7 - Shift Reduce Parser

Tuesday, 5 March 2024 10:10 AM

AIM: Write a program to implement Shift-Reduce Parser

Sample Input:

E → E+E
E → E*E
E → id

check the acceptance of id+id*id

LOGIC:

Stack	I/p string	Action.
$	id+id*id$	Shift 'id'
$id	+id*id$	Reduce as 'E→id'
$E	+id*id$	shift '+'
$E+	id*id$	Shift 'id'
$E+id	*id$	Reduce as 'E→id'
$E+E	*id$	Reduce as 'E→E+E'
$E	*id$	Shift '*'
$E*	id$	Shift 'id'
$E*id	$	Reduce as 'E→id'
$E*E	$	Reduce as 'E→E*E'
$E	$	Accept.

PROCEDURE:

1. Input the number of production rules and then input the production rule in the specified format.
2. Parse and store the production rules in a struct array.
3. input the input string.
4. iterate through the input string and for each character, add it to the stack and print current stack and remaining input.
5. then, iterate through the production rules.
6. If the RHS of a production rule matches a substring in the stack, replace that substring with the LHS of the production rule.
7. Print the updated stack, remaining substring and that there has been a reduction.
8. Repeat steps 4-7 until either the input string has been fully processed or the stack matches the start symbol.
9. If stack only contains the start symbol and if the entire input string has been processed, print "Accepted". Else, if only the entire input string has been processed, print "Not Accepted."

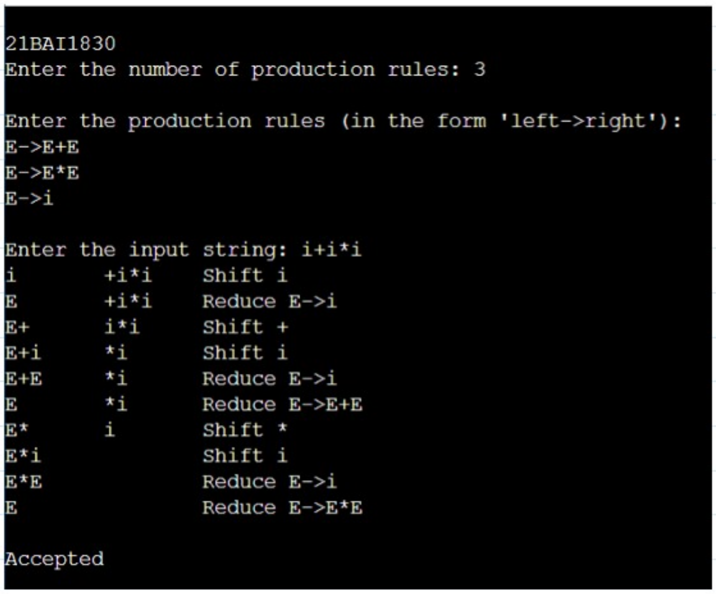
First & Follow

Grammar : A → BC
B → A | aC | ε
C → aB | Cb | ε

First (A) = { a, ε, b }
First (B) = { a, ε }
First (C) = { a, b, ε }

Follow (A) = { $, a, b }
Follow (B) = { a, $, b }
Follow (C) = { $, a, b }

SAMPLE CODE OUTPUT



```
21BAI1830
Enter the number of production rules: 3

Enter the production rules (in the form 'left->right'):
E->E+E
E->E*E
E->i

Enter the input string: i+i*i
i       +i*i    Shift i
E       +i*i    Reduce E->i
E+      i*i     Shift +
E+i     *i      Shift i
E+E     *i      Reduce E->i
E       *i      Reduce E->E+E
E*      i       Shift *
E*i             Shift i
E*E             Reduce E->i
E               Reduce E->E*E

Accepted
```

CODE

```c
#include <stdio.h>
#include <string.h>
struct ProductionRule
{
    char left[10];
    char right[10];
};
int main()
{
    printf("\n21BAI1830 ");
    char input[20], stack[50], temp[50], ch[2], *token1, *token2, *substring;
    int i, j, stack_length, substring_length, stack_top, rule_count = 0;
    struct ProductionRule rules[10];
    stack[0] = '\0';
    // User input for the number of production rules
    printf("\nEnter the number of production rules: ");
    scanf("%d", &rule_count);
    // User input for each production rule in the form 'left->right'
    printf("\nEnter the production rules (in the form 'left->right'): \n");
    for (i = 0; i < rule_count; i++)
    {
        scanf("%s", temp);
        token1 = strtok(temp, "->");
        token2 = strtok(NULL, "->");
        strcpy(rules[i].left, token1);
        strcpy(rules[i].right, token2);
    }
    // User input for the input string
    printf("\nEnter the input string: ");
    scanf("%s", input);
    i = 0;
    while (1)
    {
        // If there are more characters in the input string, add the next character to the stack
        if (i < strlen(input))
        {
            ch[0] = input[i];
            ch[1] = '\0';
            i++;
            strcat(stack, ch);
            printf("%s\t", stack);
            for (int k = i; k < strlen(input); k++)
            {
                printf("%c", input[k]);
            }
            printf("\tShift %s\n", ch);
        }

        // Iterate through the production rules
        for (j = 0; j < rule_count; j++)
        {
            // Check if the right-hand side of the production rule matches a substring in the stack
            substring = strstr(stack, rules[j].right);
            if (substring != NULL)
            {
                // Replace the matched substring with the left-hand side of the production rule
                stack_length = strlen(stack);
                substring_length = strlen(substring);
                stack_top = stack_length - substring_length;
                stack[stack_top] = '\0';
                strcat(stack, rules[j].left);
                printf("%s\t", stack);
                for (int k = i; k < strlen(input); k++)
                {
                    printf("%c", input[k]);
                }
                printf("\tReduce %s->%s\n", rules[j].left, rules[j].right);
                j = -1; // Restart the loop to ensure immediate reduction of the newly derived production rule
            }
        }
        // Check if the stack contains only the start symbol and if the entire input string has been processed
        if (strcmp(stack, rules[0].left) == 0 && i == strlen(input))
        {
            printf("\nAccepted");
            break;
        }
        // Check if the entire input string has been processed but the stack doesn't match the start symbol
        if (i == strlen(input))
        {
            printf("\nNot Accepted");
            break;
        }
    }
    return 0;
}
```