EX.NO.6

Date:

Implementation of Shift Reduce Parsing Algorithm.

AIM

The aim of implementing the Shift-Reduce Parsing Algorithm is to develop a bottom-up parser that recognizes strings according to a context-free grammar by performing shift and reduce operations to derive the start symbol from the input string.

ALGORITHM

STEP 1: Initialization:

- Begin with an empty stack.
- Append the end-of-input symbol $ to the input string.

STEP 2: **Start Parsing Loop**:

- Repeat the following steps until the input is fully processed or an error occurs.

STEP 3: **Shift Operation**:

- Move the next input symbol to the top of the stack.
- Remove this symbol from the input string.

STEP 4: Check for Reduction:

- Examine the top of the stack to see if it matches the right-hand side of any grammar rule**.**

STEP 5: **Reduce Operation**:

- If a match is found, replace the matching symbols on the stack with the corresponding non-terminal symbol from the left-hand side of the grammar rule.
- Push the non-terminal symbol back onto the stack.

STEP 6: **State Transition (Goto)**:

- After a reduction, use the goto table to transition to the next appropriate state based on the non-terminal symbol produced.

STEP 7: **Shift or Reduce**:

- Continue alternating between shift and reduce operations as necessary depending on the next input symbol and the top of the stack.

STEP 8: Acceptance Condition:

- If the stack contains only the start symbol followed by $ and the input string is empty, accept the input string as valid.

STEP 9: Error Handling:

- If no valid shift or reduce action is possible, report a syntax error and halt the parsing process.

PROGRAM

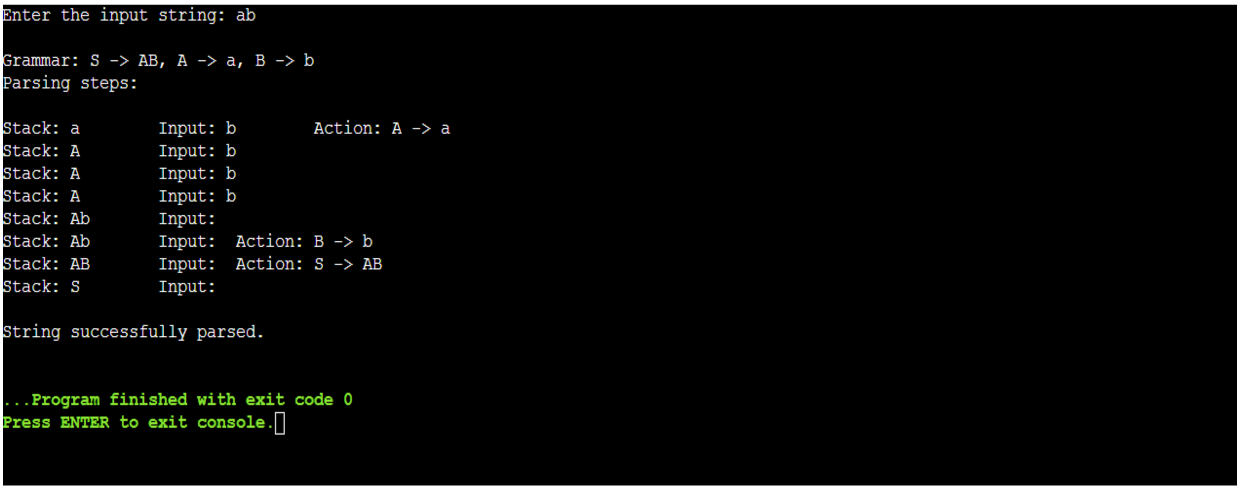
```
#include <stdio.h>
#include <string.h>
char stack[20], input[20];
int top = -1, i = 0;
void push(char c) {
    stack[++top] = c;
}



void pop() {
    stack[top--] = '\0';
}
void display() {
    printf("\nStack: %s", stack);
    printf("\tInput: %s", input + i);
}
int main() {
    printf("Enter the input string: ");
    scanf("%s", input);



    printf("\nGrammar: S -> AB, A -> a, B -> b");
    printf("\nParsing steps:\n");

    while (input[i] != '\0') {
        push(input[i]);
        i++;
        display();


 if (stack[top] == 'a') {
```

```
            pop();
            push('A');
            printf("\tAction: A -> a");
        }
        display();

        if (stack[top] == 'b') {
            pop();
            push('B');
            printf("\tAction: B -> b");
        }
        display();



        if (top > 0 && stack[top] == 'B' && stack[top - 1] == 'A') {
            pop();
            pop();
            push('S');
            printf("\tAction: S -> AB");
        }
        display();
    }



    if (top == 0 && stack[top] == 'S') {
        printf("\n\nString successfully parsed.\n");
    } else {
        printf("\n\nError: Input string not parsed.\n");
    }
    return 0;
}
```

INPUT

Enter the input string: ab

OUTPUT



```
Enter the input string: ab

Grammar: S -> AB, A -> a, B -> b
Parsing steps:

Stack: a        Input: b        Action: A -> a
Stack: A        Input: b
Stack: A        Input: b
Stack: A        Input: b
Stack: Ab       Input:
Stack: Ab       Input:  Action: B -> b
Stack: AB       Input:  Action: S -> AB
Stack: S        Input:

String successfully parsed.


...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT

The algorithm successfully parses input strings based on the grammar S -> AB, A -> a, B -> b, effectively managing shifting and reduction operations. It includes error handling for parsing errors and validates the adherence of strings to the defined grammatical structure.