RECURSIVE DECENT PARSER:

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

#define SUCCESS 1
#define FAILED 0

int E(), Edash(), T(), Tdash(), F();

const char *cursor;
char string[64];

int main() {
	puts("Enter the string:");
	scanf("%s", string);
	cursor = string;
	puts("\nInput\tAction\n----------------------");
	if (E() && !*cursor) {
		puts("----------------------\nString parsed.");
		return 0;
	}
	puts("----------------------\nError in parsing.");
	return 1;
}

int E() { printf("%-16s E -> T E'\n", cursor); return T() && Edash(); }
int Edash() {
	if (*cursor == '+') { printf("%-16s E' -> + T E'\n", cursor); cursor++; return T() && Edash(); }
	printf("%-16s E' -> $\n", cursor); return SUCCESS;
}
```

```
int T() { printf("%-16s T -> F T'\n", cursor); return F() && Tdash(); }
int Tdash() {
        if (*cursor == '*') { printf("%-16s T' -> * F T'\n", cursor); cursor++; return F() && Tdash(); }
        printf("%-16s T' -> $\n", cursor); return SUCCESS;
}
int F() {
        if (*cursor == '(') { printf("%-16s F -> ( E )\n", cursor); cursor++; return E() && (*cursor == ')')
&& cursor++; }
        if (*cursor == 'i') { printf("%-16s F -> i\n", cursor); return cursor++, SUCCESS; }
        return FAILED;
}
```

SHIFT REDUCE PARSER

```
#include <stdio.h>
#include <string.h>
struct ProductionRule {
   char left[10], right[10];
};

int main() {
   char input[20], stack[50] = "", temp[50], ch[2], *token1, *token2, *substring;
   int i = 0, j, rule_count;
   struct ProductionRule rules[10];

   printf("\nEnter the number of production rules: ");
   scanf("%d", &rule_count);
   printf("\nEnter the production rules (left->right): \n");
   for (j = 0; j < rule_count; j++) {
      scanf("%s", temp);
      token1 = strtok(temp, "->");
      token2 = strtok(NULL, "->");
```

```
    strcpy(rules[j].left, token1);
    strcpy(rules[j].right, token2);
}

printf("\nEnter the input string: ");
scanf("%s", input);

while (1) {
    if (i < strlen(input)) {
        ch[0] = input[i++];
        ch[1] = '\0';
        strcat(stack, ch);
        printf("%s\t%s\tShift %s\n", stack, input + i, ch);
    }

    for (j = 0; j < rule_count; j++) {
        if ((substring = strstr(stack, rules[j].right)) != NULL) {
            stack[strlen(stack) - strlen(substring)] = '\0';
            strcat(stack, rules[j].left);
            printf("%s\t%s\tReduce %s->%s\n", stack, input + i, rules[j].left, rules[j].right);
            j = -1;
        }
    }

    if (strcmp(stack, rules[0].left) == 0 && i == strlen(input)) {
        printf("\nAccepted\n");
        break;
    }
    if (i == strlen(input)) {
        printf("\nNot Accepted\n");
        break;
```

```
        }
    }
    return 0;
}
```

Q. Write a C program to generate three address code of any statement.

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

int tempVar = 1;
void generateTemp(char *temp) { sprintf(temp, "t%d", tempVar++); }

void generateTAC(char *expr) {
    char opStack[20], output[20], temp[5];
    int opTop = -1, outTop = 0;

    for (int i = 0; expr[i]; i++) {
        char ch = expr[i];
        if (isalnum(ch)) output[outTop++] = ch;
        else if (ch == '(') opStack[++opTop] = ch;
        else if (ch == ')') {
            while (opTop >= 0 && opStack[opTop] != '(') output[outTop++] = opStack[opTop--];
            opTop--;
        } else {
            while (opTop >= 0 && opStack[opTop] != '(' && ((ch == '+' || ch == '-') || (opStack[opTop] != '+'
&& opStack[opTop] != '-')))
                output[outTop++] = opStack[opTop--];
            opStack[++opTop] = ch;
        }
    }
```

```
    while (opTop >= 0) output[outTop++] = opStack[opTop--];
    output[outTop] = '\0';

    char stack[20][5];
    int stackTop = -1;

    for (int i = 0; output[i]; i++) {
        char ch = output[i];
        if (isalnum(ch)) sprintf(stack[++stackTop], "%c", ch);
        else {
            char op1[5], op2[5];
            strcpy(op2, stack[stackTop--]);
            strcpy(op1, stack[stackTop--]);
            generateTemp(temp);
            printf("%s = %s %c %s\n", temp, op1, ch, op2);
            strcpy(stack[++stackTop], temp);
        }
    }

    printf("Result stored in: %s\n", stack[stackTop]);
}

int main() {
    char expr[20];
    printf("Enter an arithmetic expression: ");
    scanf("%s", expr);
    generateTAC(expr);
    return 0;
}
```

Q. **Write a C program to implement simple code generator.**

sample input :

x = a + b

y = x - c

exit

```
#include <stdio.h>
#include <string.h>
int regCount = 0;
char* getRegister() {
static char reg[5];
sprintf(reg, "R%d", regCount++);
return reg;
}
void resetRegisters() {
regCount = 0;
}
void generateCode(char* line, char* buffer) {
char lhs[10], op1[10], op2[10], op;
char reg1[5], reg2[5];
if (sscanf(line, "%s = %s %c %s", lhs, op1, &op, op2) == 4) {
strcpy(reg1, getRegister());
sprintf(buffer + strlen(buffer), "Mov %s, %s\n", reg1, op1);
strcpy(reg2, getRegister());
sprintf(buffer + strlen(buffer), "Mov %s, %s\n", reg2, op2);
switch (op) {
case '+':
sprintf(buffer + strlen(buffer), "Add %s, %s\n", reg1, reg2);
break;
```

```
case '-':
sprintf(buffer + strlen(buffer), "Sub %s, %s\n", reg1, reg2);
break;
case '*':
sprintf(buffer + strlen(buffer), "Mul %s, %s\n", reg1, reg2);
break;
case '/':
sprintf(buffer + strlen(buffer), "Div %s, %s\n", reg1, reg2);
break;
default:
sprintf(buffer + strlen(buffer), "Unsupported operator: %c\n", op);
return;
}
sprintf(buffer + strlen(buffer), "Mov %s, %s\n", lhs, reg1);
}
else if (sscanf(line, "%s = %s", lhs, op1) == 2) {
strcpy(reg1, getRegister());
sprintf(buffer + strlen(buffer), "Mov %s, %s\n", reg1, op1);
sprintf(buffer + strlen(buffer), "Mov %s, %s\n", lhs, reg1);
}
}
int main() {
char line[50];
char buffer[1000] = "";
printf("Enter expressions (type 'exit' to stop):\n");
while (1) {
fgets(line, sizeof(line), stdin);
line[strcspn(line, "\n")] = 0;
if (strcmp(line, "exit") == 0) {
break;
}
```

```
generateCode(line, buffer);
}
printf("\nGenerated Code:\n%s", buffer);
resetRegisters();
return 0;
}
```

Q. Write a C / C++ program to accept a C program and perform error detection& correction for the

following:

a) Check for un-terminated string constant and single character constant in the input C program.

i.e A string constant begins with double quotes and extends to more than one line.

b) Report the error line numbers and the corrective actions to user.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
ifstream infile("file.txt");
string line;
int line_number = 0;
int line_error;
int i;
bool flag = false;
while (getline(infile, line))
{
line_number++;
```

```
if (!flag)
{
for (i = 0; i < line.length(); i++)
{
if (line[i] == '"')
{
flag = true;
line_error = line_number;
break;
}
}
}
i++;
if (flag)
{
for (; i < line.length(); i++)
{
if (line[i] == '"')
{
flag = false;
break;
}
}
}
}
if (!flag)
{
cout << "No error" << endl;
}
```

```
else
{
cout << "String opened at " << line_error << " but not closed" << endl;
}
}
```

file.txt

```
#include<stdio.h>
#include<conio.h>
int s[35]="gh";
void main()
{
int a;
char c[10]="msrit",f[]="lk";
strlen("hjkl");
a=a+/(*b);
}
```

Q. Write a C / C++ program to accept a C program and perform error detection& correction ,

indicate the user for the following :

a) Check whether the multi-line comment statement is terminated correctly or not.

b) Check whether the single line comment statement is existing in your C program and report the line numbers to the user

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
```

```
ifstream infile("test1.txt");
string line;
int line_number = 0;
int line_error;
bool multi_line_flag = false;
bool single_line_flag = false;
while (getline(infile, line)) {
line_number++;
if (!multi_line_flag) {
for (size_t i = 0; i < line.length(); i++) {
if (line[i] == '/' && i + 1 < line.length() && line[i + 1] == '*') {
multi_line_flag = true;
line_error = line_number;
break;
}
}
}

if (multi_line_flag) {
for (size_t i = 0; i < line.length(); i++) {
if (line[i] == '*' && i + 1 < line.length() && line[i + 1] == '/') {
multi_line_flag = false;
break;
}
}
}
if (!single_line_flag) {
for (size_t i = 0; i < line.length(); i++) {
if (line[i] == '/' && i + 1 < line.length() && line[i + 1] == '/') {
single_line_flag = true;
line_error = line_number;
```

```
 break;
 }
 }
 }
 }
 if (!multi_line_flag && !single_line_flag) {
 cout << "No error" << endl;
 } else if (multi_line_flag) {
 cout << "Multiline comment started at " << line_error << " but not closed" <<
endl;
 } else if (single_line_flag) {
 cout << "Single line comment at line " << line_error << endl;
 }
 return 0;
}
```

test1.txt

```
#include <stdio.h>
int main()
{
 /* Hello world */
 printf("Hello World");
 /* Hello world */
 /* Hello world */
 /* hi
}
```

Q. Write a program to separate the tokens for given input program.

sample input file:

```
#include<stdio.h>
void main()
{
        printf("Hello World");
}
```

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void tokenizeFile(const char *filename) {
FILE *file = fopen(filename, "r");
if (!file) {
printf("Error: Could not open file %s\n", filename);
return;
}
char line[256];
int lineNumber = 0, tokenNumber = 0;
printf("Line no.\tToken no.\tToken\t\tLexeme\n");
while (fgets(line, sizeof(line), file)) {
lineNumber++;
char *token = strtok(line, " \t\n<>#();\"{}");
while (token != NULL) {
char tokenType[20] = "Identifier";
if (strcmp(token, "void") == 0 || strcmp(token, "main") == 0) {
strcpy(tokenType, "Keyword");
} else if (strchr("{}", token[0]) || strchr("()", token[0]) || strchr(";", token[0])) {
strcpy(tokenType, "Special symbol");
}
```

```
printf("%7d\t%10d\t%15s\t%10s\n", lineNumber, tokenNumber++, tokenType, token);
token = strtok(NULL, " \t\n<>#();\"{}");
}
}
fclose(file);
}
int main() {
char filename[100];
printf("Enter the name of the input file: ");
scanf("%s", filename);
tokenizeFile(filename);
return 0;
}
```

Input.c

```
#include<stdio.h>
void main()
{
printf("Hello World");
}
```

Q. Write a program to implement symbol table

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>
#define MAX 100
```

```
typedef struct {
    char name[50], type[20];
    int line;
} Symbol;

Symbol symbolTable[MAX];
int count = 0;

bool isKeyword(const char *word) {
    const char *keywords[] = {"auto", "break", "case", "char", "const", "continue", "default", "do", "double", "else", "enum", "extern", "float", "for", "goto", "if", "int", "long", "register", "return", "short", "signed", "sizeof", "static", "struct", "switch", "typedef", "union", "unsigned", "void", "volatile", "while"};
    for (int i = 0; i < 32; i++) if (strcmp(word, keywords[i]) == 0) return true;
    return false;
}

void insertSymbol(char *name, char *type, int line) {
    for (int i = 0; i < count; i++) if (strcmp(symbolTable[i].name, name) == 0) return;
    strcpy(symbolTable[count].name, name);
    strcpy(symbolTable[count].type, type);
    symbolTable[count++].line = line;
}

void displaySymbolTable() {
    if (!count) { printf("Symbol table is empty.\n"); return; }
    printf("\nSymbol Table:\n----------------------------------------------------\n| %-15s | %-10s | %-5s |\n----------------------------------------------------\n", "Name", "Type", "Line");
    for (int i = 0; i < count; i++) printf("| %-15s | %-10s | %-5d |\n", symbolTable[i].name, symbolTable[i].type, symbolTable[i].line);
    printf("----------------------------------------------------\n");
}
```

```
void processFile(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (!file) { printf("Error: Could not open file %s\n", filename); return; }

    char line[256], word[50], type[20];
    int lineNumber = 0;
    while (fgets(line, sizeof(line), file)) {
        lineNumber++;
        char *token = strtok(line, " \t\n;(){}[],");
        while (token) {
            if (isKeyword(token)) strcpy(type, token);
            else if (isalpha(token[0]) || token[0] == '_') insertSymbol(token, type, lineNumber);
            token = strtok(NULL, " \t\n;(){}[],");
        }
    }
    fclose(file);
    printf("Symbol table generated successfully for %s.\n", filename);
}

int main() {
    char filename[100];
    printf("Enter the name of the .c file: ");
    scanf("%s", filename);
    if (strstr(filename, ".c") == NULL) { printf("Error: Please provide a valid .c file.\n"); return 1; }
    processFile(filename);
    displaySymbolTable();
    return 0;
}
```

Input.c

```
#include <stdio.h>

int main() {

int x = 10;

float y = 20.5;

char z = 'a';

printf("Hello, World!");

return 0;

}
```

1) PROGRAM TO RECOGNIZE INTEGER, REAL AND EXPONENTIAL

```
%{
#include<stdio.h>
%}
sign [+-]?
digit [0-9]+
exp ([eE]{sign}{digit})
%%
\+?{digit} printf("\n Number is positive...\n");
\-{digit} printf("\n Number is negative...\n");
{sign}{digit}?\.{digit}? printf("\n Number is real...\n");
{sign}{digit}(\.{digit}?)?{exp} printf("\n Number is exponential...\n");
%%
int yywrap()
{
return 1;
}
int main()
{
char myString[100];
fgets (myString, sizeof(myString), stdin);
yy_scan_string(myString);
yylex ();
}
```

2) LEX Program to scan for even and odd numbers

```
%{
/*
1.Request input of an even and an odd number
2.indicate input characteristic : Even/Odd
3.check for input's correctness and print result
*/
#include <stdlib.h>
#include <stdio.h>
int number_1;
int number_2;
%}
number_sequence [0-9]*
%%
{number_sequence}[0|2|4|6|8] {
                              printf("Even number [%d]",yyleng);
                              return 1;   }
{number_sequence}[1|3|5|7|9] {
                              printf("Odd number [%d]",yyleng);
                              return 1;
                             }
%%
int yywrap()
{
  return 1;
}
int main()
{
  printf("\nInput two numbers\n");
  number_1 = yylex();
  number_2 = yylex();
  return 1;
}
```

3) PROGRAM TO COUNT THE NUMBER OF LINES USING FILE

```
%{
int lineno=0;
%}
%%
^(.*)\n {lineno++;} printf("%4d\t%s",lineno,yytext);
%%
int yywrap()
{
return 1;
}
int main()
{
yyin=fopen("input.txt","r");
yylex();
fclose(yyin);
}
```

4) PROGRAM TO REMOVE UPPERCASE AND WHITESPACE.

```
%{
%}
%%
[A-Z]+ ;
[ \t\n,"]+ ; //removes space, tabspace, newline space, comma and double quote
%%
int yywrap()
{
return 1;
}
int main()
{
yyin=fopen("input.txt","r");
yylex();
fclose(yyin);
}
```

5) LEX program to count the number of vowels and consonants in a given string

```
%{
    int vow_count=0;
    int const_count =0;
%}
%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
int yywrap()
{
  return 1;
}
int main()
{
  printf("Enter the string of vowels and consonants:");
  char myString[100];
  fgets (myString, sizeof(myString), stdin);
  yy_scan_string(myString);
  yylex();
  printf("Number of vowels are:  %d\n", vow_count);
  printf("Number of consonants are:  %d\n", const_count);
}
```

6) Lex Program to check whether a number is Prime or Not

```
%{
/* Definition section */
#include<stdio.h>
#include<stdlib.h>
int flag,c,j;
%}
/* Rule Section */
%%
[0-9]+ {c=atoi(yytext);
                        if(c==2)
                        {   printf("\n Prime number");   }
                        else if(c==0 || c==1)
                        {   printf("\n Not a Prime number");  }
                        else
                        {
                        for(j=2;j<c;j++)
                        {
                        if(c%j==0)
                        flag=1;
                        }
                        if(flag==1)
                        printf("\n Not a prime number");
                        else if(flag==0)
                        printf("\n Prime number");
                        return 1;    }   }
%%
int yywrap()
{
  return 1;   }
int main()
{
yylex();
return 0;
}
```

7) Lex program to check if a date is valid or not. Format of Date is DD/MM/YYYY.

```
%{
  /* Definition section */
  #include<stdio.h>
  int i=0, yr=0, valid=0;
%}
/* Rule Section */
%%
([0-2][0-9]|[3][0-1])\/((0(1|3|5|7|8))|(10|12))\/([1-2][0-9][0-9][-0-9]) {valid=1;return 1;}

([0-2][0-9]|30)\/((0(4|6|9))|11)\/([1-2][0-9][0-9][0-9]) {valid=1;return 1;}

([0-1][0-9]|2[0-8])\/02\/([1-2][0-9][0-9][0-9]) {valid=1;return 1;}

29\/02\/([1-2][0-9][0-9][0-9])     { while(yytext[i]!='/')i++;       while(yytext[i]!='/')i++;       while(i<yyleng) yr=(10*yr)+(yytext[i++]-'0');
if(yr%4==0||(yr%100==0&&yr%400!=0)) valid=1; return 1;}
%%
int yywrap()
{
 return 1;
}
int main()
{
  yylex();
  if(valid==1)
  printf("It is a valid date\n");
  else
  printf("It is not a valid date\n");
return 0;
}
```

8) Write a Lex program for checking a valid URL.

```
%{
#include <stdio.h>
%}
%%
(http|https)://[a-zA-Z0-9-.]+\.[a-zA-Z]{2,}(/[a-zA-Z0-9._~:/?#@!$&'()*+,;=-]*)? {
    printf("Valid URL: %s\n", yytext);
}
.|\n {
    printf("Invalid URL: %s\n", yytext);
}
%%
int main() {
    printf("Enter a URL: ");
    yylex();
    return 0;
}
int yywrap() {
    return 1;   }
```

9)Lex program to count the frequency of the given word in a file

```
%{
#include<stdio.h>
#include<string.h>
char word[50];
int count = 0;
%}
%%
{word} { count++; }
.|\n ;
%%
int yywrap() { return 1; }
int main() {
    printf("Enter the word to search for: ");
    scanf("%s", word);
    yyin = fopen("input.txt", "r");
    if (!yyin) {
        printf("Error: Could not open file 'input.txt'.\n");
        return 1;
    }
    yylex();
    printf("The word '%s' appears %d time(s) in the file.\n", word, count);
    fclose(yyin);
    return 0;  }
```

10) LEX code to extract HTML tags from a file

```
%{
#include<stdio.h>
int tag_count = 0;
%}
%%
<[^>]+> {
    printf("HTML Tag: %s\n", yytext);
    tag_count++;
}
.|\n ;
%%
int yywrap() {
    return 1;
}
int main() {
    yyin = fopen("input.html", "r");
    if (!yyin) {
        printf("Error: Could not open file 'input.html'.\n");
        return 1;
    }
    yylex();
    printf("Total number of HTML tags: %d\n", tag_count);
    fclose(yyin);
    return 0;
}
```