( E + id         ) * id $          Reduce by F → id, push state 3
  0 4 8 6 3       ( E + F          ) * id $          Reduce by T → F, push state 9
  0 4 8 6 9       ( E + T          ) * id $          Reduce by E → E + T, push state 8
  0 4 8          ( E            ) * id $          Shift ), push state 11
  0 4 8 11        ( E )           * id $            Reduce by F → ( E ), push state 3
  0 3            F              * id $            Reduce by T → F, push state 2
  0 2            T              * id $            Shift *, push state 7
  0 2 7           T *             id $             Shift id, push state 5
  0 2 7 5         T * id           $                Reduce by F → id, push state 10
  0 2 7 10        T * F            $                 Reduce by T → T * F, push state 2
  0 2            T              $               Reduce by E → T, push state 1
  0 1            E              $               ACCEPT
------------------------------------------------------------------------
```

**Input :**

( id + id ) ) $

**Output :**

```
Enter input tokens separated by space (end with $):
( id + id ) ) $

Parsing Actions:
------------------------------------------------------------------------
Stack       Symbol Table     Input          Action
------------------------------------------------------------------------
  0             -              ( id + id ) ) $  Shift (, push state 4
  0 4            (              id + id ) ) $   Shift id, push state 5
  0 4 5          ( id            + id ) ) $     Reduce by F → id, push state 3
  0 4 3          ( F             + id ) ) $     Reduce by T → F, push state 2
  0 4 2          ( T             + id ) ) $     Reduce by E → T, push state 8
  0 4 8          ( E             + id ) ) $     Shift +, push state 6
  0 4 8 6         ( E +            id ) ) $       Shift id, push state 5
  0 4 8 6 5       ( E + id         ) ) $          Reduce by F → id, push state 3
  0 4 8 6 3       ( E + F         ) ) $          Reduce by T → F, push state 9
  0 4 8 6 9       ( E + T         ) ) $          Reduce by E → E + T, push state 8
  0 4 8         ( E           ) ) $          Shift ), push state 11
  0 4 8 11       ( E )          ) $          Reduce by F → ( E ), push state 3
  0 3           F             ) $          Reduce by T → F, push state 2
  0 2           T             ) $          Reduce by E → T, push state 1
  0 1           E             ) $          Error: No action for state 1 and token )
------------------------------------------------------------------------
```

**Input :**

id * id + * id $

**Output :**

```
Enter input tokens separated by space (end with $):
id * id + * id $

Parsing Actions:
------------------------------------------------------------------------
Stack      Symbol Table     Input          Action
------------------------------------------------------------------------
  0             -             id * id + * id $  Shift id, push state 5
  0 5           id             * id + * id $     Reduce by F → id, push state 3
  0 3           F              * id + * id $     Reduce by T → F, push state 2
  0 2           T              * id + * id $     Shift *, push state 7
  0 2 7          T *             id + * id $      Shift id, push state 5
  0 2 7 5        T * id          + * id $         Reduce by F → id, push state 10
  0 2 7 10        T * F           + * id $         Reduce by T → T * F, push state 2
  0 2           T              + * id $        Reduce by E → T, push state 1
  0 1           E              + * id $        Shift +, push state 6
  0 1 6          E +             * id $          Error: No action for state 6 and token *
------------------------------------------------------------------------
```

**Input :**

( ( id * id ) id + ) $

**Output :**

```
Enter input tokens separated by space (end with $):
( ( id * id ) id + ) $

Parsing Actions:
------------------------------------------------------------------------
Stack      Symbol Table     Input          Action
------------------------------------------------------------------------
  0            -             ( ( id * id ) id + ) $   Shift (, push state 4
  0 4           (              ( id * id ) id + ) $     Shift (, push state 4
  0 4 4          ( (            id * id ) id + ) $      Shift id, push state 5
  0 4 4 5         ( ( id          * id ) id + ) $       Reduce by F → id, push state 3
  0 4 4 3         ( ( F           * id ) id + ) $       Reduce by T → F, push state 2
  0 4 4 2         ( ( T           * id ) id + ) $       Shift *, push state 7
  0 4 4 2 7        ( ( T *         id ) id + ) $         Shift id, push state 5
  0 4 4 2 7 5     ( ( T * id       ) id + ) $            Reduce by F → id, push state 10
  0 4 4 2 7 10    ( ( T * F        ) id + ) $            Reduce by T → T * F, push state 2
  0 4 4 2         ( ( T           ) id + ) $          Reduce by E → T, push state 8
  0 4 4 8         ( ( E           ) id + ) $          Shift ), push state 11
  0 4 4 8 11      ( ( E )          id + ) $            Error: No action for state 11 and token id
------------------------------------------------------------------------
```

**Input :**

( id + id ) $

**Output :**

```
Enter input tokens separated by space (end with $):
( id + id ) $

Parsing Actions:
------------------------------------------------------------------------
 Stack       Symbol Table     Input        Action
------------------------------------------------------------------------
   0              -              ( id + id ) $   Shift (, push state 4
   0 4            (               id + id ) $     Shift id, push state 5
   0 4 5           ( id             + id ) $      Reduce by F → id, push state 3
   0 4 3           ( F              + id ) $      Reduce by T → F, push state 2
   0 4 2           ( T              + id ) $      Reduce by E → T, push state 8
   0 4 8           ( E              + id ) $      Shift +, push state 6
   0 4 8 6          ( E +            id ) $        Shift id, push state 5
   0 4 8 6 5       ( E + id          ) $           Reduce by F → id, push state 3
   0 4 8 6 3       ( E + F           ) $           Reduce by T → F, push state 9
   0 4 8 6 9       ( E + T           ) $           Reduce by E → E + T, push state 8
   0 4 8           ( E              ) $          Shift ), push state 11
   0 4 8 11         ( E )            $           Reduce by F → ( E ), push state 3
   0 3            F               $          Reduce by T → F, push state 2
   0 2            T               $          Reduce by E → T, push state 1
   0 1            E               $          ACCEPT
------------------------------------------------------------------------
```