AIM: Write a program to implement Bottom-up parsers.

```
#include <stdio.h>

#include <string.h>


#define SIZE 100


char stack[SIZE], input[SIZE];

int top = -1, ip = 0,i;


void push(char ch) {

   stack[++top] = ch;

}


void pop() {

   stack[top--] = '\0';

}


void displayStack() {

   for (i = 0; i <= top; i++) {

      printf("%c", stack[i]);

   }

}


void displayInput() {

   for (i = ip; i < strlen(input); i++) {

      printf("%c", input[i]);

   }

}
```

```
int reduce() {
    // E ? id
    if (top >= 1 && stack[top] == 'd' && stack[top - 1] == 'i') {
        pop(); pop();
        push('E');
        printf("\tReduce: E ? id\n");
        return 1;
    }
    // E ? E + E
    if (top >= 2 && stack[top] == 'E' && stack[top - 1] == '+' && stack[top - 2] == 'E') {
        pop(); pop(); pop();
        push('E');
        printf("\tReduce: E ? E+E\n");
        return 1;
    }
    // E ? E * E
    if (top >= 2 && stack[top] == 'E' && stack[top - 1] == '*' && stack[top - 2] == 'E') {
        pop(); pop(); pop();
        push('E');
        printf("\tReduce: E ? E*E\n");
        return 1;
    }
    // E ? (E)
    if (top >= 2 && stack[top] == ')' && stack[top - 1] == 'E' && stack[top - 2] == '(') {
        pop(); pop(); pop();
        push('E');
        printf("\tReduce: E ? (E)\n");
        return 1;
    }
    return 0;
}

int main() {
    printf("Enter the input string (use 'i' for id, no spaces): ");
    scanf("%s", input);

    printf("\nStack\tInput\tAction\n");
    printf("-----\t-----\t------\n");

    while (1) {
        displayStack();
        printf("\t");
        displayInput();
        printf("\t");

        // Shift
        if (ip < strlen(input)) {
            char current = input[ip++];
            push(current);
            printf("Shift: %c\n", current);
        }

        // Try to reduce as much as possible
        int reduced = 1;
        while (reduced) {
            displayStack();
            printf("\t");
            displayInput();
            printf("\t");

            reduced = reduce();
        }

        // Final acceptance condition
        if (top == 0 && stack[top] == 'E' && ip == strlen(input)) {
            printf("Accepted!\n");
            break;
        }

        // If nothing can be done and not accepted, then reject
        if (ip == strlen(input) && !(top == 0 && stack[top] == 'E')) {
            printf("Rejected!\n");
            break;
        }
    }

    return 0;
}
```

Output:

```
Enter the input string (use 'i' for id, no spaces): id+id*id


Stack   Input   Action

-----   -----   ------

        id+id*id       Shift: i

i       d+id*id i      d+id*id Shift: d

id       +id*id        Reduce: E ? id

E       +id*id  E      +id*id  Shift: +

E+      id*id   E+      id*id   Shift: i

E+i     d*id    E+i     d*id    Shift: d

E+id    *id            Reduce: E ? id

E+E     *id            Reduce: E ? E+E

E       *id     E       *id     Shift: *

E*      id      E*      id      Shift: i

E*i     d       E*i     d       Shift: d

E*id                  Reduce: E ? id

E*E                   Reduce: E ? E*E

E             Accepted!
```