AMITY UNIVERSITY
UTTAR PRADESH

LAB RECORD

BACHELOR OF TECHNOLOGY

B.TECH CS&E 2021-2025 SEMESTER (6)

(Academic Session-: 2021-2025)

Course title : Compiler Construction

Course code : CSE304

Enrollment number : A7605221152

Name of student : Suyash Pandey

Faculty name : Dr. Pawan Singh

Date of submission : _ _/_ _/_ _ _ _

Signature of student :

Department Of Computer Science & Engineering

Amity School Of Engineering &Technology

Amity University, Lucknow Campus

Practical-1

Q1. 1.Consider the following regular expressions:

a)(0 + 1)* + 0*1*

b)(ab*c + (def)+ + a*d+e)+

c)((a + b)*(c + d)*)+ + ab*c*d

Write separate programs for each of the regular expressions mentioned above.

(a)

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
bool matchRegexA(const char *text) {
   while (*text == '0' || *text == '1') {
      text++;
   }
   while (*text == '0' || *text == '1') {
      if (*text == '1') {
         text++;
      } else {
         return false;
      }
   }
   return *text == '\0';
}
int main() {
   const char *input = "001100";
   if (matchRegexA(input)) {
      printf("Match found!\n");
   } else {
      printf("No match found.\n");
   }
   return 0;
}
```

Output:

```
PS D:\TURBOC3\BIN> cd "d:\TURBOC3\" ; if ($?)
Match found!
```

(b)

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
bool matchRegexB(const char *text) {
    while (*text != '\0') {
        if (*text == 'a' && *(text + 1) == 'b') {
            text += 2;
            while (*text == 'b') {
                text++;
            }
            continue;
        } else if (*text == 'd' && *(text + 1) == 'e' && *(text + 2) == 'f') {
            text += 3;
            while (*text == 'e' || *text == 'f') {
                text++;
            }
            continue;
        } else if (*text == 'a' && *(text + 1) == 'd') {
            text += 2;
            while (*text == 'd') {
                text++;
            }
            if (*text == 'e') {
                text++;
            }
            continue;
        } else {
            return false;
        }
    }
    return true;
}
int main() {
```

```
    const char *input = "abcdefade";
    if (matchRegexB(input)) {
        printf("Match found!\n");
    } else {
        printf("No match found.\n");
    }
    return 0;
}
```

Output:

```
No match found.cd "d:\TURBOC3\" ; if ($?)
if ($?) { .\exp2 }
No match found.
```

(C)

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
bool matchRegexC(const char *text) {
    while (*text != '\0') {
        if (*text == 'a' || *text == 'b') {
            text++;
            while (*text == 'a' || *text == 'b') {
                text++;
            }
        } else if (*text == 'c' || *text == 'd') {
            text++;
            while (*text == 'c' || *text == 'd') {
                text++;
            }
        } else {
            return false;
        }
    }
    return true;
```

```
}
int main() {
    const char *input = "abccbabdd";
    if (matchRegexC(input)) {
        printf("Match found!\n");
    } else {
        printf("No match found.\n");
    }
    return 0;
}
```

Output:

```
PS D:\TURBOC3> cd "d:\TURBOC3\" ; if ($?)
if ($?) { .\exp3 }
Match found!
```

Practical-2

Q2. Design a lexical analyzer for identifying types of tokens used in C language.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
int fail(int);
void idorkeyword(char str[]);
void main(void)
{
   int i,j,state,l;
   char s[100], temp[10], c;
   i = 0;
   j = 0;
   state = 0;
   l = 0;
   printf("Enter the Expression");
   scanf("%[^\n]", s);
   l = strlen(s);
   while(i <= l)
   {
      switch(state)
      {
         case 0: c = s[i];
         if(c==' ')
         {
            state = 0;
            i++;
         }
         else if(c == '<')
         {
            state = 1;
            i++;
         }
         else if(c == '=')
```

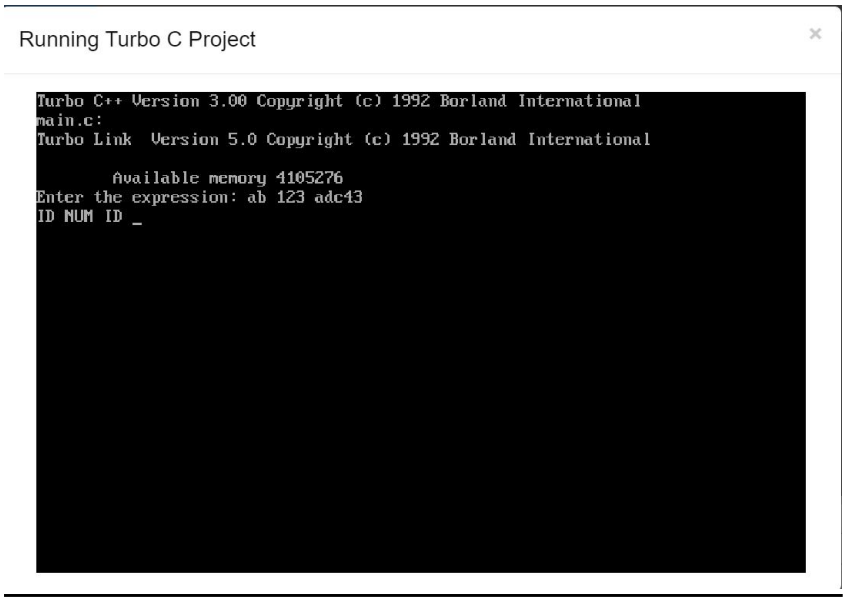
```
{
   state = 5;
   i++;
}
else if(c == '>')
{
   state = 6;
   i++;
}
else state = fail(state);
break;
case 1: c = s[i];
if(c == '=')
{
   state = 2;
   i++;
}
if(c == '>')
{
   state = 3;
   i++;
}
else state = 4;
break;
case 2: printf("RELOP_LE");
i++;
state = 9;
break;
case 3: printf("RELOP_NE");
i++;
state = 9;
break;
case 4: printf("RELOP_LT");
state = 9;
break;
```

```
case 5: printf("RELOP_LE");
i++;
state = 9;
break;
case 6:c = s[i];
if(c == '=')
{
   state = 7;
   i++;
}
else state = 8;
break;
case 7: printf("RELOP_GE");
i++;
state = 9;
break;
case 9: c = s[i];
if(isalpha(c))
{state = 10;
i++;
temp[i] = c;
}
else state = fail(state);
break;
case 10: c = s[i];
if(isalpha(c))
{
   state = 10;
   i++;
   j++;
   temp[j] = c;
}
else if(isdigit(c))
{
   state = 10;
```

```
    i++;
    j++;
    temp[j] = c;
}
case 11: j++;temp[j] = '\0'; idorkeyword(temp);j = 0;state = 12; break;
case 12: c = s[i];
if(isdigit(c))
{
    state = 13;
    i++;
}
else state = fail(state); break;
case 13:c = s[i];
if(isdigit(c))
{
    state = 13;
    i++;
}
else if(c == '.')
{
    state = 14;
    i++;
}
else if(c == '.')
{
    state = 16;
    i++;
}
else if(c == 'E')
{
    state = 16;
    i++;
}
else state = 19;
break;
```

```
            case 19: printf("NUM");
            state = 0;
            break;
        }
    }
}
int fail(int start)
    {
        switch(start)
        {
            case 0: start = 9; break;
            case 9: start = 12; break;
            case 12: start = 0; break;
        }
        return(start);
    }
    void idorkeyword(char str[10])
    {
        char *key1 = "if", *key2 = "then", *key3 = "else";
        if(strcmp(str, key1) == 0||strcmp(str,key2)==0||strcmp(str,key3)==0)
        printf("%S", str);
        else printf("ID");
    }
```

Output:



Practical-3

Q3. Write a program in C to remove left recursion.

```
#include <stdio.h>
#include <string.h>
int main()
{
   char expr[100], *l, *r, *temp, tempprod[20], productions[25][50];
   int i = 0, j = 0, flag = 0;
   printf("Enter the grammar:\n");
   fgets(expr, sizeof(expr), stdin);
   expr[strcspn(expr, "\n")] = '\0';
   l = strtok(expr, "->");
   r = strtok(NULL, "->");
   if (l[0] == r[1])
   {
     flag = 1;
   }
   if (flag)
   {
      strcpy(tempprod, l);
      strcat(tempprod, "'");
      printf("The grammar after eliminating left recursion is:\n");
      printf("%s -> %s%s | %s\n", l, r + 1, tempprod,r + 1, tempprod);
   }
   else
   {
      printf("The grammar is not left recursive.\n");
   }
   return 0;
}
```

Output:

```
Original grammar:
E E+T T T*F F (E) id
Grammar after eliminating left recursion:
E T+E T F*T F (E) id
```

Practical-4

Q4. Write a program in C to remove left factoring.

```
#include<stdio.h>
#include<string.h>
void main()
{
   char gram[100], part1[20], part2[20], modifiedGram[20],newgram[20], newGram[20], tempGram[20];
   int i, j = 0, k = 0, l = 0, pos;
   printf("Enter the productions: A->");
   fgets(gram, sizeof(gram),stdin);
   for(i = 0; gram[i] != '|'; i++, j++)
   {
      part1[j] = gram[i];
   }part1[j] = '\0';
   for(j = ++i,i = 0;gram[j] != '\0';j++,i++)
   {
      part2[i] = gram[i];
   }part2[i] = '\0';
   for(i = 0; i < strlen(part1)||i < strlen(part2); i++)
   {
      if(part1[i] == part2[i])
      {
         modifiedGram[k] = part1[i];
         k++;
         pos = i + 1;
      }
   }
   for(i = pos, j = 0; part1[i] != '\0'; i++, j++)
   {
      newGram[j] = part1[i];
   }
   newGram[j++] = '|';
   for(i = pos; part2[i] != '\0';i++,j++)
   {
      newGram[j] = part2[i];
   }
   modifiedGram[k] = 'X';
   modifiedGram[++k] = '\0';
```

```
    newGram[j] = '\0';
    printf("\nGrammer Without Left Factoring : : \n");
    printf("A->%s", modifiedGram);
    printf("\nX->%s\n", newGram);
}
```

Output: