**Shift Reduce Parser:**

Shift Reduce Parser is a class of efficient, table-driven bottom-up parsing methods for computer languages & other notations formally defined by a grammar. The parsing method s most commonly used for parsing programming languages, LR parsing and its variations, are shift-reduce methods. The precedence parsers used before the invention the invention of LR parsing are also shift-reduce methods. All shift-reduce parsers have similar outward effects, in the incremental order in which they build a parse tree or call specific output actions. Thus, Shift Reduce Parsing is a process of reducing a string to the start symbol of a grammar. It uses a stack to hold the grammar and an input to the string.

Shift reduce parsing performs **two** actions:

1.Shift

2.Reduce.

Accept & Error are the basic operations.

**Process:**

\* At Shift action, the current symbol in the input string is pushed to a stack.

\* At each reduction, the symbol gets replaced by the non-terminals.

\* The symbol is the right side of the production & non-terminal is the left side of the production.

**Grammar**:

S🡪 S+S

S🡪S-S

S🡪(S)

S🡪 a

**Input String:**

a1-(a2+a3)

**Explanation:**

Two functions, Driver function (main function) & check () (void) function have been used in this code.

In the driver function, we first take the sample input a1-(a2+a3) on a variable ‘a’. We make columns of ‘stack’, ’input’, ‘action’ to print the result according to our sample output. Then we start a loop from the initial position of our sample string & run till the end of the length of the string. Our goal is to check every character of the string. If the character is an alphanumeric, we push the character to the i’th position of the stack array we declared globally. Then we initialize our initial array with a blank space character. If our character is a ‘symbol’ we do the same thing by pushing the character to our ‘stk’ array and then initialize the array with a blank space. Basically we did the shift operation of “shift-reduced parsing”. Then at each point we check if the stack has similarity to the given grammar.

The ‘check ()’ function verifies the possibility of reducing our string according to our given grammar. Thus, the check () function does the process of “reduction” of the given string.

The code is attached to the report.

**Output:**

