1. C Program to check whether a given line is a comment or not

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

bool isComment(char *line) {
    // Check if the line starts with "//" or "/*"
    if (strncmp(line, "//", 2) == 0 || strncmp(line, "/*", 2) == 0) {
        return true;
    }
    return false;
}

int main() {
    char line[100];

    printf("Enter a line of code: ");
    fgets(line, sizeof(line), stdin);

    if (isComment(line)) {
        printf("The given line is a comment.\n");
    } else {
        printf("The given line is not a comment.\n");
    }

    return 0;
}
```

1.1 (txt input) C Program to count no of single and multi-line comments

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

/**
 * Using preprocessor macros instead of enums, per request; normally
 * I would use enums, since they obey scoping rules and
 * show up in debuggers.
 */
```

```
#define TEXT          0
#define SAW_SLASH      1
#define SAW_STAR       2
#define SINGLE_COMMENT 3
#define MULTI_COMMENT  4

#define TOTAL_STATES   5

#define NO_ACTION      0
#define INC_TOTAL      1
#define INC_SINGLE     2
#define INC_MULTI      4

/**
 * This example assumes 7-bit ASCII, for a total of
 * 128 character encodings.  You'll want to change this
 * to handle other encodings.
 */
#define ENCODINGS    128

/**
 * Need a state table to control state transitions and an action
 * table to specify what happens on a transition.  Each table
 * is indexed by the state and the next input character.
 */
static int  state[TOTAL_STATES][ENCODINGS]; // Since these tables are declared at file scope,
they will be initialized to
static int action[TOTAL_STATES][ENCODINGS]; // all elements 0, which correspond to the
"default" states defined above.

/**
 * Initialize our state table.
 */
void initState( int (*state)[ENCODINGS] )
{
  /**
   * If we're in the TEXT state and see a '/' character, move to the SAW_SLASH
   * state, otherwise stay in the TEXT state
   */
  state[TEXT]['/'] = SAW_SLASH;

  /**
   * If we're in the SAW_SLASH state, we can go one of three ways depending
   * on the next character.
   */
  state[SAW_SLASH]['/'] = SINGLE_COMMENT;
  state[SAW_SLASH]['*'] = MULTI_COMMENT;
  state[SAW_SLASH]['\n'] = TEXT;

  /**
   * For all but a few specific characters, if we're in any one of
   * the SAW_STAR, SINGLE_COMMENT, or MULTI_COMMENT states,
   * we stay in that state.
   */
  for ( size_t i = 0; i < ENCODINGS; i++ )
  {
    state[SAW_STAR][i] = MULTI_COMMENT;
    state[SINGLE_COMMENT][i] = SINGLE_COMMENT;
    state[MULTI_COMMENT][i] = MULTI_COMMENT;
  }

  /**
   * Exceptions to the loop above.
   */
  state[SAW_STAR]['/'] = TEXT;
  state[SAW_STAR]['*'] = SAW_STAR;

  state[SINGLE_COMMENT]['\n'] = TEXT;
  state[MULTI_COMMENT]['*'] = SAW_STAR;
}

/**
 * Initialize our action table
 */
void initAction( int (*action)[ENCODINGS] )
{
  action[TEXT]['\n'] = INC_TOTAL;
  action[SAW_STAR]['/'] = INC_MULTI;
  action[MULTI_COMMENT]['\n'] = INC_MULTI | INC_TOTAL;   // Multiple actions are
bitwise-OR'd
  action[SINGLE_COMMENT]['\n'] = INC_SINGLE | INC_TOTAL; // together
  action[SAW_SLASH]['\n'] = INC_TOTAL;
}

/**
 * Scan the input file for comments
 */
void countComments( FILE *stream, size_t *totalLines, size_t *single, size_t *multi )
```

```
{
  *totalLines = *single = *multi = 0;

  int c;
  int curState = TEXT, curAction = NO_ACTION;

  while ( ( c = fgetc( stream ) ) != EOF )
  {
    curAction = action[curState][c]; // Read the action before we overwrite the state
    curState = state[curState][c];   // Get the new state (which may be the same as the old state)

    if ( curAction & INC_TOTAL )     // Execute the action.
      (*totalLines)++;

    if ( curAction & INC_SINGLE )
      (*single)++;

    if ( curAction & INC_MULTI )
      (*multi)++;
  }
}

/**
 * Main function.
 */
int main( int argc, char **argv )
{
  /**
   * Input sanity check
   */
  if ( argc < 2 )
  {
    fprintf( stderr, "USAGE: %s <filename>\n", argv[0] );
    exit( EXIT_FAILURE );
  }

  /**
   * Open the input file
   */
  FILE *fp = fopen( argv[1], "r" );
  if ( !fp )
  {
    fprintf( stderr, "Cannot open file %s\n", argv[1] );
    exit( EXIT_FAILURE );
  }

  /**
   * If input file was successfully opened, initialize our
   * state and action tables.
   */
  initState( state );
  initAction( action );

  size_t totalLines, single, multi;

  /**
   * Do the thing.
   */
  countComments( fp, &totalLines, &single, &multi );
  fclose( fp );

  printf( "File                  : %s\n", argv[1] );
  printf( "Total lines           : %zu\n", totalLines );
  printf( "Single-comment lines : %zu\n", single );
  printf( "Multi-comment lines  : %zu\n", multi );

  return EXIT_SUCCESS;
}
```
input.txt
```
#include <stdio.h>

// Function to add two integers
int add(int a, int b) {
    return a + b;
}

/* Function to subtract two integers */
int subtract(int a, int b) {
    return a - b;
}

int main() {
    int x = 10; // Variable x
    int y = 5;  // Variable y

    // Print the result of addition
    printf("Sum: %d\n", add(x, y));

    /*
    Print the result of subtraction
    Multi-line comment example
    */
    printf("Difference: %d\n", subtract(x, y));

    return 0;
}
```
terminal
```
❯ ls
input.txt q1p2.c

  ~/Desktop/final ··························· base   at 10:44:16
❯ gcc q1p2.c

  ~/Desktop/final ··························· base   at 10:44:38
❯ ls
a.out     input.txt q1p2.c

  ~/Desktop/final ··························· base   at 10:44:41
❯ ./a.out input.txt
File                 : input.txt
Total lines          : 26
Single-comment lines : 4
Multi-comment lines  : 5

  ~/Desktop/final ··························· base   at 10:45:11
❯
```

1.2 Check single comment from txt file and give line number

```
#include <stdio.h>
#include <string.h>
int main(){
int s=0,line=1;
```

~~~~
FILE *file= fopen("myfile.txt","r");
if(file==NULL)
{
printf("error in reading the file.");
return 1;
}
char buffer[1024];
while(fgets(buffer,1024,file)){
for(int i=0;i<=sizeof(buffer);i++)
{
if(buffer[i] =='/'&& buffer[i+1]=='/'){
s++;
printf("line no %d \n",line);
}
}
line++;
}
printf("%d \n",s);
fclose(file);
return 0;
}
```
myfile.txt
```
#include <stdio.h>
int main()
{
// declare the variables
int a=5;
int b=8;
int c=a*b;  //multiply two numbers
//print the product
printf( “The sum is %d",c);
 return 0;
}
```
~~~~

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

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

bool matchA(char *str) {
    return (strlen(str) == 1 && str[0] == 'a');
}

bool matchAStarBPlus(char *str) {
    int length = strlen(str);
    if (length == 0)
        return false;

    bool foundA = false;
    bool foundB = false;

    for (int i = 0; i < length; i++) {
        if (str[i] == 'a') {
            foundA = true;
        } else if (foundA && str[i] == 'b') {
            foundB = true;
        } else if (!foundA || foundB || str[i] != 'b') {
            return false;
        }
    }

    return foundA && foundB;
}

bool matchABB(char *str) {
    int length = strlen(str);
    if (length != 3)
        return false;

    return (str[0] == 'a' && str[1] == 'b' && str[2] == 'b');
}

int main() {
    char input[100];
    printf("Enter a string: ");
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = '\0'; // Remove newline character

    if (matchA(input)) {
        printf("The string matches the pattern 'a'.\n");
    } else if (matchABB(input)) {
        printf("The string matches the pattern 'abb'.\n");
    } else if (matchAStarBPlus(input)) {
        printf("The string matches the pattern 'a*b+'.\n");
    } else {
        printf("The string does not match any recognized pattern.\n");
    }

    return 0;
}
```

3. Write a C program to test whether a given identifier is valid or not.

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

bool isValidIdentifier(char *identifier) {
    // Check if the first character is alphabetic or underscore
    if (!isalpha(identifier[0]) && identifier[0] != '_')
        return false;

    // Check each character for validity
    for (int i = 1; i < strlen(identifier); i++) {
        if (!isalnum(identifier[i]) && identifier[i] != '_')
            return false;
    }

    // Check if the identifier is not a keyword
    char keywords[32][10] = {"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", "main", "return", "for", "include", "case", "bool"};

    for (int i = 0; i < 32; i++) {
        if (strcmp(identifier, keywords[i]) == 0)
            return false;
    }

    return true;
}

int main() {
    char identifier[50];

    printf("Enter an identifier: ");
    scanf("%s", identifier);

    if (isValidIdentifier(identifier)) {
        printf("'%s' is a valid identifier.\n", identifier);
    } else {
        printf("'%s' is not a valid identifier.\n", identifier);
    }

    return 0;
}
```

## 4. Write a C program to simulate lexical analyzer for validating operators.
```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

// Function to check if a character is a valid operator
bool isValidOperator(char c) {
    char operators[] = "+-*/%=><&|^!~";

    for (int i = 0; i < strlen(operators); i++) {
        if (c == operators[i]) {
            return true;
        }
    }
    return false;
}
```

```
int main() {
    char input[100];

    printf("Enter a string to validate operators: ");
    fgets(input, sizeof(input), stdin);

    printf("Valid operators in the string:\n");
    for (int i = 0; i < strlen(input); i++) {
        if (isValidOperator(input[i])) {
            printf("%c ", input[i]);
        }
    }
    printf("\n");

    return 0;
}
```
example
```
Enter a string to validate operators: +-*&=Hello!$^
Valid operators in the string:
+ - * & =
```

4. Lexical analyzer for validating operators (myfile.txt) file

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

// Function to check if an operator is valid
bool isValidOperator(char op) {
    switch (op) {
        case '+':
        case '-':
        case '*':
        case '/':
        case '%':
        case '=':
        case '<':
```

```
        case '>':
        case '&':
        case '|':
        case '!':
            return true;
        default:
            return false;
    }
}

int main() {
    FILE *inputFile;
    char operator;

    if ((inputFile = fopen("myfile.txt", "r")) == NULL) {
        printf("Unable to open the input file.\n");
        return 1;
    }

    printf("Operators:\n");

    while ((operator = fgetc(inputFile)) != EOF) {
        if (!isspace(operator)) {
            if (isValidOperator(operator)) {
                printf("%c is a valid operator.\n", operator);
            }
        }
    }

    fclose(inputFile);

    return 0;
}
```

## 5. Write a program to find out the total number of vowels, and consonants from the given input string.

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

// Function to check if a character is a vowel
int isVowel(char ch) {
    ch = toupper(ch); // Convert to uppercase for easier comparison
    return (ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U');
}

int main() {
    char input[100];
    int vowels = 0, consonants = 0;

    printf("Enter a string: ");
    fgets(input, sizeof(input), stdin);

    for (int i = 0; input[i] != '\0'; i++) {
        if (isalpha(input[i])) { // Check if the character is an alphabet
            if (isVowel(input[i])) {
                vowels++;
            } else {
                consonants++;
            }
        }
    }

    printf("Total number of vowels: %d\n", vowels);
    printf("Total number of consonants: %d\n", consonants);

    return 0;
}
```

5.1 Write a program in C to design a Lexical Analyser that identifies the keywords, identifiers, separators, and operators present in a input file (code.txt) code.

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

int isKeyword(char *word) {
    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","main","return","for","include","case","bool"};

    int numKeywords = sizeof(keywords) / sizeof(keywords[0]);
    for (int i = 0; i < numKeywords; i++) {
        if (strcmp(word, keywords[i]) == 0) {
            return 1;
        }
    }
    return 0;
}

int isOperator(char c) {
    char operators[] = "+-*/%<>=&|^!";
    for (int i = 0; operators[i] != '\0'; i++) {
        if (c == operators[i]) {
            return 1;
        }
    }
    return 0;
}

int isSeparator(char c) {
    char separators[] = "(){}[],;:";
    for (int i = 0; separators[i] != '\0'; i++) {
        if (c == separators[i]) {
            return 1;
        }
    }
    return 0;
}

int isDuplicate(char *word, char **array, int count) {
    for(int i=0;i<count;i++) {
        if (strcmp(word, array[i]) == 0) {
            return 1;
        }
    }
    return 0;
}

int isDuplicateOperator(char c, char array[], int count) {
```

```c
    for(int i=0;i<count;i++){
        if (c == array[i]) {
            return 1;
        }
    }
    return 0;
}

int isDuplicateSeparator(char c, char array[], int count) {
    for(int i=0;i<count;i++){
        if (c == array[i]) {
            return 1;
        }
    }
    return 0;
}

int main() {

    char *filename = "code.txt"; FILE *fp = fopen(filename, "r");

    if (fp == NULL) {
        printf("Error Occurred / File doesn't exist\n");
        return 1;
    }

    char line[256];
    char word[256];
    char *keywords[256];
    int keyword_count = 0;
    char *identifiers[256];
    int identifier_count = 0;
    char operators[256];
    int operator_count = 0;
    char separators[256];
    int separator_count = 0;

    while (fgets(line, 256, fp)) {
        int word_start = -1;
        for (int i = 0; i <= strlen(line); i++) {
            if (line[i] == '#') {
                break;
            }
            if (isalnum(line[i]) || line[i] == '_') {
```

```
            if (word_start == -1) {
                word_start = i;
            }
        }else{
            if (word_start != -1) {
                int len = i - word_start;
                strncpy(word, &line[word_start], len);
                word[len] = '\0';
                if (!isDuplicate(word, keywords, keyword_count) && isKeyword(word)) {
                    keywords[keyword_count++] = strdup(word);
                } else if (!isDuplicate(word, identifiers, identifier_count)) {
                    identifiers[identifier_count++] = strdup(word);
                }
                word_start = -1;
            }
            if (isOperator(line[i]) && !isDuplicateOperator(line[i], operators, operator_count)) {
                operators[operator_count++] = line[i];
            }
            if (isSeparator(line[i]) && !isDuplicateSeparator(line[i], separators, separator_count)) {
                separators[separator_count++] = line[i];
            }
        }
    }
}
printf("Keywords: ");
for (int i = 0; i < keyword_count; i++) {
    printf("%s ", keywords[i]);
    free(keywords[i]);
}
printf("\nIdentifiers: ");
for (int i = 0; i < identifier_count; i++) {
    printf("%s ", identifiers[i]);
    free(identifiers[i]);
}
printf("\nOperators: ");
for (int i = 0; i < operator_count; i++) {
    printf("%c ", operators[i]);
}
printf("\nSeparators: ");
for (int i = 0; i < separator_count; i++) {
    printf("%c ", separators[i]);
}
fclose(fp);
return 0;
```

```
}
```

6. Write a C program to create a symbol table.

## You can take help from the below mentioned algorithm to replicate the actions of the symbol table

```
Algorithm:
Start the program for performing insert, display, delete, search and modify option in symbol table
Define the structure of the Symbol Table
Enter the choice for performing the operations in the symbol Table
If the entered choice is 1, search the symbol table for the symbol to be inserted. If the symbol is already present, it displays “Duplicate Symbol”. Else, insert the symbol and the corresponding address in the symbol table.
If the entered choice is 2, the symbols present in the symbol table are displayed.
If the entered choice is 3, the symbol to be deleted is searched in the symbol table.
If it is not found in the symbol table it displays “Label Not found”. Else, the symbol is deleted.
If the entered choice is 5, the symbol to be modified is searched in the symbol table. The label or address or both can be modified.
```

Code:

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

// Define the structure of the Symbol Table
struct SymbolTable {
    char label[50];
    int address;
};

// Global variables
struct SymbolTable symbolTable[100]; // Assuming a maximum of 100 entries
int count = 0; // To keep track of the number of symbols in the table

// Function to insert a symbol into the symbol table
void insertSymbol(char label[], int address) {
    // Check if the symbol already exists
    for (int i = 0; i < count; i++) {
        if (strcmp(symbolTable[i].label, label) == 0) {
            printf("Duplicate Symbol\n");
            return;
        }
    }

    // Insert the symbol and address into the table
    strcpy(symbolTable[count].label, label);
    symbolTable[count].address = address;
    count++;
    printf("Symbol inserted successfully\n");
}

// Function to display all symbols in the symbol table
void displaySymbols() {
    if (count == 0) {
        printf("Symbol Table is empty\n");
        return;
    }

    printf("Symbol Table:\n");
    printf("Label\tAddress\n");
    for (int i = 0; i < count; i++) {
        printf("%s\t%d\n", symbolTable[i].label, symbolTable[i].address);
    }
}

// Function to delete a symbol from the symbol table
void deleteSymbol(char label[]) {
    int found = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(symbolTable[i].label, label) == 0) {
            // Delete the symbol by shifting elements
            for (int j = i; j < count - 1; j++) {
                strcpy(symbolTable[j].label, symbolTable[j + 1].label);
                symbolTable[j].address = symbolTable[j + 1].address;
            }
            count--;
            found = 1;
            printf("Symbol deleted successfully\n");
            break;
        }
    }
    if (!found) {
        printf("Label Not found\n");
    }
```

```
}

// Function to modify a symbol in the symbol table
void modifySymbol(char label[], int newAddress) {
    int found = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(symbolTable[i].label, label) == 0) {
            symbolTable[i].address = newAddress;
            found = 1;
            printf("Symbol modified successfully\n");
            break;
        }
    }
    if (!found) {
        printf("Label Not found\n");
    }
}

int main() {
    int choice;
    char label[50];
    int address, newAddress;

    while (1) {
        printf("\nSymbol Table Operations:\n");
        printf("1. Insert Symbol\n");
        printf("2. Display Symbols\n");
        printf("3. Delete Symbol\n");
        printf("4. Modify Symbol\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter symbol label: ");
                scanf("%s", label);
                printf("Enter address: ");
                scanf("%d", &address);
                insertSymbol(label, address);
                break;
            case 2:
                displaySymbols();
                break;
            case 3:
                printf("Enter symbol label to delete: ");
                scanf("%s", label);
                deleteSymbol(label);
                break;
            case 4:
                printf("Enter symbol label to modify: ");
                scanf("%s", label);
                printf("Enter new address: ");
                scanf("%d", &newAddress);
                modifySymbol(label, newAddress);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```
Sample
```
Symbol Table Operations:
1. Insert Symbol
2. Display Symbols
3. Delete Symbol
4. Modify Symbol
5. Exit
Enter your choice: 1
Enter symbol label: var1
Enter address: 100
Symbol inserted successfully
```

7. Write a C program to detect the datatype of a user input and store it in an appropriate identifier.

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

```
#include <stdbool.h>
#include <ctype.h>

// Function to check if the input is an integer
bool isInteger(const char *input) {
    char *endptr;
    strtol(input, &endptr, 10);
    return (*endptr == '\0');
}

// Function to check if the input is a floating-point number
bool isFloat(const char *input) {
    char *endptr;
    strtof(input, &endptr);
    return (*endptr == '\0');
}

// Function to check if the input is a character
bool isCharacter(const char *input) {
    return (strlen(input) == 1 && isalpha(input[0]));
}

int main() {
    char input[100];
    printf("Enter a value: ");
    scanf("%s", input);

    if (isInteger(input)) {
        int intValue = atoi(input);
        printf("Detected datatype: Integer\n");
        printf("Stored in identifier 'integerValue': %d\n", intValue);
    } else if (isFloat(input)) {
        float floatValue = atof(input);
        printf("Detected datatype: Float\n");
        printf("Stored in identifier 'floatValue': %f\n", floatValue);
    } else if (isCharacter(input)) {
        printf("Detected datatype: Character\n");
        printf("Stored in identifier 'charValue': %c\n", input[0]);
    } else {
        printf("Detected datatype: String\n");
        printf("Stored in identifier 'stringValue': %s\n", input);
    }
    return 0;
}
```

8)i) Write a C program to remove left recursion from a given grammar.

```
#include<stdio.h>
#include<string.h>
#define SIZE 10

int main () {
    char non_terminal;
    char beta,alpha;
    int num;
    char production[10][SIZE];
    int index=3; /* starting of the string following "->" */
    printf("Enter Number of Production : ");
    scanf("%d",&num);
    printf("Enter the grammar as E->E-A :\n");
    for(int i=0;i<num;i++){
        scanf("%s",production[i]);
    }
    for(int i=0;i<num;i++){
        printf("\nGRAMMAR : : : %s",production[i]);
        non_terminal=production[i][0];
        if(non_terminal==production[i][index]) {
            alpha=production[i][index+1];
            printf(" is left recursive.\n");
            while(production[i][index]!=0 && production[i][index]!='|')
                index++;
            if(production[i][index]!=0) {
                beta=production[i][index+1];
                printf("Grammar without left recursion:\n");
                printf("%c->%c%c\'",non_terminal,beta,non_terminal);
                printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);
            }
            else
                printf(" can't be reduced\n");
        }
        else
            printf(" is not left recursive.\n");
        index=3;
    }
}
```
example
```
Enter Number of Production : 3
Enter the grammar as E->E-A :
E->E+T|T
T->T*F|F
F->(E)|id

GRAMMAR : : : E->E+T|T is left recursive.
Grammar without left recursion:
E->TE'
E'->+E'|E

GRAMMAR : : : T->T*F|F is left recursive.
Grammar without left recursion:
T->FT'
T'->*T'|E

GRAMMAR : : : F->(E)|id is not left recursive.
```

8)ii) Write a C program to remove left factoring from a given grammar

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

#define MAX_PROD_LEN 20
#define MAX_PROD_NUM 10

// Structure to represent a production
typedef struct Production {
    char lhs; // Left hand side of the production
    char rhs[MAX_PROD_LEN]; // Right hand side of the production
} Production;

// Function prototypes
void remove_left_factoring(Production* grammar, int* grammar_size);
```

```
int main() {
    // Define your grammar here
    Production grammar[MAX_PROD_NUM] = {
        {'S', "aAb"},
        {'S', "aAc"},
        {'A', "d"},
        {'A', ""}
    };

    int grammar_size = 4; // Number of productions in the grammar

    printf("Original Grammar:\n");
    for (int i = 0; i < grammar_size; i++) {
        printf("%c -> %s\n", grammar[i].lhs, grammar[i].rhs);
    }

    remove_left_factoring(grammar, &grammar_size);

    printf("\nGrammar after Left Factoring:\n");
    for (int i = 0; i < grammar_size; i++) {
        printf("%c -> %s\n", grammar[i].lhs, grammar[i].rhs);
    }

    return 0;
}

// Function to remove left factoring from the grammar
void remove_left_factoring(Production* grammar, int* grammar_size) {
    for (int i = 0; i < *grammar_size; i++) {
        char common_prefix[MAX_PROD_LEN] = "";
        int prefix_length = 0;
        for (int j = i + 1; j < *grammar_size; j++) {
            int k;
            for (k = 0; grammar[i].rhs[k] && grammar[j].rhs[k]; k++) {
                if (grammar[i].rhs[k] != grammar[j].rhs[k]) {
                    break;
                }
                common_prefix[k] = grammar[i].rhs[k];
            }
            common_prefix[k] = '\0';
            if (k > prefix_length) {
                prefix_length = k;
            }
        }

        if (prefix_length > 0) {
            // Create new non-terminal for common prefix
            char new_lhs = grammar[*grammar_size - 1].lhs + 1;

            // Update original production
            strcpy(grammar[i].rhs, &grammar[i].rhs[prefix_length]);
            grammar[i].lhs = new_lhs;

            // Add new productions for common prefix
            for (int j = i + 1; j < *grammar_size; j++) {
                if (strncmp(grammar[j].rhs, common_prefix, prefix_length) == 0) {
                    strcpy(grammar[j].rhs, &grammar[j].rhs[prefix_length]);
                    grammar[j].lhs = new_lhs;
                }
            }

            // Add new production for common prefix
            grammar[*grammar_size].lhs = new_lhs;
            strcpy(grammar[*grammar_size].rhs, common_prefix);
            (*grammar_size)++;
        }
    }
}
```

Boxofcodes

```
#include<stdio.h>
#include<string.h>
int main()
{
    char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
    int i,j=0,k=0,l=0,pos;
    printf("Enter Production : A->");
    gets(gram);
    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[j];
    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("\nGrammar Without Left Factoring : : \n");
    printf(" A->%s",modifiedGram);
    printf("\n X->%s\n",newGram);
}
```

8)iii) Write a C program to find the first and follow of any given grammar.

sarnabh's code

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


void followfirst(char, int, int);
void follow(char c);
void findfirst(char, int, int);
```

```
int count, n = 0;
char calc_first[10][100];
char calc_follow[10][100];
int m = 0;
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;

int main(int argc, char** argv)
{
      int jm = 0;
      int km = 0;
      int i, choice;
      char c, ch;
      printf("Enter the number of productions (up to 10): ");
  scanf("%d", &count);
  printf("Enter the productions:\n");
  for (i = 0; i < count; i++) {
    printf("Production %d: ", i + 1);
    scanf("%s", production[i]);
  }
  int kay;
      char done[count];
      int ptr = -1;
      for (k = 0; k < count; k++) {
            for (kay = 0; kay < 100; kay++) {
                  calc_first[k][kay] = '!';
            }
      }
      int point1 = 0, point2, xxx;
  for (k = 0; k < count; k++) {
            c = production[k][0];
            point2 = 0;
            xxx = 0;
            for (kay = 0; kay <= ptr; kay++)
                  if (c == done[kay])
                        xxx = 1;

            if (xxx == 1)
                  continue;
            findfirst(c, 0, 0);
```

```
        ptr += 1;
        done[ptr] = c;
        printf("\n First(%c) = { ", c);
        calc_first[point1][point2++] = c;
        for (i = 0 + jm; i < n; i++) {
            int lark = 0, chk = 0;

            for (lark = 0; lark < point2; lark++) {

                if (first[i] == calc_first[point1][lark]) {
                    chk = 1;
                    break;
                }
            }
            if (chk == 0) {
                printf("%c, ", first[i]);
                calc_first[point1][point2++] = first[i];
            }
        }
        printf("}\n");
        jm = n;
        point1++;
    }
    printf("\n");
    printf("-----------------------------------------------"
        "\n\n");
    char donee[count];
    ptr = -1;
    for (k = 0; k < count; k++) {
        for (kay = 0; kay < 100; kay++) {
            calc_follow[k][kay] = '!';
        }
    }
    point1 = 0;
    int land = 0;
    for (e = 0; e < count; e++) {
        ck = production[e][0];
        point2 = 0;
        xxx = 0;
        for (kay = 0; kay <= ptr; kay++)
            if (ck == donee[kay])
                xxx = 1;

        if (xxx == 1)
```

```
                        continue;
                land += 1;
                follow(ck);
                ptr += 1;
                donee[ptr] = ck;
                printf(" Follow(%c) = { ", ck);
                calc_follow[point1][point2++] = ck;
                for (i = 0 + km; i < m; i++) {
                        int lark = 0, chk = 0;
                        for (lark = 0; lark < point2; lark++) {
                                if (f[i] == calc_follow[point1][lark]) {
                                        chk = 1;
                                        break;
                                }
                        }
                        if (chk == 0) {
                                printf("%c, ", f[i]);
                                calc_follow[point1][point2++] = f[i];
                        }
                }
                printf(" }\n\n");
                km = m;
                point1++;
        }
}

void follow(char c)
{
        int i, j;
        if (production[0][0] == c) {
                f[m++] = '$';
        }
        for (i = 0; i < 10; i++) {
                for (j = 2; j < 10; j++) {
                        if (production[i][j] == c) {
                                if (production[i][j + 1] != '\0') {
                                        followfirst(production[i][j + 1], i,
                                                                (j + 2));
                                }

                                if (production[i][j + 1] == '\0'
                                        && c != production[i][0]) {
                                        follow(production[i][0]);
                                }
```

```
                    }
                }
            }
        }

        void findfirst(char c, int q1, int q2)
        {
            int j;
            if (!(isupper(c))) {
                first[n++] = c;
            }
            for (j = 0; j < count; j++) {
                if (production[j][0] == c) {
                    if (production[j][2] == '#') {
                        if (production[q1][q2] == '\0')
                            first[n++] = '#';
                        else if (production[q1][q2] != '\0'
                                && (q1 != 0 || q2 != 0)) {
                            findfirst(production[q1][q2], q1,
                                    (q2 + 1));
                        }
                        else
                            first[n++] = '#';
                    }
                    else if (!isupper(production[j][2])) {
                        first[n++] = production[j][2];
                    }
                    else {
                        findfirst(production[j][2], j, 3);
                    }
                }
            }
        }

        void followfirst(char c, int c1, int c2)
        {
            int k;
            if (!(isupper(c)))
                f[m++] = c;
            else {
                int i = 0, j = 1;
                for (i = 0; i < count; i++) {
                    if (calc_first[i][0] == c)
                        break;
                }
                while (calc_first[i][j] != '!') {
                    if (calc_first[i][j] != '#') {
                        f[m++] = calc_first[i][j];
                    }
                    else {
                        if (production[c1][c2] == '\0') {
                            follow(production[c1][0]);
                        }
                        else {
                            followfirst(production[c1][c2], c1,
                                            c2 + 1);
                        }
                    }
                    j++;
                }
            }
        }
```

<u>my code</u>

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

void followfirst(char , int , int);
void findfirst(char , int , int);
void follow(char c);

int count,n=0;
char calc_first[10][100];
char calc_follow[10][100];
int m=0;
char production[10][10], first[10];
char f[10];
int k;
char ck;
int e;

int main(int argc,char **argv)
{
    int jm=0;
```

```
	int km=0;
	int i,choice;
	char c,ch;
	printf("How many productions ? :");
	scanf("%d",&count);
	printf("\nEnter %d productions in form A=B where A and B are grammar symbols
:\n\n",count);
	for(i=0;i<count;i++)
	{
		scanf("%s%c",production[i],&ch);
	}
	int kay;
	char done[count];
	int ptr = -1;
	for(k=0;k<count;k++){
		for(kay=0;kay<100;kay++){
			calc_first[k][kay] = '!';
		}
	}
	int point1 = 0,point2,xxx;
	for(k=0;k<count;k++)
	{
		c=production[k][0];
		point2 = 0;
		xxx = 0;
		for(kay = 0; kay <= ptr; kay++)
			if(c == done[kay])
				xxx = 1;
		if (xxx == 1)
			continue;
		findfirst(c,0,0);
		ptr+=1;
		done[ptr] = c;
		printf("\n First(%c)= { ",c);
		calc_first[point1][point2++] = c;
		for(i=0+jm;i<n;i++){
			int lark = 0,chk = 0;
			for(lark=0;lark<point2;lark++){
				if (first[i] == calc_first[point1][lark]){
					chk = 1;
					break;
				}
			}
			if(chk == 0){
```

```
                printf("%c, ",first[i]);
                calc_first[point1][point2++] = first[i];
            }
        }
        printf("}\n");
        jm=n;
        point1++;
    }
    printf("\n");
    printf("-----------------------------------------------\n\n");
    char donee[count];
    ptr = -1;
    for(k=0;k<count;k++){
        for(kay=0;kay<100;kay++){
            calc_follow[k][kay] = '!';
        }
    }
    point1 = 0;
    int land = 0;
    for(e=0;e<count;e++)
    {
        ck=production[e][0];
        point2 = 0;
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
            if(ck == donee[kay])
                xxx = 1;
        if (xxx == 1)
            continue;
        land += 1;
        follow(ck);
        ptr+=1;
        donee[ptr] = ck;
        printf(" Follow(%c) = { ",ck);
        calc_follow[point1][point2++] = ck;
        for(i=0+km;i<m;i++){
            int lark = 0,chk = 0;
            for(lark=0;lark<point2;lark++){
                if (f[i] == calc_follow[point1][lark]){
                    chk = 1;
                    break;
                }
            }
            if(chk == 0){
```

```
					printf("%c, ",f[i]);
					calc_follow[point1][point2++] = f[i];
				}
			}
			printf(" }\n\n");
			km=m;
			point1++;
		}
		char ter[10];
		for(k=0;k<10;k++){
			ter[k] = '!';
		}
		int ap,vp,sid = 0;
		for(k=0;k<count;k++){
			for(kay=0;kay<count;kay++){
				if(!isupper(production[k][kay]) && production[k][kay]!= '#' &&
production[k][kay] != '=' && production[k][kay] != '\0'){
					vp = 0;
					for(ap = 0;ap < sid; ap++){
						if(production[k][kay] == ter[ap]){
							vp = 1;
							break;
						}
					}
					if(vp == 0){
						ter[sid] = production[k][kay];
						sid ++;
					}
				}
			}
		}
		ter[sid] = '$';
		sid++;
		char first_prod[count][sid];
		for(ap=0;ap<count;ap++){
			int destiny = 0;
			k = 2;
			int ct = 0;
			char tem[100];
			while(production[ap][k] != '\0'){
				if(!isupper(production[ap][k])){
					tem[ct++] = production[ap][k];
					tem[ct++] = '_';
					tem[ct++] = '\0';
```

```
            k++;
            break;
        }
        else{
            int zap=0;
            int tuna = 0;
            for(zap=0;zap<count;zap++){
                if(calc_first[zap][0] == production[ap][k]){
                    for(tuna=1;tuna<100;tuna++){
                        if(calc_first[zap][tuna] != '!'){
                            tem[ct++] = calc_first[zap][tuna];
                        }
                        else
                            break;
                    }
                break;
                }
            }
            tem[ct++] = '_';
        }
        k++;
    }
    int zap = 0,tuna;
    for(tuna = 0;tuna<ct;tuna++){
        if(tem[tuna] == '#'){
            zap = 1;
        }
        else if(tem[tuna] == '_'){
            if(zap == 1){
                zap = 0;
            }
            else
                break;
        }
        else{
            first_prod[ap][destiny++] = tem[tuna];
        }
    }
}
char table[land][sid+1];
ptr = -1;
for(ap = 0; ap < land ; ap++){
    for(kay = 0; kay < (sid + 1) ; kay++){
        table[ap][kay] = '!';
```

```
		}
	}
	for(ap = 0; ap < count ; ap++){
		ck = production[ap][0];
		xxx = 0;
		for(kay = 0; kay <= ptr; kay++)
			if(ck == table[kay][0])
				xxx = 1;
		if (xxx == 1)
			continue;
		else{
			ptr = ptr + 1;
			table[ptr][0] = ck;
		}
	}
	for(ap = 0; ap < count ; ap++){
		int tuna = 0;
		while(first_prod[ap][tuna] != '\0'){
			int to,ni=0;
			for(to=0;to<sid;to++){
				if(first_prod[ap][tuna] == ter[to]){
					ni = 1;
				}
			}
			if(ni == 1){
				char xz = production[ap][0];
				int cz=0;
				while(table[cz][0] != xz){
					cz = cz + 1;
				}
				int vz=0;
				while(ter[vz] != first_prod[ap][tuna]){
					vz = vz + 1;
				}
				table[cz][vz+1] = (char)(ap + 65);
			}
			tuna++;
		}
	}
	for(k=0;k<sid;k++){
		for(kay=0;kay<100;kay++){
			if(calc_first[k][kay] == '!'){
				break;
			}
```

```
                else if(calc_first[k][kay] == '#'){
                        int fz = 1;
                        while(calc_follow[k][fz] != '!'){
                                char xz = production[k][0];
                                int cz=0;
                                while(table[cz][0] != xz){
                                        cz = cz + 1;
                                }
                                int vz=0;
                                while(ter[vz] != calc_follow[k][fz]){
                                        vz = vz + 1;
                                }
                                table[k][vz+1] = '#';
                                fz++;
                        }
                        break;
                }
        }
}
for(ap = 0; ap < land ; ap++){
        printf("");
        for(kay = 1; kay < (sid + 1) ; kay++){
                if(table[ap][kay] == '!')
                        printf("");
                else if(table[ap][kay] == '#')
                        printf("");
                else{
                        int mum = (int)(table[ap][kay]);
                        mum -= 65;
                        printf("");
                }
        }
}
int j;
char input[100];
scanf("%s%c",input,&ch);
int i_ptr = 0,s_ptr = 1;
char stack[100];
stack[0] = '$';
stack[1] = table[0][0];
while(s_ptr != -1){
        printf("\t\t\t\t\t\t");
        int vamp = 0;
        for(vamp=0;vamp<=s_ptr;vamp++){
```

```
        printf("%c",stack[vamp]);
    }
    printf("\t\t\t");
    vamp = i_ptr;
    while(input[vamp] != '\0'){
        printf("%c",input[vamp]);
        vamp++;
    }
    printf("\t\t\t");
    char her = input[i_ptr];
    char him = stack[s_ptr];
    s_ptr--;
    if(!isupper(him)){
        if(her == him){
            i_ptr++;
        }
        else{
            exit(0);
        }
    }
    else{
        for(i=0;i<sid;i++){
            if(ter[i] == her)
                break;
        }
        char produ[100];
        for(j=0;j<land;j++){
            if(him == table[j][0]){
                if (table[j][i+1] == '#'){
                    printf("%c=#\n",table[j][0]);
                    produ[0] = '#';
                    produ[1] = '\0';
                }
                else if(table[j][i+1] != '!'){
                    int mum = (int)(table[j][i+1]);
                    mum -= 65;
                    strcpy(produ,production[mum]);
                    printf("%s\n",produ);
                }
                else{
                    exit(0);
                }
            }
        }
```

```
                        int le = strlen(produ);
                        le = le - 1;
                        if(le == 0){
                                continue;
                        }
                        for(j=le;j>=2;j--){
                                s_ptr++;
                                stack[s_ptr] = produ[j];
                        }
                }
        }
}

void follow(char c)
{
        int i ,j;
        if(production[0][0]==c){
                f[m++]='$';
        }
        for(i=0;i<10;i++)
        {
                for(j=2;j<10;j++)
                {
                        if(production[i][j]==c)
                        {
                        if(production[i][j+1]!='\0'){
                                        followfirst(production[i][j+1],i,(j+2));
                                }
                        if(production[i][j+1]=='\0'&&c!=production[i][0]){
                                follow(production[i][0]);
                                }
                        }
                }
        }
}

void findfirst(char c ,int q1 , int q2)
{
        int j;
        if(!(isupper(c))){
                first[n++]=c;
        }
        for(j=0;j<count;j++)
        {
```

```
                    if(production[j][0]==c)
                    {
                              if(production[j][2]=='#'){
                                        if(production[q1][q2] == '\0')
                                                  first[n++]='#';
                                        else if(production[q1][q2] != '\0' && (q1 != 0 || q2 != 0))
                                        {
                                                  findfirst(production[q1][q2], q1, (q2+1));
                                        }
                                        else
                                                  first[n++]='#';
                              }
                              else if(!isupper(production[j][2])){
                                        first[n++]=production[j][2];
                              }
                              else {
                                        findfirst(production[j][2], j, 3);
                              }
                    }
          }
}

void followfirst(char c, int c1 , int c2)
{
   int k;
   if(!(isupper(c)))
                    f[m++]=c;
          else{
                    int i=0,j=1;
                    for(i=0;i<count;i++)
                    {
                              if(calc_first[i][0] == c)
                                        break;
                    }
                    while(calc_first[i][j] != '!')
                    {
                              if(calc_first[i][j] != '#'){
                                        f[m++] = calc_first[i][j];
                              }
                              else{
                                        if(production[c1][c2] == '\0'){
                                                  follow(production[c1][0]);
                                        }
                                        else{
```

```
                                        followfirst(production[c1][c2],c1,c2+1);
                                    }
                            }
                            j++;
                    }
            }
}
```

9. Write a C program to check whether an LL(1) parser will accept a given grammar or not.

#### Sambit ka Code (sort of working) ig

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

#define MAX_RULES 100
#define MAX_STRING_LENGTH 100

// Structure to represent a production rule
typedef struct {
    char nonTerminal;
    char *production;

} Rule;



// Function prototypes
int isLL1ParserAcceptable(Rule rules[], int numRules, char *inputString);
int isFirst(char nonTerminal, char terminal, Rule rules[], int numRules);



int main() {
    Rule rules[MAX_RULES];
    int numRules;
    char inputString[MAX_STRING_LENGTH];

    // Input the number of rules
    printf("Enter the number of production rules: ");
    scanf("%d", &numRules);
    getchar(); // consume the newline character

    // Input the production rules
    printf("Enter the production rules in the format 'NonTerminal=Production':\n");
    for (int i = 0; i < numRules; ++i) {
        char input[MAX_STRING_LENGTH];
        fgets(input, sizeof(input), stdin);
        sscanf(input, "%c=%s", &rules[i].nonTerminal, inputString); // Replace "#" with '\0' to
represent epsilon
        if (strcmp(inputString, "#") == 0) {
            rules[i].production = strdup("\0");
        } else {
            rules[i].production = strdup(inputString);
        }
    }

    // Input the string to parse
    printf("Enter the string you want to parse: ");
    fgets(inputString, sizeof(inputString), stdin);
    inputString[strcspn(inputString, "\n")] = '\0'; // remove newline character

    // Check if the LL(1) parser will accept the given grammar
    if (isLL1ParserAcceptable(rules, numRules, inputString)) {
        printf("The LL(1) parser will accept the given grammar for the input string.\n");
    } else {
        printf("The LL(1) parser will not accept the given grammar for the input string.\n");
    }

    // Free memory allocated for production rules
    for (int i = 0; i < numRules; ++i) {
        free(rules[i].production);
    }

    return 0;
}


// Function to check if an LL(1) parser will accept the given grammar for the input string
```

```
int isLL1ParserAcceptable(Rule rules[], int numRules, char *inputString) {
    for (int i = 0; i < numRules; ++i) {
        if (isFirst(rules[i].nonTerminal, inputString[0], rules, numRules)) {
            return 1; // Grammar acceptable
        }
    }
    return 0; // Grammar not acceptable
}


// Function to check if the given non-terminal derives the given terminal as its first symbol

int isFirst(char nonTerminal, char terminal, Rule rules[], int numRules) {
    for (int i = 0; i < numRules; ++i) {
        if (rules[i].nonTerminal == nonTerminal) {
            if (rules[i].production[0] == terminal || (rules[i].production[0] != nonTerminal &&
isFirst(rules[i].production[0], terminal, rules, numRules))) {
                return 1;
            }
        }
    }
    return 0;
}
```

<br>
<br>

#### mera wala

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

void followfirst(char , int , int);
void findfirst(char , int , int);
void follow(char c);

int count,n=0;
char calc_first[10][100];
char calc_follow[10][100];
int m=0;
```

```
char production[10][10], first[10];
char f[10];
int k;
char ck;
int e;

int main(int argc,char **argv)
{
        int jm=0;
        int km=0;
        int i,choice;
        char c,ch;
        printf("How many productions ? :");
        scanf("%d",&count);
        printf("\nEnter %d productions in form A=B where A and B are grammar symbols :\n\n",
count);
        for(i=0;i<count;i++)
        {
                scanf("%s%c",production[i],&ch);
        }
        int kay;
        char done[count];
        int ptr = -1;
        for(k=0;k<count;k++){
                for(kay=0;kay<100;kay++){
                        calc_first[k][kay] = '!';
                }
        }
        int point1 = 0,point2,xxx;
        for(k=0;k<count;k++)
        {
                c=production[k][0];
                point2 = 0;
                xxx = 0;
                for(kay = 0; kay <= ptr; kay++)
                        if(c == done[kay])
                                xxx = 1;
                if (xxx == 1)
                        continue;
                findfirst(c,0,0);
                ptr+=1;
                done[ptr] = c;
                // printf("\n First(%c)= { ",c);
                calc_first[point1][point2++] = c;
```

```
		for(i=0+jm;i<n;i++){
			int lark = 0,chk = 0;
			for(lark=0;lark<point2;lark++){
				if (first[i] == calc_first[point1][lark]){
					chk = 1;
					break;
				}
			}
			if(chk == 0){
				// printf("%c, ",first[i]);
				calc_first[point1][point2++] = first[i];
			}
		}
		// printf("}\n");
		jm=n;
		point1++;
	}
	// printf("-----------------------------------------------\n\n");
	char donee[count];
	ptr = -1;
	for(k=0;k<count;k++){
		for(kay=0;kay<100;kay++){
			calc_follow[k][kay] = '!';
		}
	}
	point1 = 0;
	int land = 0;
	for(e=0;e<count;e++)
	{
		ck=production[e][0];
		point2 = 0;
		xxx = 0;
		for(kay = 0; kay <= ptr; kay++)
			if(ck == donee[kay])
				xxx = 1;
		if (xxx == 1)
			continue;
		land += 1;
		follow(ck);
		ptr+=1;
		donee[ptr] = ck;
		// printf(" Follow(%c) = { ",ck);
		calc_follow[point1][point2++] = ck;
		for(i=0+km;i<m;i++){
			int lark = 0,chk = 0;
			for(lark=0;lark<point2;lark++){
				if (f[i] == calc_follow[point1][lark]){
					chk = 1;
					break;
				}
			}
			if(chk == 0){
				// printf("%c, ", f[i]);
				calc_follow[point1][point2++] = f[i];
			}
		}
		// printf(" }\n\n");
		km=m;
		point1++;
	}
	char ter[10];
	for(k=0;k<10;k++){
		ter[k] = '!';
	}
	int ap,vp,sid = 0;
	for(k=0;k<count;k++){
		for(kay=0;kay<count;kay++){
			if(!isupper(production[k][kay]) && production[k][kay]!= '#' &&
production[k][kay] != '=' && production[k][kay] != '\0'){
				vp = 0;
				for(ap = 0;ap < sid; ap++){
					if(production[k][kay] == ter[ap]){
						vp = 1;
						break;
					}
				}
				if(vp == 0){
					ter[sid] = production[k][kay];
					sid ++;
				}
			}
		}
	}
	ter[sid] = '$';
	sid++;
	// printf("\n\t\t\t\t\t\t\t The LL(1) Parsing Table for the above grammer :-");
	// printf("\n\t\t\t\t\t\t\t^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n");
```

```
        //
printf("\n\t\t\t====================================================================
==========================================================\n");
        // printf("\t\t\t\t|\t");
        // for(ap = 0;ap < sid; ap++){
        //        printf("");
        // }
        //
printf("\n\t\t\t====================================================================
==========================================================\n");
        char first_prod[count][sid];
        for(ap=0;ap<count;ap++){
                int destiny = 0;
                k = 2;
                int ct = 0;
                char tem[100];
                while(production[ap][k] != '\0'){
                        if(!isupper(production[ap][k])){
                                tem[ct++] = production[ap][k];
                                tem[ct++] = '_';
                                tem[ct++] = '\0';
                                k++;
                                break;
                        }
                        else{
                                int zap=0;
                                int tuna = 0;
                                for(zap=0;zap<count;zap++){
                                        if(calc_first[zap][0] == production[ap][k]){
                                                for(tuna=1;tuna<100;tuna++){
                                                        if(calc_first[zap][tuna] != '!'){
                                                                tem[ct++] = calc_first[zap][tuna];
                                                        }
                                                        else
                                                                break;
                                                }
                                        break;
                                        }
                                }
                                tem[ct++] = '_';
                        }
                        k++;
                }
                int zap = 0,tuna;
        for(tuna = 0;tuna<ct;tuna++){
            if(tem[tuna] == '#'){
                zap = 1;
            }
            else if(tem[tuna] == '_'){
                if(zap == 1){
                    zap = 0;
                }
                else
                    break;
            }
            else{
                first_prod[ap][destiny++] = tem[tuna];
            }
        }
    }
    char table[land][sid+1];
    ptr = -1;
    for(ap = 0; ap < land ; ap++){
        for(kay = 0; kay < (sid + 1) ; kay++){
            table[ap][kay] = '!';
        }
    }
    for(ap = 0; ap < count ; ap++){
        ck = production[ap][0];
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
            if(ck == table[kay][0])
                xxx = 1;
        if (xxx == 1)
            continue;
        else{
            ptr = ptr + 1;
            table[ptr][0] = ck;
        }
    }
    for(ap = 0; ap < count ; ap++){
        int tuna = 0;
        while(first_prod[ap][tuna] != '\0'){
            int to,ni=0;
            for(to=0;to<sid;to++){
                if(first_prod[ap][tuna] == ter[to]){
                    ni = 1;
                }
            }
            if(ni == 1){
                char xz = production[ap][0];
                int cz=0;
                while(table[cz][0] != xz){
                    cz = cz + 1;
                }
                int vz=0;
                while(ter[vz] != first_prod[ap][tuna]){
                    vz = vz + 1;
                }
                table[cz][vz+1] = (char)(ap + 65);
            }
            tuna++;
        }
    }
    for(k=0;k<sid;k++){
        for(kay=0;kay<100;kay++){
            if(calc_first[k][kay] == '!'){
                break;
            }
            else if(calc_first[k][kay] == '#'){
                int fz = 1;
                while(calc_follow[k][fz] != '!'){
                    char xz = production[k][0];
                    int cz=0;
                    while(table[cz][0] != xz){
                        cz = cz + 1;
                    }
                    int vz=0;
                    while(ter[vz] != calc_follow[k][fz]){
                        vz = vz + 1;
                    }
                    table[k][vz+1] = '#';
                    fz++;
                }
                break;
            }
        }
    }
    for(ap = 0; ap < land ; ap++){
        // printf("\t\t\t   %c\t|\t",table[ap][0]);
        for(kay = 1; kay < (sid + 1) ; kay++){
            if(table[ap][kay] == '!')
    				printf("");
    			else if(table[ap][kay] == '#')
    				printf("");
    			else{
    				int mum = (int)(table[ap][kay]);
    				mum -= 65;
    				// printf("%s\t\t",production[mum]);
    			}
    		}
    		//
    printf("\t\t\t------------------------------------------------------------------------------------------------------------------------
    ---");
    		// printf("\n");
    	}
    	int j;
    	printf("\n\nPlease enter the desired INPUT STRING (make sure to include a '$' symbol
    after the string) (e.g. abbb$) = ");
    	char input[100];
    	scanf("%s%c",input,&ch);
    	//
    printf("\n\t\t\t\t\t========================================================================
    ==============\n");
    	// printf("\t\t\t\t\t\tStack\t\t\tInput\t\t\tAction");
    	//
    printf("\n\t\t\t\t\t========================================================================
    ==============\n");
    	int i_ptr = 0,s_ptr = 1;
    	char stack[100];
    	stack[0] = '$';
    	stack[1] = table[0][0];
    	while(s_ptr != -1){
    		// printf("\t\t\t\t\t\t");
    		int vamp = 0;
    		for(vamp=0;vamp<=s_ptr;vamp++){
    			// printf("%c",stack[vamp]);
    		}
    		// printf("\t\t\t");
    		vamp = i_ptr;
    		while(input[vamp] != '\0'){
    			// printf("%c",input[vamp]);
    			vamp++;
    		}
    		// printf("\t\t\t");
    		char her = input[i_ptr];
    char him = stack[s_ptr];
    s_ptr--;
    if(!isupper(him)){
            if(her == him){
                    i_ptr++;
                    // printf("POP ACTION\n");
            }
            else{
                    printf("\nString Not Accepted by LL(1) Parser !!\n");
                    exit(0);
            }
    }
    else{
            for(i=0;i<sid;i++){
                    if(ter[i] == her)
                            break;
            }
            char produ[100];
            for(j=0;j<land;j++){
                    if(him == table[j][0]){
                            if (table[j][i+1] == '#'){
                                    // printf("%c=#\n",table[j][0]);
                                    produ[0] = '#';
                                    produ[1] = '\0';
                            }
                            else if(table[j][i+1] != '!'){
                                    int mum = (int)(table[j][i+1]);
                                    mum -= 65;
                                    strcpy(produ,production[mum]);
                                    // printf("%s\n",produ);
                            }
                            else{
                                    printf("\nString Not Accepted by LL(1) Parser !!\n");
                                    exit(0);
                            }
                    }
            }
            int le = strlen(produ);
            le = le - 1;
            if(le == 0){
                    continue;
            }
            for(j=le;j>=2;j--){
                    s_ptr++;
```

```
                                        stack[s_ptr] = produ[j];
                              }
                    }
          }
          //
printf("\n\t\t\t=================================================
===========================================\n");
          if (input[i_ptr] == '\0'){
                    printf("\nYOUR STRING HAS BEEN ACCEPTED !!\n");
          }
          else{
       printf("\nYOUR STRING HAS BEEN REJECTED !!\n");
   }
}

void follow(char c)
{
          int i ,j;
          if(production[0][0]==c){
                    f[m++]='$';
          }
          for(i=0;i<10;i++)
          {
                    for(j=2;j<10;j++)
                    {
                              if(production[i][j]==c)
                              {
                              if(production[i][j+1]!='\0'){
                                                  followfirst(production[i][j+1],i,(j+2));
                                        }
                              if(production[i][j+1]=='\0'&&c!=production[i][0]){
                                        follow(production[i][0]);
                                        }
                              }
                    }
          }
}

void findfirst(char c ,int q1 , int q2)
{
          int j;
          if(!(isupper(c))){
                    first[n++]=c;
          }
```

```
        for(j=0;j<count;j++)
        {
                if(production[j][0]==c)
                {
                        if(production[j][2]=='#'){
                                if(production[q1][q2] == '\0')
                                        first[n++]='#';
                                else if(production[q1][q2] != '\0' && (q1 != 0 || q2 != 0))
                                {
                                        findfirst(production[q1][q2], q1, (q2+1));
                                }
                                else
                                        first[n++]='#';
                        }
                        else if(!isupper(production[j][2])){
                                first[n++]=production[j][2];
                        }
                        else {
                                findfirst(production[j][2], j, 3);
                        }
                }
        }
}

void followfirst(char c, int c1 , int c2)
{
   int k;
   if(!(isupper(c)))
                f[m++]=c;
        else{
                int i=0,j=1;
                for(i=0;i<count;i++)
                {
                        if(calc_first[i][0] == c)
                                break;
                }
                while(calc_first[i][j] != '!')
                {
                        if(calc_first[i][j] != '#'){
                                f[m++] = calc_first[i][j];
                        }
                        else{
                                if(production[c1][c2] == '\0'){
                                        follow(production[c1][0]);
                }
                else{
                    followfirst(production[c1][c2],c1,c2+1);
                }
            }
            j++;
        }
    }
}
```

10. Write a C program to implement LL(1) parser using stack.

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

void followfirst(char , int , int);
void findfirst(char , int , int);
void follow(char c);

int count,n=0;
char calc_first[10][100];
char calc_follow[10][100];
int m=0;
char production[10][10], first[10];
char f[10];
int k;
char ck;
int e;

int main(int argc,char **argv)
{
    int jm=0;
    int km=0;
    int i,choice;
    char c,ch;
    printf("How many productions ? :");
    scanf("%d",&count);
    printf("\nEnter %d productions in form A=B where A and B are grammar symbols
:\n\n",count);
```

```
for(i=0;i<count;i++)
{
	scanf("%s%c",production[i],&ch);
}
int kay;
char done[count];
int ptr = -1;
for(k=0;k<count;k++){
	for(kay=0;kay<100;kay++){
		calc_first[k][kay] = '!';
	}
}
int point1 = 0,point2,xxx;
for(k=0;k<count;k++)
{
	c=production[k][0];
	point2 = 0;
	xxx = 0;
	for(kay = 0; kay <= ptr; kay++)
		if(c == done[kay])
			xxx = 1;
	if (xxx == 1)
		continue;
	findfirst(c,0,0);
	ptr+=1;
	done[ptr] = c;
	calc_first[point1][point2++] = c;
	for(i=0+jm;i<n;i++){
		int lark = 0,chk = 0;
		for(lark=0;lark<point2;lark++){
			if (first[i] == calc_first[point1][lark]){
				chk = 1;
				break;
			}
		}
		if(chk == 0){
			calc_first[point1][point2++] = first[i];
		}
	}
	jm=n;
	point1++;
}
char donee[count];
ptr = -1;
```

```
for(k=0;k<count;k++){
        for(kay=0;kay<100;kay++){
                calc_follow[k][kay] = '!';
        }
}
point1 = 0;
int land = 0;
for(e=0;e<count;e++)
{
        ck=production[e][0];
        point2 = 0;
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
                if(ck == donee[kay])
                        xxx = 1;
        if (xxx == 1)
                continue;
        land += 1;
        follow(ck);
        ptr+=1;
        donee[ptr] = ck;
        calc_follow[point1][point2++] = ck;
        for(i=0+km;i<m;i++){
                int lark = 0,chk = 0;
                for(lark=0;lark<point2;lark++){
                        if (f[i] == calc_follow[point1][lark]){
                                chk = 1;
                                break;
                        }
                }
                if(chk == 0){
                        calc_follow[point1][point2++] = f[i];
                }
        }
        km=m;
        point1++;
}
char ter[10];
for(k=0;k<10;k++){
        ter[k] = '!';
}
int ap,vp,sid = 0;
for(k=0;k<count;k++){
        for(kay=0;kay<count;kay++){
```

```
                    if(!isupper(production[k][kay]) && production[k][kay]!= '#' &&
production[k][kay] != '=' && production[k][kay] != '\0'){
                        vp = 0;
                        for(ap = 0;ap < sid; ap++){
                            if(production[k][kay] == ter[ap]){
                                vp = 1;
                                break;
                            }
                        }
                        if(vp == 0){
                            ter[sid] = production[k][kay];
                            sid ++;
                        }
                    }
                }
            }
            ter[sid] = '$';
            sid++;
            char first_prod[count][sid];
            for(ap=0;ap<count;ap++){
                int destiny = 0;
                k = 2;
                int ct = 0;
                char tem[100];
                while(production[ap][k] != '\0'){
                    if(!isupper(production[ap][k])){
                        tem[ct++] = production[ap][k];
                        tem[ct++] = '_';
                        tem[ct++] = '\0';
                        k++;
                        break;
                    }
                    else{
                        int zap=0;
                        int tuna = 0;
                        for(zap=0;zap<count;zap++){
                            if(calc_first[zap][0] == production[ap][k]){
                                for(tuna=1;tuna<100;tuna++){
                                    if(calc_first[zap][tuna] != '!'){
                                        tem[ct++] = calc_first[zap][tuna];
                                    }
                                    else
                                        break;
                                }
```

```
                                        break;
                                        }
                                }
                                tem[ct++] = '_';
                        }
                        k++;
                }
                int zap = 0,tuna;
                for(tuna = 0;tuna<ct;tuna++){
                        if(tem[tuna] == '#'){
                                zap = 1;
                        }
                        else if(tem[tuna] == '_'){
                                if(zap == 1){
                                        zap = 0;
                                }
                                else
                                        break;
                        }
                        else{
                                first_prod[ap][destiny++] = tem[tuna];
                        }
                }
        }
        char table[land][sid+1];
        ptr = -1;
        for(ap = 0; ap < land ; ap++){
                for(kay = 0; kay < (sid + 1) ; kay++){
                        table[ap][kay] = '!';
                }
        }
        for(ap = 0; ap < count ; ap++){
                ck = production[ap][0];
                xxx = 0;
                for(kay = 0; kay <= ptr; kay++)
                        if(ck == table[kay][0])
                                xxx = 1;
                if (xxx == 1)
                        continue;
                else{
                        ptr = ptr + 1;
                        table[ptr][0] = ck;
                }
        }
```

```
for(ap = 0; ap < count ; ap++){
        int tuna = 0;
        while(first_prod[ap][tuna] != '\0'){
                int to,ni=0;
                for(to=0;to<sid;to++){
                        if(first_prod[ap][tuna] == ter[to]){
                                ni = 1;
                        }
                }
                if(ni == 1){
                        char xz = production[ap][0];
                        int cz=0;
                        while(table[cz][0] != xz){
                                cz = cz + 1;
                        }
                        int vz=0;
                        while(ter[vz] != first_prod[ap][tuna]){
                                vz = vz + 1;
                        }
                        table[cz][vz+1] = (char)(ap + 65);
                }
                tuna++;
        }
}
for(k=0;k<sid;k++){
        for(kay=0;kay<100;kay++){
                if(calc_first[k][kay] == '!'){
                        break;
                }
                else if(calc_first[k][kay] == '#'){
                        int fz = 1;
                        while(calc_follow[k][fz] != '!'){
                                char xz = production[k][0];
                                int cz=0;
                                while(table[cz][0] != xz){
                                        cz = cz + 1;
                                }
                                int vz=0;
                                while(ter[vz] != calc_follow[k][fz]){
                                        vz = vz + 1;
                                }
                                table[k][vz+1] = '#';
                                fz++;
                        }
                        break;
                    }
                }
            }
            for(ap = 0; ap < land ; ap++){
                printf("");
                for(kay = 1; kay < (sid + 1) ; kay++){
                    if(table[ap][kay] == '!')
                        printf("");
                    else if(table[ap][kay] == '#')
                        printf("");
                    else{
                        int mum = (int)(table[ap][kay]);
                        mum -= 65;
                        printf("");
                    }
                }
            }
            int j;
            printf("\n\nPlease enter the desired INPUT STRING (make sure to include a '$' symbol
after the string) (e.g. abbb$) = ");
            char input[100];
            scanf("%s%c",input,&ch);

printf("\n\t\t\t\t\t=======================================================================
==============\n");
            printf("\t\t\t\t\t\tStack\t\t\tInput\t\t\tAction");

printf("\n\t\t\t\t\t=======================================================================
==============\n");
            int i_ptr = 0,s_ptr = 1;
            char stack[100];
            stack[0] = '$';
            stack[1] = table[0][0];
            while(s_ptr != -1){
                printf("\t\t\t\t\t\t");
                int vamp = 0;
                for(vamp=0;vamp<=s_ptr;vamp++){
                    printf("%c",stack[vamp]);
                }
                printf("\t\t\t");
                vamp = i_ptr;
                while(input[vamp] != '\0'){
                    printf("%c",input[vamp]);
```

```
		vamp++;
	}
	printf("\t\t\t");
	char her = input[i_ptr];
	char him = stack[s_ptr];
	s_ptr--;
	if(!isupper(him)){
		if(her == him){
			i_ptr++;
			printf("POP ACTION\n");
		}
		else{
			printf("\nString Not Accepted by LL(1) Parser !!\n");
			exit(0);
		}
	}
	else{
		for(i=0;i<sid;i++){
			if(ter[i] == her)
				break;
		}
		char produ[100];
		for(j=0;j<land;j++){
			if(him == table[j][0]){
				if (table[j][i+1] == '#'){
					printf("%c=#\n",table[j][0]);
					produ[0] = '#';
					produ[1] = '\0';
				}
				else if(table[j][i+1] != '!'){
					int mum = (int)(table[j][i+1]);
					mum -= 65;
					strcpy(produ,production[mum]);
					printf("%s\n",produ);
				}
				else{
					printf("\nString Not Accepted by LL(1) Parser !!\n");
					exit(0);
				}
			}
		}
		int le = strlen(produ);
		le = le - 1;
		if(le == 0){
```

```
                                        continue;
                                }
                                for(j=le;j>=2;j--){
                                        s_ptr++;
                                        stack[s_ptr] = produ[j];
                                }
                        }
                }
                if (input[i_ptr] == '\0'){
                        printf("YOUR STRING HAS BEEN ACCEPTED !!\n");
                }
                else
                        printf("\nYOUR STRING HAS BEEN REJECTED !!\n");
        }

        void follow(char c)
        {
                int i ,j;
                if(production[0][0]==c){
                        f[m++]='$';
                }
                for(i=0;i<10;i++)
                {
                        for(j=2;j<10;j++)
                        {
                                if(production[i][j]==c)
                                {
                                if(production[i][j+1]!='\0'){
                                                followfirst(production[i][j+1],i,(j+2));
                                        }
                                if(production[i][j+1]=='\0'&&c!=production[i][0]){
                                        follow(production[i][0]);
                                        }
                                }
                        }
                }
        }

        void findfirst(char c ,int q1 , int q2)
        {
                int j;
                if(!(isupper(c))){
                        first[n++]=c;
                }
                for(j=0;j<count;j++)
                {
                        if(production[j][0]==c)
                        {
                                if(production[j][2]=='#'){
                                        if(production[q1][q2] == '\0')
                                                first[n++]='#';
                                        else if(production[q1][q2] != '\0' && (q1 != 0 || q2 != 0))
                                        {
                                                findfirst(production[q1][q2], q1, (q2+1));
                                        }
                                        else
                                                first[n++]='#';
                                }
                                else if(!isupper(production[j][2])){
                                        first[n++]=production[j][2];
                                }
                                else {
                                        findfirst(production[j][2], j, 3);
                                }
                        }
                }
        }

        void followfirst(char c, int c1 , int c2)
        {
           int k;
           if(!(isupper(c)))
                        f[m++]=c;
                else{
                        int i=0,j=1;
                        for(i=0;i<count;i++)
                        {
                                if(calc_first[i][0] == c)
                                        break;
                        }
                        while(calc_first[i][j] != '!')
                        {
                                if(calc_first[i][j] != '#'){
                                        f[m++] = calc_first[i][j];
                                }
                                else{
                                        if(production[c1][c2] == '\0'){
                                                follow(production[c1][0]);
                        }
                        else{
                            followfirst(production[c1][c2],c1,c2+1);
                        }
                    }
                    j++;
                }
            }
        }
```

11. Write a C program to check whether a given grammar will be accepted by LR(0) parser or not.

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

#define MAX_PROD_LEN 20
#define MAX_PROD_NUM 10
#define MAX_ITEM_SET_SIZE 50

// Structure to represent a production
typedef struct Production {
    char lhs; // Left hand side of the production
    char rhs[MAX_PROD_LEN]; // Right hand side of the production
} Production;

// Structure to represent a LR(0) item
typedef struct Item {
    int prod_index; // Index of the production
    int position;   // Position of the dot
} Item;

// Function prototypes
void closure(Item item, Production* grammar, int grammar_size, bool* visited);
bool is_same_item(Item item1, Item item2);
bool contains_item(Item* item_set, int set_size, Item item);
bool is_accepted_by_lr0(Production* grammar, int grammar_size);
void lr0_parser(Production* grammar, int grammar_size);
```

```
int main() {
    // Define your grammar here
    Production grammar[MAX_PROD_NUM] = {
        {'S', "AB"},
        {'A', "a"},
        {'B', "a"}
    };

    int grammar_size = 3; // Number of productions in the grammar

    if (is_accepted_by_lr0(grammar, grammar_size))
        printf("Grammar is accepted by LR(0) parser.\n");
    else
        printf("Grammar is not accepted by LR(0) parser.\n");

    return 0;
}

// Function to compute the closure of an LR(0) item
void closure(Item item, Production* grammar, int grammar_size, bool* visited) {
    visited[item.prod_index] = true;

    char symbol = grammar[item.prod_index].rhs[item.position];

    if (symbol >= 'A' && symbol <= 'Z') {
        for (int i = 0; i < grammar_size; i++) {
            if (grammar[i].lhs == symbol && !visited[i]) {
                Item new_item = {i, 0};
                closure(new_item, grammar, grammar_size, visited);
            }
        }
    }
}

// Function to check if two LR(0) items are the same
bool is_same_item(Item item1, Item item2) {
    return (item1.prod_index == item2.prod_index) && (item1.position == item2.position);
}

// Function to check if an LR(0) item set contains a specific item
bool contains_item(Item* item_set, int set_size, Item item) {
    for (int i = 0; i < set_size; i++) {
        if (is_same_item(item_set[i], item))
            return true;
    }
    return false;
}

// Function to check if the grammar is accepted by LR(0) parser
bool is_accepted_by_lr0(Production* grammar, int grammar_size) {
    // Initialize the closure of the start production
    bool visited[grammar_size];
    memset(visited, false, sizeof(visited));

    Item start_item = {0, 0};
    closure(start_item, grammar, grammar_size, visited);

    // Check if all productions were visited
    for (int i = 0; i < grammar_size; i++) {
        if (!visited[i])
            return false;
    }

    return true;
}
```

12. Design a lexical analyzer for language given in the manual and the lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Simulate the same in C language.

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

#define MAX_IDENTIFIER_LENGTH 50
#define MAX_TOKEN_LENGTH 100

// Token types
typedef enum {
    IDENTIFIER,
    INTEGER_CONSTANT,
    FLOAT_CONSTANT,
    KEYWORD,
    OPERATOR,
    DELIMITER,
    COMMENT,
    END_OF_FILE // Added enumeration value for EOF
} TokenType;

// Structure to represent a token
typedef struct {
    TokenType type;
    char lexeme[MAX_TOKEN_LENGTH];
} Token;

// Function to check if a character is a valid identifier character
int isValidIdentifierChar(char ch) {
    return isalnum(ch) || ch == '_';
}

// Function to get the next token from the input stream
Token getNextToken(FILE *input) {
    Token token;
    char ch;

    // Skip whitespace and comments
    do {
        ch = fgetc(input);
        if (ch == '/') {
            ch = fgetc(input);
            if (ch == '/') {  // Line comment
                while ((ch = fgetc(input)) != '\n' && ch != EOF) {}
            } else if (ch == '*') {  // Block comment
                char prev_ch = '\0';
                while ((ch = fgetc(input)) != EOF) {
                    if (prev_ch == '*' && ch == '/') {
                        break;
                    }
                    prev_ch = ch;
                }
            } else {
                ungetc(ch, input);
                break;
            }
        }
    } while (isspace(ch));

    // Check for EOF
    if (ch == EOF) {
        token.type = END_OF_FILE;
        return token;
    }

    // Identify token type based on the first character
    if (isalpha(ch) || ch == '_') {
        token.type = IDENTIFIER;
        int i = 0;
        token.lexeme[i++] = ch;
        while ((ch = fgetc(input)) != EOF && isValidIdentifierChar(ch) && i <
MAX_IDENTIFIER_LENGTH - 1) {
            token.lexeme[i++] = ch;
        }
        token.lexeme[i] = '\0';
        ungetc(ch, input);
    } else {
        // Handle other token types here
        // For simplicity, let's assume everything else is an operator or delimiter
        token.type = OPERATOR;
        token.lexeme[0] = ch;
        token.lexeme[1] = '\0';
    }

    return token;
}

// Function to print token information
void printToken(Token token) {
    switch (token.type) {
        case IDENTIFIER:
            printf("Identifier: %s\n", token.lexeme);
            break;
        case OPERATOR:
            printf("Operator or Delimiter: %s\n", token.lexeme);
            break;
        case END_OF_FILE:
            printf("End of File\n");
            break;
        default:
            printf("Unknown Token\n");
            break;
    }
}

int main() {
    FILE *input = fopen("input.txt", "r");
    if (input == NULL) {
        perror("Error opening file");
        return EXIT_FAILURE;
    }

    Token token;
    do {
        token = getNextToken(input);
        printToken(token);
    } while (token.type != END_OF_FILE);

    fclose(input);
    return EXIT_SUCCESS;
}
```
input.txt
```
// Sample input file for the lexical analyzer

int main() {
    int a = 10;
    float b = 3.14;
    if (a == 10) {
        b = b + 1.5;
    } else {
        b = b - 1.5;
    }
    return 0;
}

```

SLR Parser

```
```

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int z = 0, i = 0, j = 0, c = 0;
char a[16], ac[20], stk[15], act[10];
void check() {
        strcpy(ac,"REDUCE TO E -> ");
        for(z = 0; z < c; z++) {
                if(stk[z] == '4')
                {       printf("%s4", ac);
                        stk[z] = 'E';
                        stk[z + 1] = '\0';
                        printf("\n$%s\t%s$\t", stk, a);
                }}
        for(z = 0; z < c - 2; z++) {
                if(stk[z] == '2' && stk[z + 1] == 'E' && stk[z + 2] == '2')
                {       printf("%s2E2", ac);
                        stk[z] = 'E';
                        stk[z + 1] = '\0';
                        stk[z + 2] = '\0';
                        printf("\n$%s\t%s$\t", stk, a);
                        i = i - 2;
                }}
        for(z=0; z<c-2; z++) {
                if(stk[z] == '3' && stk[z + 1] == 'E' && stk[z + 2] == '3')
                {       printf("%s3E3", ac);
                        stk[z]='E';
                        stk[z + 1]='\0';
                        stk[z + 1]='\0';
                        printf("\n$%s\t%s$\t", stk, a);
                        i = i - 2;
                }}
        return ; }
int main() {
        printf("GRAMMAR is -\nS->CC \nC->cC|d\n");
        strcpy(a,"cdcd");
        c=strlen(a);
        strcpy(act,"SHIFT");
        printf("\nstack \t input \t action");
        printf("\n$\t%s$\t", a);
        for(i = 0; j < c; i++, j++) {
                printf("%s", act);
                stk[i] = a[j];
                stk[i + 1] = '\0';
                a[j]=' ';
                printf("\n$%s\t%s$\t", stk, a);
                check(); }

        check();
        if(stk[0] == 'S' && stk[1] == '\0')
                printf("Accept\n");
        else
                printf("Reject\n"); }
```

SOURCE FOR 8.3,9,10,11

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

void followfirst(char , int , int);
void findfirst(char , int , int);
void follow(char c);

int count,n=0;
char calc_first[10][100];
char calc_follow[10][100];
int m=0;
char production[10][10], first[10];
char f[10];
int k;
char ck;
int e;

int main(int argc,char **argv)
{
    int jm=0;
    int km=0;
    int i,choice;
    char c,ch;
    printf("How many productions ? :");
    scanf("%d",&count);
```

```
	printf("\nEnter %d productions in form A=B where A and B are grammar symbols
:\n\n",count);
	for(i=0;i<count;i++)
	{
		scanf("%s%c",production[i],&ch);
	}
	int kay;
	char done[count];
	int ptr = -1;
	for(k=0;k<count;k++){
		for(kay=0;kay<100;kay++){
			calc_first[k][kay] = '!';
		}
	}
	int point1 = 0,point2,xxx;
	for(k=0;k<count;k++)
	{
		c=production[k][0];
		point2 = 0;
		xxx = 0;
		for(kay = 0; kay <= ptr; kay++)
			if(c == done[kay])
				xxx = 1;
		if (xxx == 1)
			continue;
		findfirst(c,0,0);
		ptr+=1;
		done[ptr] = c;
		printf("\n First(%c)= { ",c);
		calc_first[point1][point2++] = c;
		for(i=0+jm;i<n;i++){
			int lark = 0,chk = 0;
			for(lark=0;lark<point2;lark++){
				if (first[i] == calc_first[point1][lark]){
					chk = 1;
					break;
				}
			}
			if(chk == 0){
				printf("%c, ",first[i]);
				calc_first[point1][point2++] = first[i];
			}
		}
		printf("}\n");
```

```
        jm=n;
        point1++;
}
printf("\n");
printf("-----------------------------------------------\n\n");
char donee[count];
ptr = -1;
for(k=0;k<count;k++){
        for(kay=0;kay<100;kay++){
                calc_follow[k][kay] = '!';
        }
}
point1 = 0;
int land = 0;
for(e=0;e<count;e++)
{
        ck=production[e][0];
        point2 = 0;
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
                if(ck == donee[kay])
                        xxx = 1;
        if (xxx == 1)
                continue;
        land += 1;
        follow(ck);
        ptr+=1;
        donee[ptr] = ck;
        printf(" Follow(%c) = { ",ck);
        calc_follow[point1][point2++] = ck;
        for(i=0+km;i<m;i++){
                int lark = 0,chk = 0;
                for(lark=0;lark<point2;lark++){
                        if (f[i] == calc_follow[point1][lark]){
                                chk = 1;
                                break;
                        }
                }
                if(chk == 0){
                        printf("%c, ",f[i]);
                        calc_follow[point1][point2++] = f[i];
                }
        }
        printf(" }\n\n");
        km=m;
        point1++;
    }
    char ter[10];
    for(k=0;k<10;k++){
        ter[k] = '!';
    }
    int ap,vp,sid = 0;
    for(k=0;k<count;k++){
        for(kay=0;kay<count;kay++){
            if(!isupper(production[k][kay]) && production[k][kay]!= '#' &&
production[k][kay] != '=' && production[k][kay] != '\0'){
                vp = 0;
                for(ap = 0;ap < sid; ap++){
                    if(production[k][kay] == ter[ap]){
                        vp = 1;
                        break;
                    }
                }
                if(vp == 0){
                    ter[sid] = production[k][kay];
                    sid ++;
                }
            }
        }
    }
    ter[sid] = '$';
    sid++;
    printf("\n\t\t\t\t\t\t\t The LL(1) Parsing Table for the above grammer :-");
    printf("\n\t\t\t\t\t\t\t^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n");

printf("\n\t\t\t==================================================================
=========================================================\n");
    printf("\t\t\t\t|\t");
    for(ap = 0;ap < sid; ap++){
        printf("%c\t\t",ter[ap]);
    }

printf("\n\t\t\t==================================================================
=========================================================\n");
    char first_prod[count][sid];
    for(ap=0;ap<count;ap++){
        int destiny = 0;
        k = 2;
```

```
int ct = 0;
char tem[100];
while(production[ap][k] != '\0'){
        if(!isupper(production[ap][k])){
                tem[ct++] = production[ap][k];
                tem[ct++] = '_';
                tem[ct++] = '\0';
                k++;
                break;
        }
        else{
                int zap=0;
                int tuna = 0;
                for(zap=0;zap<count;zap++){
                        if(calc_first[zap][0] == production[ap][k]){
                                for(tuna=1;tuna<100;tuna++){
                                        if(calc_first[zap][tuna] != '!'){
                                                tem[ct++] = calc_first[zap][tuna];
                                        }
                                        else
                                                break;
                                }
                        break;
                        }
                }
                tem[ct++] = '_';
        }
        k++;
}
int zap = 0,tuna;
for(tuna = 0;tuna<ct;tuna++){
        if(tem[tuna] == '#'){
                zap = 1;
        }
        else if(tem[tuna] == '_'){
                if(zap == 1){
                        zap = 0;
                }
                else
                        break;
        }
        else{
                first_prod[ap][destiny++] = tem[tuna];
        }
        }
}
char table[land][sid+1];
ptr = -1;
for(ap = 0; ap < land ; ap++){
        for(kay = 0; kay < (sid + 1) ; kay++){
                table[ap][kay] = '!';
        }
}
for(ap = 0; ap < count ; ap++){
        ck = production[ap][0];
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
                if(ck == table[kay][0])
                        xxx = 1;
        if (xxx == 1)
                continue;
        else{
                ptr = ptr + 1;
                table[ptr][0] = ck;
        }
}
for(ap = 0; ap < count ; ap++){
        int tuna = 0;
        while(first_prod[ap][tuna] != '\0'){
                int to,ni=0;
                for(to=0;to<sid;to++){
                        if(first_prod[ap][tuna] == ter[to]){
                                ni = 1;
                        }
                }
                if(ni == 1){
                        char xz = production[ap][0];
                        int cz=0;
                        while(table[cz][0] != xz){
                                cz = cz + 1;
                        }
                        int vz=0;
                        while(ter[vz] != first_prod[ap][tuna]){
                                vz = vz + 1;
                        }
                        table[cz][vz+1] = (char)(ap + 65);
                }
                tuna++;
                }
        }
        for(k=0;k<sid;k++){
                for(kay=0;kay<100;kay++){
                        if(calc_first[k][kay] == '!'){
                                break;
                        }
                        else if(calc_first[k][kay] == '#'){
                                int fz = 1;
                                while(calc_follow[k][fz] != '!'){
                                        char xz = production[k][0];
                                        int cz=0;
                                        while(table[cz][0] != xz){
                                                cz = cz + 1;
                                        }
                                        int vz=0;
                                        while(ter[vz] != calc_follow[k][fz]){
                                                vz = vz + 1;
                                        }
                                        table[k][vz+1] = '#';
                                        fz++;
                                }
                                break;
                        }
                }
        }
        for(ap = 0; ap < land ; ap++){
                printf("\t\t\t   %c\t|\t",table[ap][0]);
                for(kay = 1; kay < (sid + 1) ; kay++){
                        if(table[ap][kay] == '!')
                                printf("\t\t");
                        else if(table[ap][kay] == '#')
                                printf("%c=#\t\t",table[ap][0]);
                        else{
                                int mum = (int)(table[ap][kay]);
                                mum -= 65;
                                printf("%s\t\t",production[mum]);
                        }
                }
                printf("\n");

printf("\t\t\t-------------------------------------------------------------------------------------------------------------------
---");
                printf("\n");
```

```
	}
	int j;
	printf("\n\nPlease enter the desired INPUT STRING (make sure to include a '$' symbol
after the string) (e.g. abbb$) = ");
	char input[100];
	scanf("%s%c",input,&ch);

printf("\n\t\t\t\t\t===============================================================
==============\n");
	printf("\t\t\t\t\t\tStack\t\t\tInput\t\t\tAction");

printf("\n\t\t\t\t\t===============================================================
==============\n");
	int i_ptr = 0,s_ptr = 1;
	char stack[100];
	stack[0] = '$';
	stack[1] = table[0][0];
	while(s_ptr != -1){
		printf("\t\t\t\t\t\t");
		int vamp = 0;
		for(vamp=0;vamp<=s_ptr;vamp++){
			printf("%c",stack[vamp]);
		}
		printf("\t\t\t");
		vamp = i_ptr;
		while(input[vamp] != '\0'){
			printf("%c",input[vamp]);
			vamp++;
		}
		printf("\t\t\t");
		char her = input[i_ptr];
		char him = stack[s_ptr];
		s_ptr--;
		if(!isupper(him)){
			if(her == him){
				i_ptr++;
				printf("POP ACTION\n");
			}
			else{
				printf("\nString Not Accepted by LL(1) Parser !!\n");
				exit(0);
			}
		}
		else{
```

```
                        for(i=0;i<sid;i++){
                                if(ter[i] == her)
                                        break;
                        }
                        char produ[100];
                        for(j=0;j<land;j++){
                                if(him == table[j][0]){
                                        if (table[j][i+1] == '#'){
                                                printf("%c=#\n",table[j][0]);
                                                produ[0] = '#';
                                                produ[1] = '\0';
                                        }
                                        else if(table[j][i+1] != '!'){
                                                int mum = (int)(table[j][i+1]);
                                                mum -= 65;
                                                strcpy(produ,production[mum]);
                                                printf("%s\n",produ);
                                        }
                                        else{
                                                printf("\nString Not Accepted by LL(1) Parser !!\n");
                                                exit(0);
                                        }
                                }
                        }
                        int le = strlen(produ);
                        le = le - 1;
                        if(le == 0){
                                continue;
                        }
                        for(j=le;j>=2;j--){
                                s_ptr++;
                                stack[s_ptr] = produ[j];
                        }
                }
        }

printf("\n\t\t\t====================================================
======================================================\n");
        if (input[i_ptr] == '\0'){
                printf("\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN ACCEPTED !!\n");
        }
        else
                printf("\n\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN REJECTED !!\n");
printf("\t\t\t==========================================================================
=============================================================\n");
}

void follow(char c)
{
        int i ,j;
        if(production[0][0]==c){
                f[m++]='$';
        }
        for(i=0;i<10;i++)
        {
                for(j=2;j<10;j++)
                {
                        if(production[i][j]==c)
                        {
                        if(production[i][j+1]!='\0'){
                                        followfirst(production[i][j+1],i,(j+2));
                                }
                        if(production[i][j+1]=='\0'&&c!=production[i][0]){
                                follow(production[i][0]);
                                }
                        }
                }
        }
}

void findfirst(char c ,int q1 , int q2)
{
        int j;
        if(!(isupper(c))){
                first[n++]=c;
        }
        for(j=0;j<count;j++)
        {
                if(production[j][0]==c)
                {
                        if(production[j][2]=='#'){
                                if(production[q1][q2] == '\0')
                                        first[n++]='#';
                                else if(production[q1][q2] != '\0' && (q1 != 0 || q2 != 0))
                                {
                                        findfirst(production[q1][q2], q1, (q2+1));
                }
                else
                    first[n++]='#';
            }
            else if(!isupper(production[j][2])){
                first[n++]=production[j][2];
            }
            else {
                findfirst(production[j][2], j, 3);
            }
        }
    }
}

void followfirst(char c, int c1 , int c2)
{
  int k;
  if(!(isupper(c)))
        f[m++]=c;
    else{
        int i=0,j=1;
        for(i=0;i<count;i++)
        {
            if(calc_first[i][0] == c)
                break;
        }
        while(calc_first[i][j] != '!')
        {
            if(calc_first[i][j] != '#'){
                f[m++] = calc_first[i][j];
            }
            else{
                if(production[c1][c2] == '\0'){
                    follow(production[c1][0]);
                }
                else{
                    followfirst(production[c1][c2],c1,c2+1);
                }
            }
            j++;
        }
    }
}
```