# Assignment No.7

## AIM:

Implement the shift- reduce parser.

## THEORY:

### Bottom up parsing

- Bottom-up parse corresponds to the construction of a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top).
- It uses rightmost derivation to construct the parse tree.
- The rightmost derivation is the one in which we always expand the rightmost non-terminal.

### Handle with example

- A "handle" of a string is a substring that matches the right side of a production, and whose reduction to the nonterminal on the left side of the production represents one step along the reverse of a rightmost derivation.
- The process of discovering a handle & reducing it to the appropriate left-hand side is called handle pruning.
- Rightmost derivation in reverse can be obtained by handle pruning.

  Example:

  $E \rightarrow E+T \mid T$

  $T \rightarrow T*F \mid F$

  $F \rightarrow id$

| Right Sentential Form | Handle | Reducing Production |
|---|---|---|

| | | |
|---|---|---|
| id * id | id | F → id |
| F * id | F | T → F |
| T * id | id | F → id |
| T * F | T * F | T → T * F |

## Stack implementation of shift reduce parsing

- A convenient way to implement a shift-reduce parser is to use a stack to hold grammar symbols and an input buffer to hold the string w to be parsed. We use $ to mark the bottom of the stack and also the right end of the input. Initially, the stack is emply, and the string *w* is on the input, as follows:

  Stack Input

  $ w$

- The parser operates by shifting zero or more input symbols onto the stack until a handle β is on top of the stack. The parser then reduces β to the left *side* of the appropriate production. The parser repeats this cycle until it has detected an error or until the stack contains the start symbol and thc input is empty :

  Stack Input

  $S $

- After entering this configuration, the parser halts and announces successful completion of parsing.
- While the primary operations of the parser are shift and reduce. There are actually four possible actions, a shift-reduce parser can make: (1) shift. (2) reduce, (3) accept, and (4) error.

1. In a shift action, the next input symbol is shifted onto the top of the stack.
2. In a reduce action, the parser knows the right end of the handle is at the top of the stack. It must then locate the left end of the handle within the stack and decide with what nonterminal to replace the handle.
3. In an *accept* action, the parser announces successful completion of parsing.
4. In an *error* action, the parser discovers that syntax error has occurred and calls an error recovery routine.

- Example:

  E → E+T | T

  T → T*F | F

  F → id

  Input string is id * id

  Rightmost derivation

  E => T

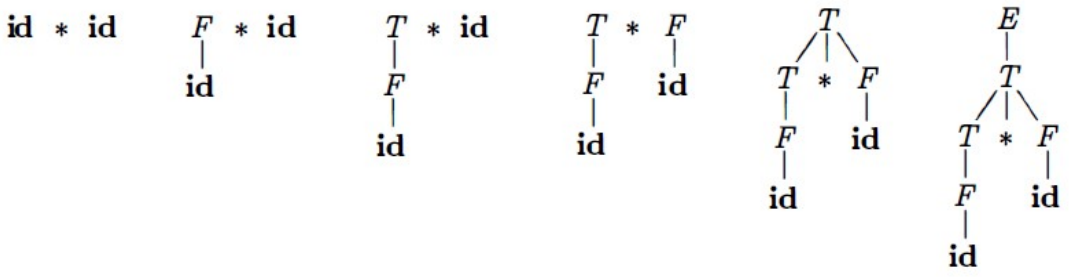
  => T * F

  => T * **id**

  => F * **id**

  => **id** * **id**

  Thus this process is like tracing out the right most derivations in reverse



| STACK | INPUT | ACTION |
|---|---|---|
| $ | id * id $ | Shift |
| $ id | * id $ | Reduce by F → id |
| $ F | * id $ | Reduce by T → F |
| $ T | * id $ | Shift |
| $ T * | id $ | Shift |
| $ T * id | $ | Reduce by F → id |
| $ T * F | $ | Reduce by T → T * F |
| $ T | $ | Reduce by E → T |
| $ E | $ | Accept |

**Conflicts During Shift-Reduce Parsing**

1. Shift/Reduce conflict: Every shift-reduce parser for such a grammar can reach a configuration in which the parser, knowing the entire stack contents and the next input symbol, cannot decide whether to shift or to reduce.

   Example**:**

   *stmt* → **if** *expr* **then** *stmt*

   | **if** *expr* **then** *stmt* **else** *stmt*

   | other

   We cannot tell whether "**if** *expr* **then** *stmt"* is the handle

| Stack | Input |
|---|---|

| …**if** *expr* **then** *stmt* | **else** …$ |
|---|---|

2. Reduce/Reduce conflict**:** the parser cannot decide which of several reductions to make.

   Example**:**

   1. stmt → id ( parameter_list )
   2. stmt → expr := expr
   3. parameter_list → parameter_list , parameter
   4. parameter_list → parameter
   5. parameter → id
   6. expr → id ( expr_list )
   7. expr → id
   8. expr_list → expr_list , expr
   9. expr_list → expr

   "id" - The handle on top of the stack must be reduced, but by which production? production (5)? or production (7)?

| **Stack** | **Input** |
|---|---|
| … id ( id | , id ) … |

## PROGRAM:

## INPUT:

1. Grammar is

   S->aS

   S->aT

   T->bW

   W->c

   Enter Input String:

   aaabc

2. Grammar is

   S->aS

   S->aT

   T->bW

   W->c

   Enter Input String:

   abcd

## OUTPUT:

1. 

| Stack | Action |
|---|---|
| a | Shift a |
| aa | Shift a |
| aaa | Shift a |
| aaab | Shift b |
| aaabc | Shift c |
| aaabW | Reduce W->c |

aaaT Reduce T->bW

aaS Reduce S->aT

aS Reduce S->aS

S Reduce S->aS

String is accepted

2. Stack Action

a Shift a

ab Shift b

abc Shift c

abcd Shift d

String is not accepted

## CONCLUSION:

Thus the shift- reduce parser which is the bottom up parser is implemented.

## REFERENCES:


- Compilers - Principles, Techniques and Tools - A.V. Aho, R. Shethi and J. D. Ullman (Pearson Education)