Shift Reduce Parser in Compi

Prerequisite – Parsing | Set 2 (Bottom Up or Shift Reduce Parsers)

Shift Reduce parser attempts for the construction of parse in a similar manner as done in bottom up parsing i.e. the parse tree is constructed from leaves(bottom) to the root(up). A more general form of shift reduce parser is LR parser.
This parser requires some data structures i.e.

- A input buffer for storing the input string.
- A stack for storing and accessing the production rules.

Basic Operations –

- **Shift:** This involves moving of symbols from input buffer onto the stack.
- **Reduce:** If the handle appears on top of the stack then, its reduction by using appropriate production rule is done i.e. RHS of production rule is popped out of stack and LHS of production rule is pushed onto the stack.
- **Accept:** If only start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When accept action is obtained, it is means successful parsing is done.
- **Error:** This is the situation in which the parser can neither perform shift action nor reduce action and not even accept action.

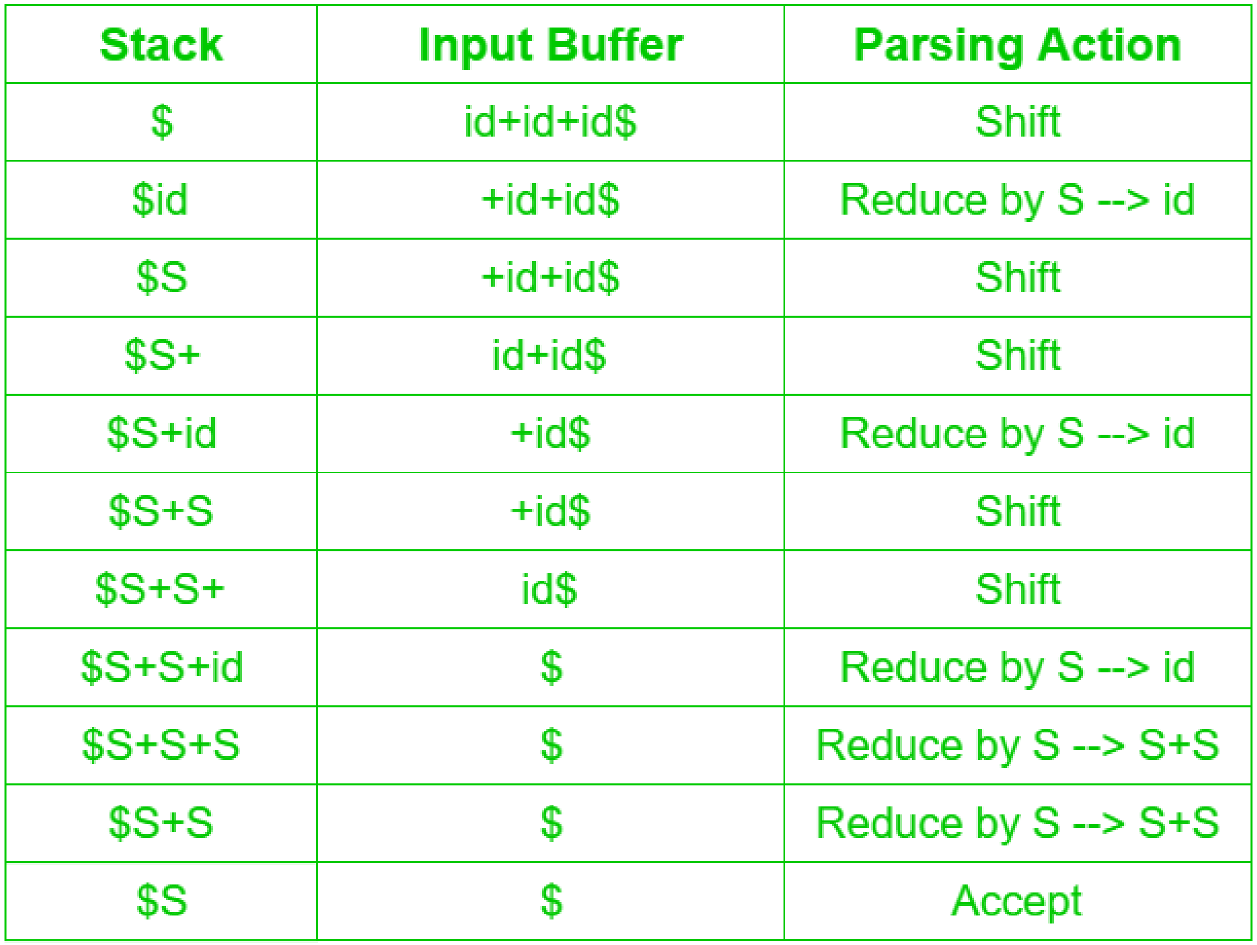
Example 1 – Consider the grammar

S –> S + S

S –> S * S

S –> id

Perform Shift Reduce parsing for input string “id + id + id”.



Stack	Input Buffer	Parsing Action
$	id+id+id$	Shift
$id	+id+id$	Reduce by S --> id
$S	+id+id$	Shift
$S+	id+id$	Shift
$S+id	+id$	Reduce by S --> id
$S+S	+id$	Shift
$S+S+	id$	Shift
$S+S+id	$	Reduce by S --> id
$S+S+S	$	Reduce by S --> S+S
$S+S	$	Reduce by S --> S+S
$S	$	Accept

Example 2 – Consider the grammar

E –> 2E2

E –> 3E3

E –> 4

Perform Shift Reduce parsing for input string “32423”.

Stack	Input Buffer	Parsing Action
$	32423$	Shift
$3	2423$	Shift
$32	423$	Shift
$324	23$	Reduce by E --> 4
$32E	23$	Shift
$32E2	3$	Reduce by E --> 2E2
$3E	3$	Shift
$3E3	$	Reduce by E --> 3E3
$E	$	Accept

Following is the implementation in C-

```
// Including Libraries
#include <bits/stdc++.h>
using namespace std;

// Global Variables
int z = 0, i = 0, j = 0, c = 0;

// Modify array size to increase
// length of string to be parsed
char a[16], ac[20], stk[15], act[10];

// This Function will check whether
// the stack contain a production rule
```

```
    // which is to be Reduce.
    // Rules can be E->2E2 , E->3E3 , E->4
    void check()
    {
        // Coping string to be printed as action
        strcpy(ac,"REDUCE TO E -> ");

        // c=length of input string
        for(z = 0; z < c; z++)
        {
            // checking for producing rule E->4
            if(stk[z] == '4')
            {
                printf("%s4", ac);
                stk[z] = 'E';
                stk[z + 1] = '\0';

                //pinting action
                printf("\n$%s\t%s$\t", stk, a);
            }
        }

        for(z = 0; z < c - 2; z++)
        {
            // checking for another production
            if(stk[z] == '2' && stk[z + 1] == 'E' &&
                                    stk[z + 2] == '2')
            {
                printf("%s2E2", ac);
                stk[z] = 'E';
                stk[z + 1] = '\0';
                stk[z + 2] = '\0';
                printf("\n$%s\t%s$\t", stk, a);
                i = i - 2;
            }

        }

        for(z = 0; z < c - 2; z++)
        {
            //checking for E->3E3
            if(stk[z] == '3' && stk[z + 1] == 'E' &&
                                stk[z + 2] == '3')
            {
                printf("%s3E3", ac);
                stk[z]='E';
                stk[z + 1]='\0';
                stk[z + 1]='\0';
                printf("\n$%s\t%s$\t", stk, a);
                i = i - 2;
            }
        }
        return ; // return to main
    }

    // Driver Function
    int main()
    {
        printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");

        // a is input string
        strcpy(a,"32423");

        // strlen(a) will return the length of a to c
        c=strlen(a);

        // "SHIFT" is copied to act to be printed
        strcpy(act,"SHIFT");

        // This will print Lables (column name)
        printf("\nstack \t input \t action");

        // This will print the initial
        // values of stack and input
        printf("\n$\t%s$\t", a);

        // This will Run upto length of input string
        for(i = 0; j < c; i++, j++)
        {
            // Printing action
            printf("%s", act);

            // Pushing into stack
            stk[i] = a[j];
            stk[i + 1] = '\0';

            // Moving the pointer
            a[j]=' ';

            // Printing action
            printf("\n$%s\t%s$\t", stk, a);

            // Call check function ..which will
            // check the stack whether its contain
            // any production or not
            check();
        }

        // Rechecking last time if contain
        // any valid production then it will
        // replace otherwise invalid
        check();

        // if top of the stack is E(starting symbol)
        // then it will accept the input
        if(stk[0] == 'E' && stk[1] == '\0')
            printf("Accept\n");
        else //else reject
            printf("Reject\n");
    }
    // This code is contributed by Shubhamsingh10
```

C

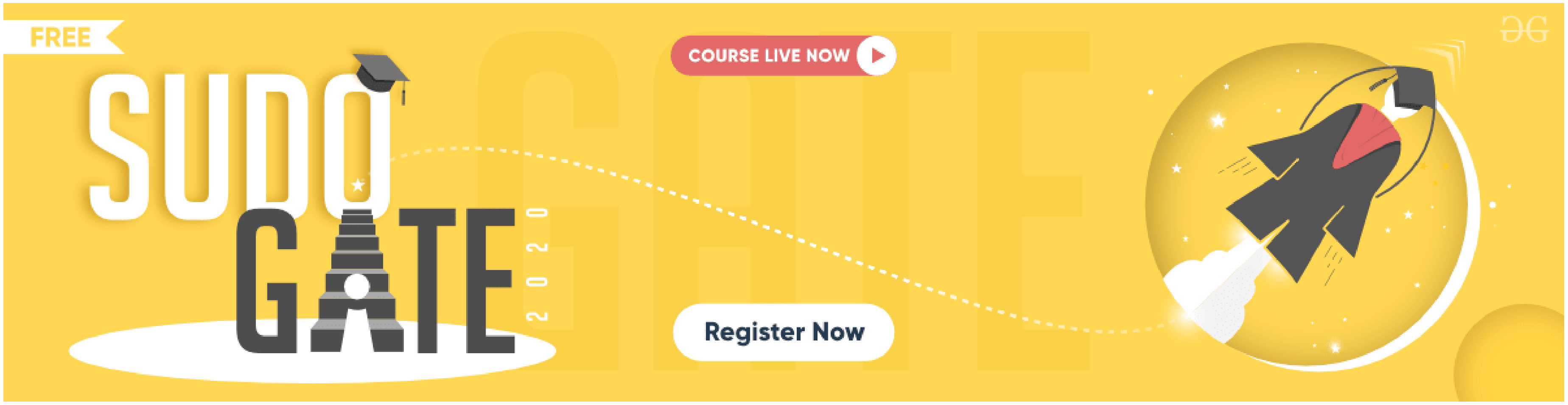
```
//Including Libraries
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

//Global Variables
int z = 0, i = 0, j = 0, c = 0;

// Modify array size to increase
```

```
    // length of string to be parsed
    char a[16], ac[20], stk[15], act[10];

    // This Function will check whether
    // the stack contain a production rule
    // which is to be Reduce.
    // Rules can be E->2E2 , E->3E3 , E->4
    void check()
    {
        // Coping string to be printed as action
        strcpy(ac,"REDUCE TO E -> ");

        // c=length of input string
        for(z = 0; z < c; z++)
        {
            //checking for producing rule E->4
            if(stk[z] == '4')
            {
                printf("%s4", ac);
                stk[z] = 'E';
                stk[z + 1] = '\0';

                //pinting action
                printf("\n$%s\t%s$\t", stk, a);
            }
        }

        for(z = 0; z < c - 2; z++)
        {
            //checking for another production
            if(stk[z] == '2' && stk[z + 1] == 'E' &&
                                    stk[z + 2] == '2')
            {
                printf("%s2E2", ac);
                stk[z] = 'E';
                stk[z + 1] = '\0';
                stk[z + 2] = '\0';
                printf("\n$%s\t%s$\t", stk, a);
                i = i - 2;
            }

        }
        for(z=0; z<c-2; z++)
        {
            //checking for E->3E3
            if(stk[z] == '3' && stk[z + 1] == 'E' &&
                                    stk[z + 2] == '3')
            {
                printf("%s3E3", ac);
                stk[z]='E';
                stk[z + 1]='\0';
                stk[z + 1]='\0';
                printf("\n$%s\t%s$\t", stk, a);
                i = i - 2;
            }
        }
        return ; //return to main
    }

    //Driver Function
    int main()
    {
        printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");

        // a is input string
        strcpy(a,"32423");

        // strlen(a) will return the length of a to c
        c=strlen(a);

        // "SHIFT" is copied to act to be printed
        strcpy(act,"SHIFT");

        // This will print Lables (column name)
        printf("\nstack \t input \t action");

        // This will print the initial
        // values of stack and input
        printf("\n$\t%s$\t", a);

        // This will Run upto length of input string
        for(i = 0; j < c; i++, j++)
        {
            // Printing action
            printf("%s", act);

            // Pushing into stack
            stk[i] = a[j];
            stk[i + 1] = '\0';

            // Moving the pointer
            a[j]=' ';

            // Printing action
            printf("\n$%s\t%s$\t", stk, a);

            // Call check function ..which will
            // check the stack whether its contain
            // any production or not
            check();
        }

        // Rechecking last time if contain
        // any valid production then it will
        // replace otherwise invalid
        check();

        // if top of the stack is E(starting symbol)
        // then it will accept the input
        if(stk[0] == 'E' && stk[1] == '\0')
            printf("Accept\n");
        else //else reject
            printf("Reject\n");
    }
    // This code is contributed by Ritesh Aggarwal
```

Output

```
GRAMMAR is -
E->2E2
E->3E3
```

```
E->4

stack      input      action
$    32423$    SHIFT
$3     2423$    SHIFT
$32      423$    SHIFT
$324       23$    REDUCE TO E -> 4
$32E       23$    SHIFT
$32E2        3$    REDUCE TO E -> 2E2
$3E        3$    SHIFT
$3E3         $    REDUCE TO E -> 3E3
$E         $    Accept
```



Recommended Posts:

Bottom Up or Shift Reduce Parsers | Set 2

Difference between LL and LR parser

StAX XML Parser in Java

Recursive Descent Parser

Parsing ambiguos grammars using LR parser

Operator grammar and precedence parser in TOC

Shift Registers in Digital Logic

Reduce N to 1 with minimum number of given operations

Tensorflow | tf.data.Dataset.reduce()

Compiler Theory | Set 1

Compiler Theory | Set 2

Phases of a Compiler

Three address code in Compiler

Difference between Compiler and Assembler

Symbol Table in Compiler

Ankit87
Check out this Author's contributed articles.

Improved By : imkiller, SHUBHAMSINGH10

Article Tags : Compiler Design GATE CS Technical Scripter

2

2

☐ To-do ☐ Done

Based on **3** vote(s)

Feedback/ Suggest Improvement Add Notes Improve Article

▲

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments