Shift Reduce Parser
Compiler Design – Lab 7

Name: Ojas Patil
Reg No: 21BAI1106

AIM

Implement a Shift reduce parser for a given production using C/C++/Java program.

Algorithm

1. Input Production Rules: Prompt the user to input the number of production rules and store each rule in the format 'left->right'.
2. Input String: Prompt the user to input the string to be processed by the parser.
3. Shift Operation: Parse the input string character by character, shifting each character onto the stack and displaying the current state of the stack and the remaining input string.
4. Reduce Operation: Iterate through production rules, replacing any matched substrings in the stack with the left-hand side of the rule, and displaying the updated stack and input string.
5. Acceptance Check: Verify if the stack contains only the start symbol and if the entire input string has been processed; if so, output "Accepted".
6. Rejection Check: If the input string has been fully processed but the stack doesn't contain the start symbol, output "Not Accepted".

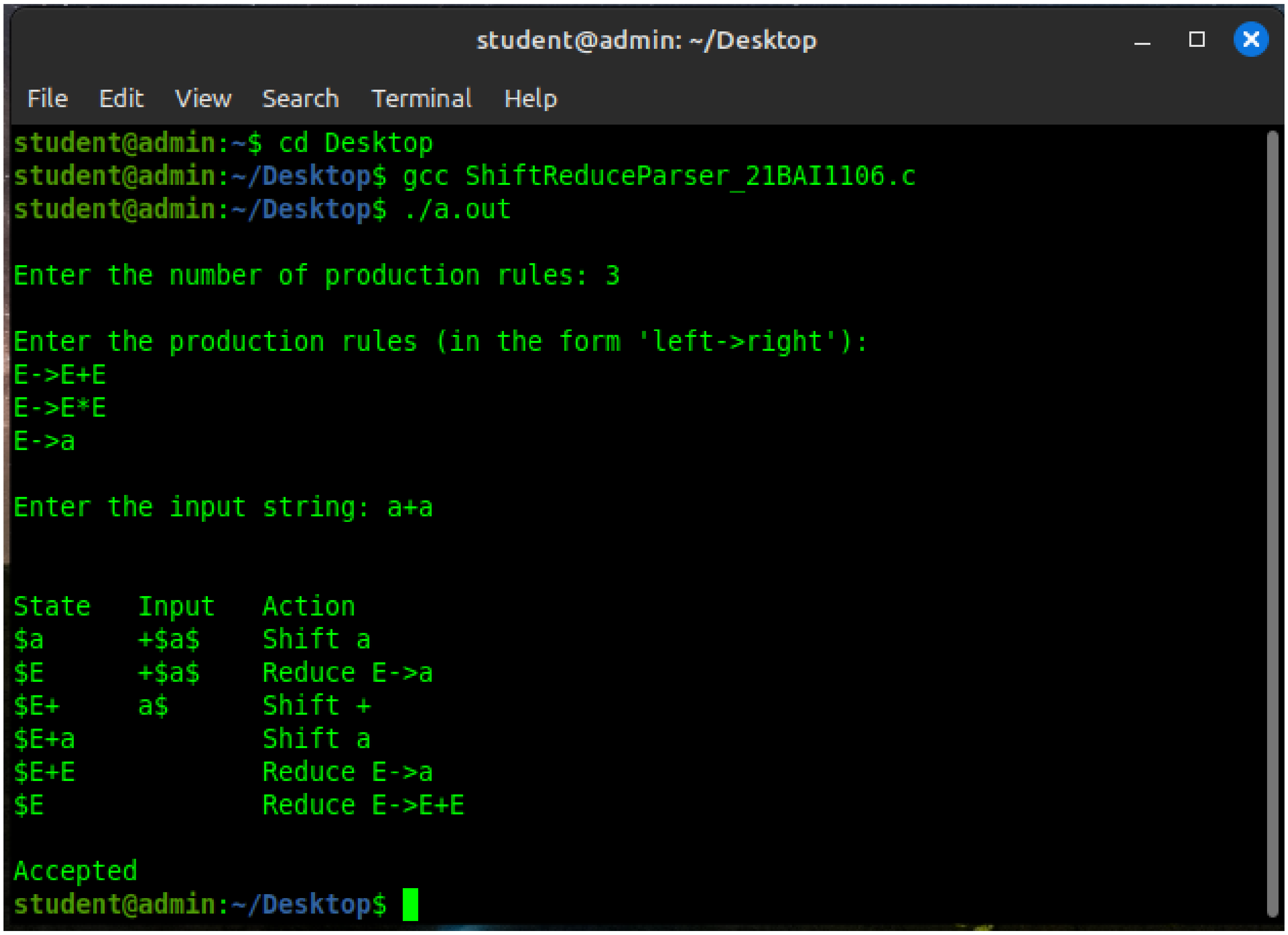
Explanation

The code implements a shift-reduce parser where the user defines production rules and provides an input string. It processes the input string by shifting characters onto a stack and reducing substrings based on the rules. If the stack contains only the start symbol and the input string is fully processed, it's accepted; otherwise, it's rejected. This demonstrates the parsing technique to determine the input string's acceptability according to the provided rules.

Sample Input

```
E->E+E
E->E*E
E->a
```

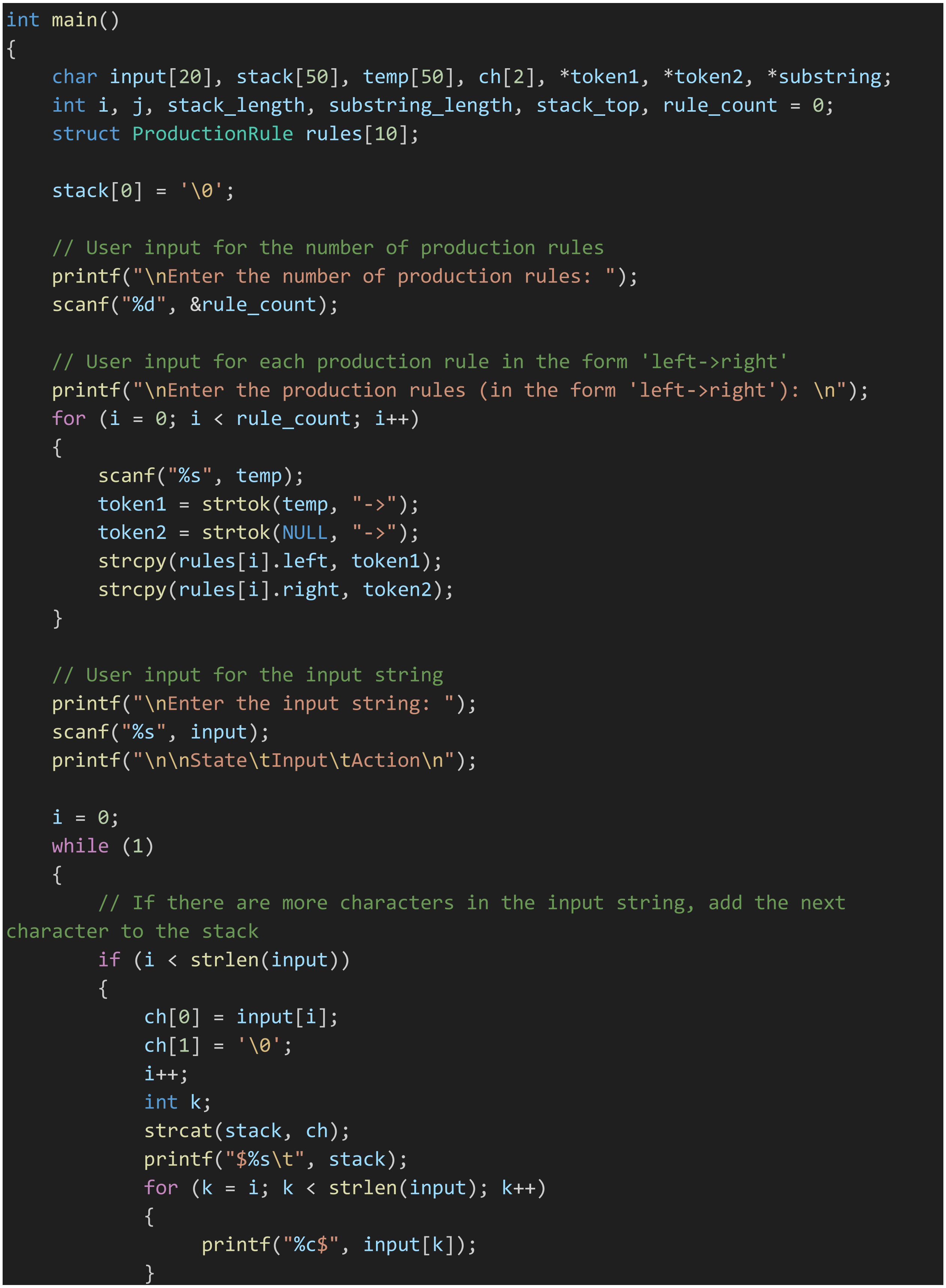
Output Screenshot



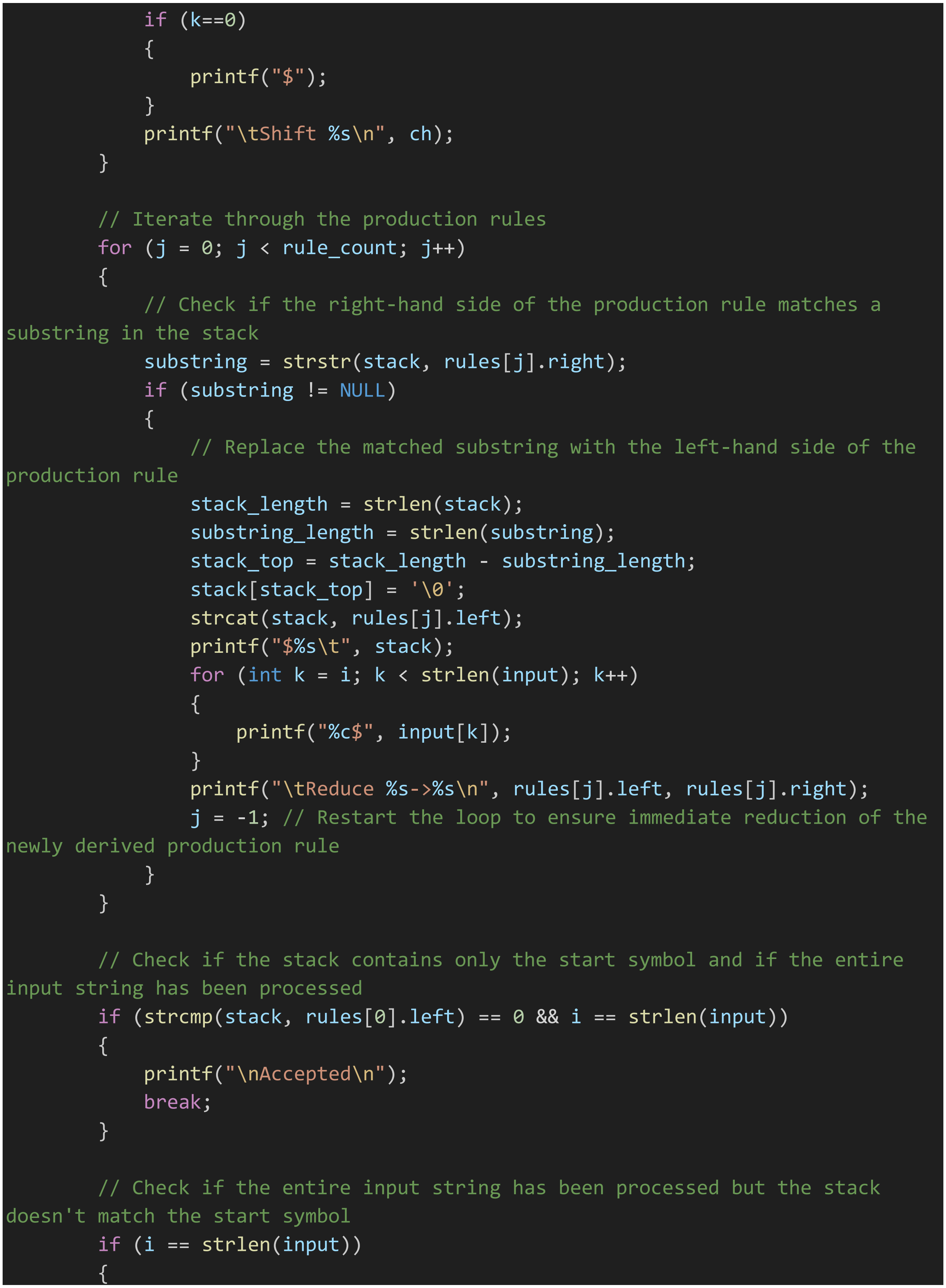
```
student@admin:~$ cd Desktop
student@admin:~/Desktop$ gcc ShiftReduceParser_21BAI1106.c
student@admin:~/Desktop$ ./a.out

Enter the number of production rules: 3

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

Enter the input string: a+a


State   Input   Action
$a      +$a$    Shift a
$E      +$a$    Reduce E->a
$E+     a$      Shift +
$E+a            Shift a
$E+E            Reduce E->a
$E              Reduce E->E+E

Accepted
student@admin:~/Desktop$ 
```

Source Code

```c
#include <stdio.h>
#include <string.h>

struct ProductionRule
{
    char left[10];
    char right[10];
};
```



```
int main()
{
    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);
    printf("\n\nState\tInput\tAction\n");

    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++;
            int k;
            strcat(stack, ch);
            printf("$%s\t", stack);
            for (k = i; k < strlen(input); k++)
            {
                 printf("%c$", input[k]);
            }
```



```
            if (k==0)
            {
                printf("$");
            }
            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\n");
            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\n");
            break;
        }
    }

    return 0;
}
```

Result

Thus, we have written an C code to take input grammar and perform Shift Reduce Parsing.