ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

# COMPILER DESIGN LAB

## *EXP 7 - SHIFT REDUCE PARSING*

## AIM :

To construct a shift reduce parser using C++

## REQUIREMENTS :

1. Knowledge of the concepts of parsing and parse tree
2. Knowledge of the concept of shift reduce parser
3. Online C++ compiler to execute code

## THEORY :

**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 the shift-reduce parser is the LR parser.
This parser requires some data structures i.e.

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

**Basic Operations –**

- **Shift:** This involves moving symbols from the 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. The RHS of a production rule

is popped out of a stack and the LHS of a production rule is pushed onto the stack.

- **Accept:** If only the start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When an accepted action is obtained, it 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.

## ALGORITHM :

- We take the input string 32423 and strcpy to variable a
- then take the action as SHIFT and copy it to act variable
- print the initial values of stack and input
- a FOR LOOP RUN upto input string
- Call check function ..which will check the stack whether it contains any production or not
- // Rechecking last time if contain any valid production then it will replace otherwise invalid
- void check():
- This Function will check whether the stack contains a production rule which is to be Reduce.
- Rules can be E->2E2 , E->3E3 , E->4
- for first for loop check for production E->4
- in second FOR LOOP check for another production
- net for LOOP for checking for E->3E3
- return ; return to main
- // 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");

## SOURCE CODE :

```
#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()
{
    // Copying 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';

            //printing 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 Labels (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
```

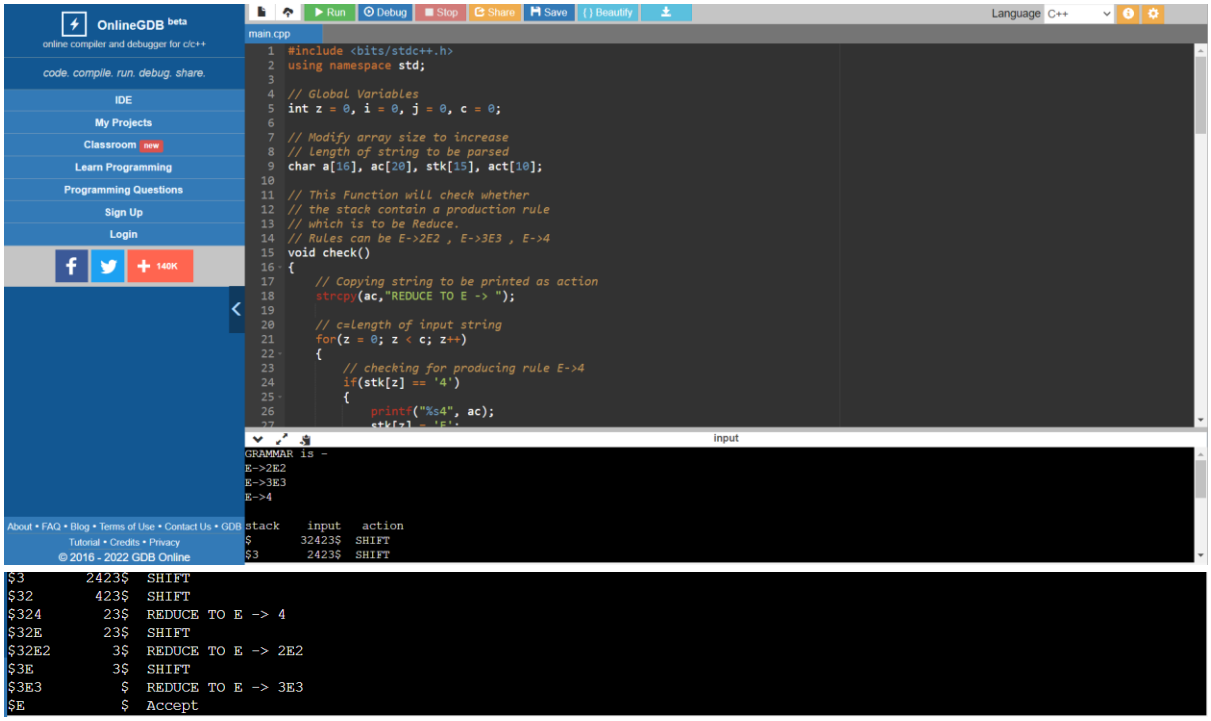
```
        check();
    }

    // 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");

}
```

# SCREENSHOTS OF OUTPUT:




# OBSERVATION:

Hence we observe that the stack is empty with $ sign which means we have successfully implemented a reduce parser program using C++.

**RESULT :**

Thus we have successfully implemented a shift reduce parser program using C++