

**BCSE307P**  
**Compiler Design Lab**  
**Lab Assignment 1**



**VIT<sup>®</sup>**

---

**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

**Name – Ishan Kapoor**  
**Registration Number – 21BCE5882**  
**Submitted to – Prof. S. Srisakthi**

1. Write a C program to identify whether a given line is a comment or not.

### CODE:

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  #include <string.h>
4
5  bool isComment(const char *line) {
6      // Check for single-line comment
7      if (strncmp(line, "//", 2) == 0) {
8          return true;
9      }
10
11     // Check for multi-line comment
12     if (strncmp(line, "/*", 2) == 0) {
13         return true;
14     }
15
16     return false;
17 }
18 int main() {
19     char line[100];
20
21     // Read the line from input
22     printf("Enter a line: ");
23     fgets(line, sizeof(line), stdin);
24
25     // Remove the newline character at the end
26     line[strcspn(line, "\n")] = '\0';
27
28     // Check if the line is a comment
29     if (isComment(line)) {
30         printf("The line is a comment.\n");
31     } else {
32         printf("The line is not a comment.\n");
33     }
34
35     return 0;
36 }
```

## OUTPUT:

```
Enter a line: //x = x+1;  
The line is a comment.
```

```
-----  
Process exited after 68.78 seconds with return value 0  
Press any key to continue . . . |
```

```
Enter a line: /*x=x+1*/  
The line is a comment.
```

```
-----  
Process exited after 8.642 seconds with return value 0  
Press any key to continue . . . |
```

```
Enter a line: x = x+1;  
The line is not a comment.
```

```
-----  
Process exited after 5.961 seconds with return value 0  
Press any key to continue . . . |
```

2. Write a C program to recognize strings under 'a', 'a\*b+', 'abb'.

## CODE:

```
1  #include<stdio.h>  
2  #include<conio.h>  
3  #include<string.h>  
4  #include<stdlib.h>  
5  void main()  
6  {  
7      char s[20],c;  
8      int state=0,i=0;  
9      printf("\n Enter a string: ");  
10     gets(s);  
11     while(s[i]!='\0')  
12     {  
13         switch(state)  
14         {  
15             case 0: c=s[i++];  
16             if(c=='a')  
17                 state=1;  
18             else if(c=='b')  
19                 state=2;  
20             else  
21                 state=6;  
22             break;
```

```
23 case 1: c=s[i++];
24 if(c=='a')
25 state=3;
26 else if(c=='b')
27 state=4;
28 else
29 state=6;
30 break;
31 case 2: c=s[i++];
32 if(c=='a')
33 state=6;
34 else if(c=='b')
35 state=2;
36 else
37 state=6;
38 break;
39 case 3: c=s[i++];
40 if(c=='a')
41 state=3;
42 else if(c=='b')
43 state=2;
44 else
45 state=6;
46 break;
47 case 4: c=s[i++];
48 if(c=='a')
49 state=6;
50 else if(c=='b')
51 state=5;
52 else
53 state=6;
54 break;
55 case 5: c=s[i++];
56 if(c=='a')
```

```
57 state=6;
58 else if(c=='b')
59 state=2;
60 else
61 state=6;
62 break;
63 case 6: printf("\n %s is not recognised.",s);
64 exit(0);
65 }
66 }
67 if(state==1)
68 printf("\n %s is accepted under rule 'a'",s);
69 else if((state==2)||(state==4))
70 printf("\n %s is accepted under rule 'a*b+'",s);
71 else if(state==5)
72 printf("\n %s is accepted under rule 'abb'",s);
73 getch();
74 }
```

## OUTPUT:

```
Enter a string: ab
```

```
ab is accepted under rule 'a*b+'  
-----
```

```
Process exited after 7.618 seconds with return value 13  
Press any key to continue . . .
```

```
Enter a string: aab
```

```
aab is accepted under rule 'a*b+'  
-----
```

```
Process exited after 5.32 seconds with return value 13  
Press any key to continue . . .
```

```
Enter a string: abb
```

```
abb is accepted under rule 'abb'  
-----
```

```
Process exited after 9.228 seconds with return value 13  
Press any key to continue . . .
```

```
Enter a string: bab
```

```
bab is not recognised.  
-----
```

```
Process exited after 1.533 seconds with return value 0  
Press any key to continue . . .
```

### 3. Tokenizing a file using C.

#### CODE:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5  #define MAX_TOKEN_LENGTH 100
6
7  // Function to check if a character is a valid identifier character
8  int isValidIdentifierChar(char ch) {
9      return isalnum(ch) || ch == '_';
10 }
11 // Function to tokenize a string
12 void tokenizeString(const char* input) {
13     int length = strlen(input);
14     char token[MAX_TOKEN_LENGTH];
15
16     int i = 0;
17     while (i < length) {
18         // Skip whitespace characters
19         while (i < length && isspace(input[i])) {
20             i++;
21         }
22
23         // Check for operators
24         if (input[i] == '+' || input[i] == '--' || input[i] == '*' || input[i] == '/' ||
25             input[i] == '=' || input[i] == '-' || input[i] == '%' || input[i] == '>') {
26             printf("Operator: %c\n", input[i]);
27             i++;
28         }
29         // Check for punctuation
30         else if (input[i] == '(' || input[i] == ')' || input[i] == '{' || input[i] == '}' ||
31             input[i] == ',' || input[i] == ';') {
32             printf("Punctuation: %c\n", input[i]);
33             i++;
34         }
35         // Check for constants
36         else if (isdigit(input[i])) {
```

```
37             int j = 0;
38             while (i < length && isdigit(input[i])) {
39                 token[j++] = input[i++];
40             }
41             token[j] = '\0';
42             printf("Constant: %s\n", token);
43         }
```

```

44 // Check for string constants
45 else if (input[i] == '"') {
46     int j = 0;
47     token[j++] = input[i++]; // Store the opening double quote
48
49     while (i < length && input[i] != '"' && j < MAX_TOKEN_LENGTH - 1) {
50         token[j++] = input[i++];
51     }
52     if (i < length && input[i] == '"') {
53         token[j++] = input[i++]; // Store the closing double quote
54         token[j] = '\0';
55         printf("String Constant: %s\n", token);
56     } else {
57         printf("Invalid String Constant\n");
58     }
59 }

```

```

60 // Check for identifiers or keywords
61 else if (isalpha(input[i]) || input[i] == '_') {
62     int j = 0;
63     while (i < length && isValidIdentifierChar(input[i])) {
64         token[j++] = input[i++];
65     }
66     token[j] = '\0';
67     if (strcmp(token, "if") == 0 || strcmp(token, "else") == 0 ||
68         strcmp(token, "while") == 0 || strcmp(token, "for") == 0 ||
69         strcmp(token, "int") == 0) {
70         printf("Keyword: %s\n", token);
71     } else {
72         printf("Identifier: %s\n", token);
73     }
74 }

```

```

74 // Invalid token
75 else {
76     printf("Invalid token: %c\n", input[i]);
77     i++;
78 }
79
80 }
81 int main() {
82     const char* input = "int main() { int x = 5; while (x > 0) { printf(\"x = %d\\n\", x); x--; } return 0; }";
83     tokenizeString(input);
84
85     return 0;
86 }

```

## OUTPUT:

```
Keyword: int
Identifier: main
Punctuation: (
Punctuation: )
Punctuation: {
Keyword: int
Identifier: x
Operator: =
Constant: 5
Punctuation: ;
Keyword: while
Punctuation: (
Identifier: x
Operator: >
Constant: 0
Punctuation: )
Punctuation: {
Identifier: printf
Punctuation: (
String Constant: "x = %d\n"
Punctuation: ,
Identifier: x
Punctuation: )
Punctuation: ;
Identifier: x
Operator: -
Operator: -
Punctuation: ;
Punctuation: }
Identifier: return
Constant: 0
Punctuation: ;
Punctuation: }

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



4. Write a C program for LL (1) Parsing.

1.  $E \rightarrow T E'$
2.  $E' \rightarrow + T E'$
3.  $E' \rightarrow \text{epsilon}$
4.  $T \rightarrow F T'$
5.  $T' \rightarrow * F T'$
6.  $T' \rightarrow \text{epsilon}$
7.  $F \rightarrow ( E )$
8.  $F \rightarrow \text{id}$

### CODE:

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  #include <string.h>
4  // Global variables
5  char input[100];
6  int position;
7  /* Function to check if the current symbol
8  matches the expected symbol*/
9  bool match(char expected) {
10     if (input[position] == expected) {
11         position++;
12         return true;
13     } else {
14         return false;
15     }
16 }
17 // Function to perform the parsing
18 bool E();
19 bool EPrime();
20 bool T();
21 bool TPrime();
22 bool F();
23
24 bool E() {
25     if (T() && EPrime()) {
26         return true;
27     }
28     return false;
29 }
```

```

30 bool EPrime() {
31     int savedPosition = position;
32
33     if (match('+') && T() && EPrime()) {
34         return true;
35     }
36
37     position = savedPosition; // backtrack
38
39     // Epsilon production
40     return true;
41 }
42
43 bool T() {
44     if (F() && TPrime()) {
45         return true;
46     }
47     return false;
48 }

```

```

51 bool TPrime() {
52     int savedPosition = position;
53
54     if (match('*') && F() && TPrime()) {
55         return true;
56     }
57
58     position = savedPosition; // backtrack
59
60     // Epsilon production
61     return true;
62 }
63 bool F() {
64     if (match('(') && E() && match('')) {
65         return true;
66     } else if (match('i')) {
67         return true;
68     }
69     return false;
70 }
71
72 int main() {
73     // Read the input from the user
74     printf("Enter the input string: ");
75     scanf("%s", input);
76
77     // Perform the parsing
78     if (E() && input[position] != '\0') {
79         printf("The input string is valid.\n");
80     } else {
81         printf("The input string not is valid.\n");
82     }
83     return 0;
84 }

```

## **OUTPUT:**

```
Enter the input string: id+id*id
The input string is valid.
```

```
-----
```

```
Process exited after 30.86 seconds with return value 0
Press any key to continue . . .
```