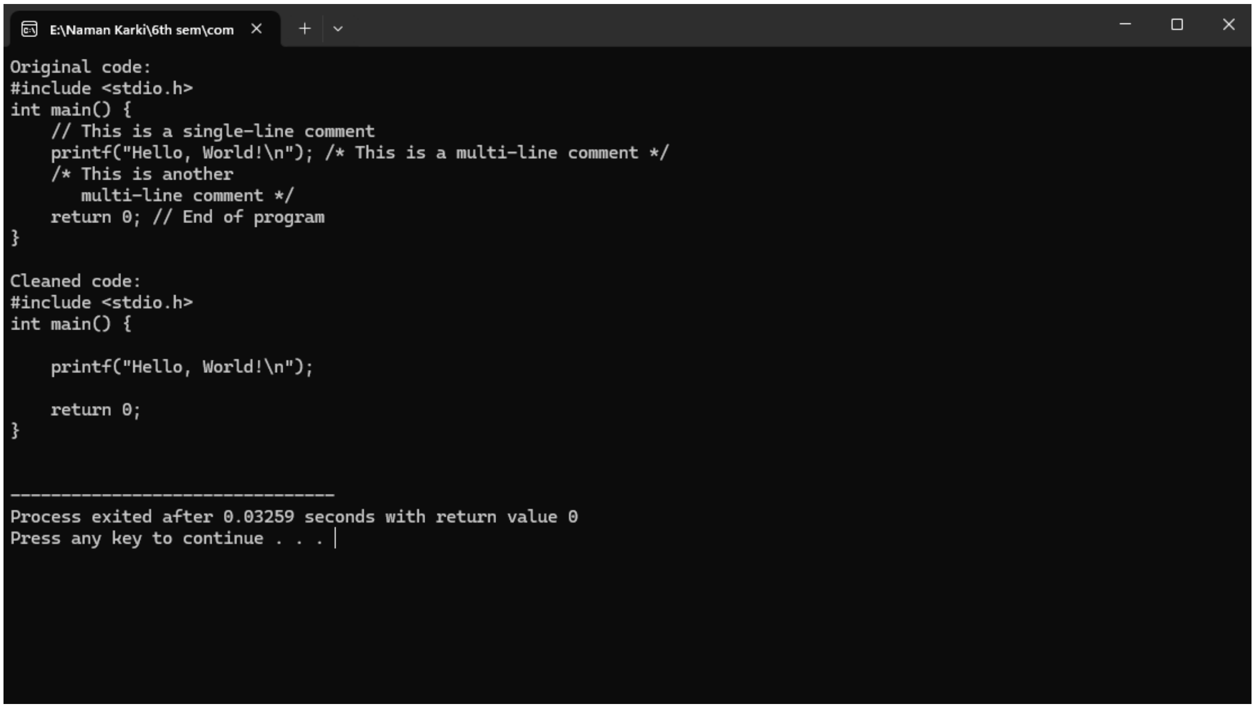
1) Write a program to check whether given string is valid comment or not.

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

void removeComments(const char *code, char *result) {
bool inSingleLineComment = false;
bool inMultiLineComment = false;
int j = 0;

for (int i = 0; code[i] != '\0'; ++i) {
if (inSingleLineComment) {
if (code[i] == '\n') {
inSingleLineComment = false;
result[j++] = code[i];
}
} else if (inMultiLineComment) {
if (code[i] == '*' && code[i + 1] == '/') {
inMultiLineComment = false;
i++;  // Skip '/'
}
} else {
if (code[i] == '/' && code[i + 1] == '/') {
inSingleLineComment = true;
i++;  // Skip '/'
} else if (code[i] == '/' && code[i + 1] == '*') {
inMultiLineComment = true;
i++;  // Skip '*'
} else {
result[j++] = code[i];
}
}
}
result[j] = '\0';
}

int main() {
const char *code = "#include <stdio.h>\n"
"int main() {\n"
"    // This is a single-line comment\n"
"    printf(\"Hello, World!\\n\"); /* This is a multi-line comment */\n"
"    /* This is another\n"
"       multi-line comment */\n"
"    return 0; // End of program\n"
"}\n";

char cleanedCode[1024];
removeComments(code, cleanedCode);
printf("Original code:\n%s\n", code);
printf("Cleaned code:\n%s\n", cleanedCode);
return 0;
}
```

Output:



```
Original code:
#include <stdio.h>
int main() {
    // This is a single-line comment
    printf("Hello, World!\n"); /* This is a multi-line comment */
    /* This is another
       multi-line comment */
    return 0; // End of program
}

Cleaned code:
#include <stdio.h>
int main() {

    printf("Hello, World!\n");

    return 0;
}


--------------------------------
Process exited after 0.03259 seconds with return value 0
Press any key to continue . . .
```

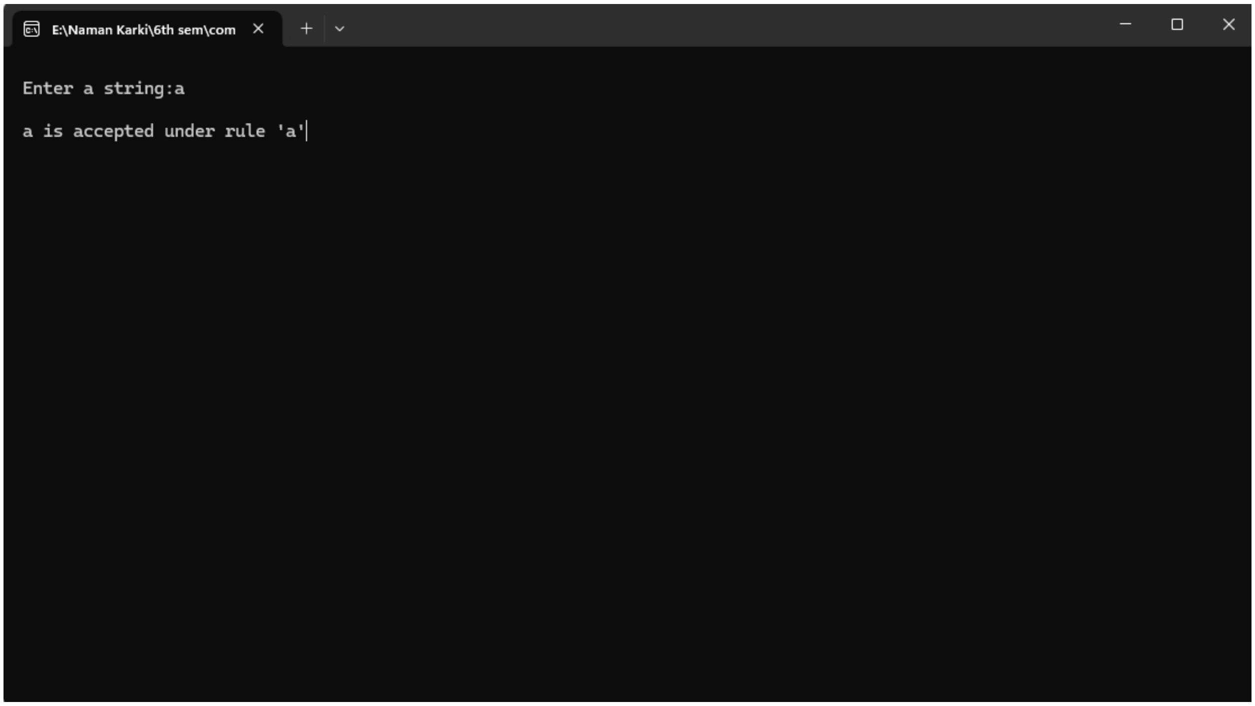
2)Write a program to recognize strings under a*, a*b+, abb.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
char s[20], c;
int state = 0, i = 0;
// clrscr();
printf("\n Enter a string:");
gets(s);
while (s[i] != '\0') {
switch (state) {
case 0:
c = s[i++];
if (c == 'a')
state = 1;
else if (c == 'b')
state = 2;
else
state = 6;
break;
case 1:
c = s[i++];
if (c == 'a')
state = 3;
else if (c == 'b')
state = 4;
else
state = 6;
break;
```

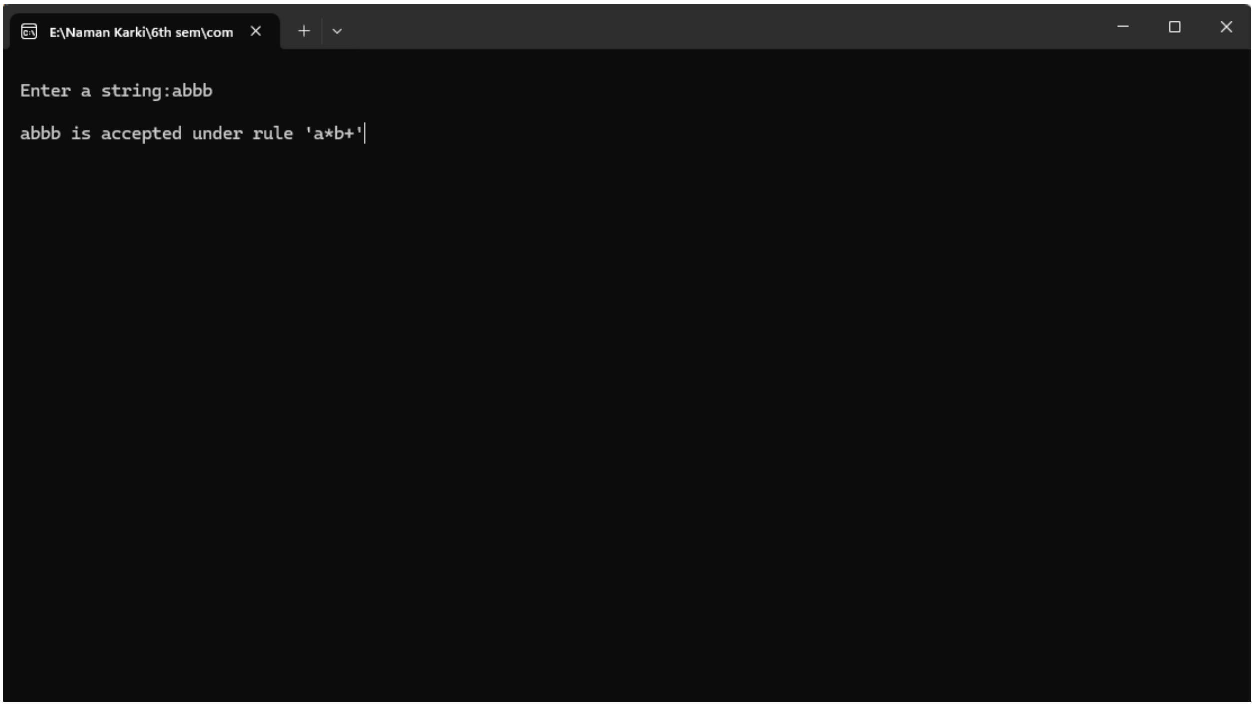
```
case 2:
c = s[i++];
if (c == 'a')
state = 6;
else if (c == 'b')
state = 2;
else
state = 6;
break;
case 3:
c = s[i++];
if (c == 'a')
state = 3;
else if (c == 'b')
state = 2;
else
state = 6;
break;
case 4:
c = s[i++];
if (c == 'a')
state = 6;
else if (c == 'b')
state = 5;
else
state = 6;
break;
case 5:
c = s[i++];
if (c == 'a')
state = 6;
else if (c == 'b')
```

```
state = 2;
else
state = 6;
break;
case 6:
printf("\n %s is not recognised.", s);
exit(0);
}
}
if (state == 1)
printf("\n %s is accepted under rule 'a'", s);
else if ((state == 2) || (state == 4))
printf("\n %s is accepted under rule 'a*b+'", s);
else if (state == 5)
printf("\n %s is accepted under rule 'abb'", s);
getch();
}
```

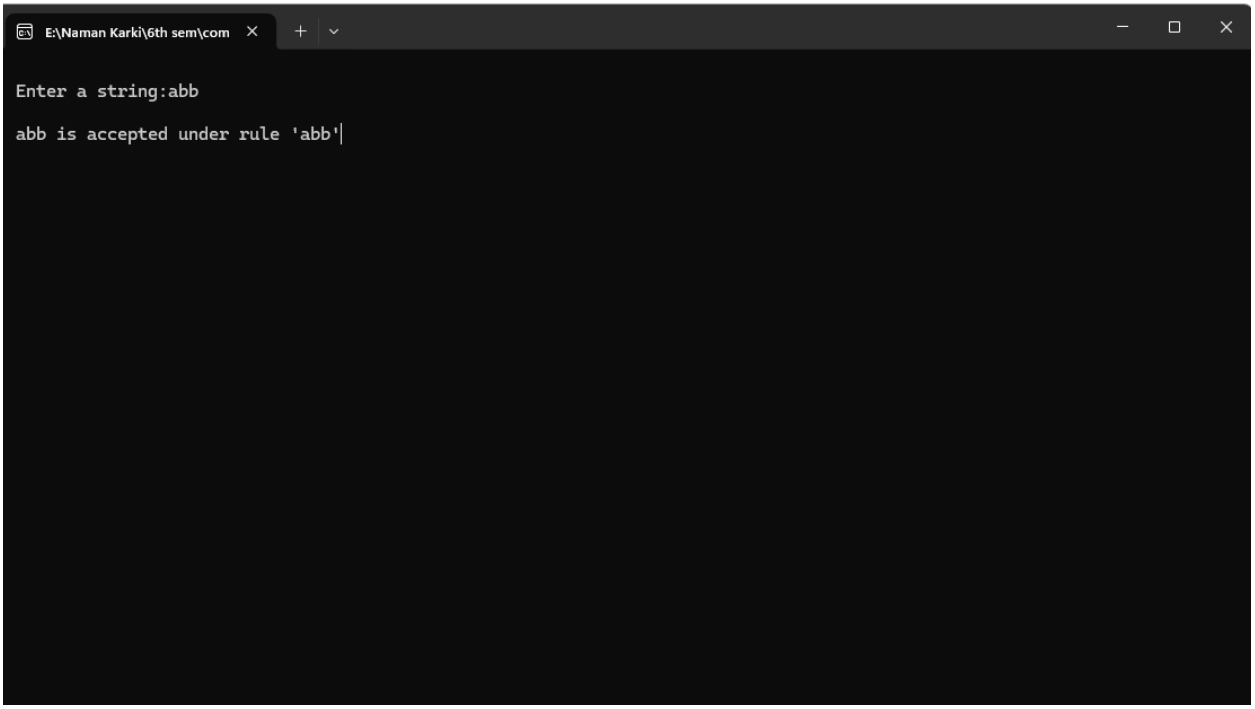
Output:



```
Enter a string:a

a is accepted under rule 'a'
```



```
Enter a string:abbb

abbb is accepted under rule 'a*b+'
```



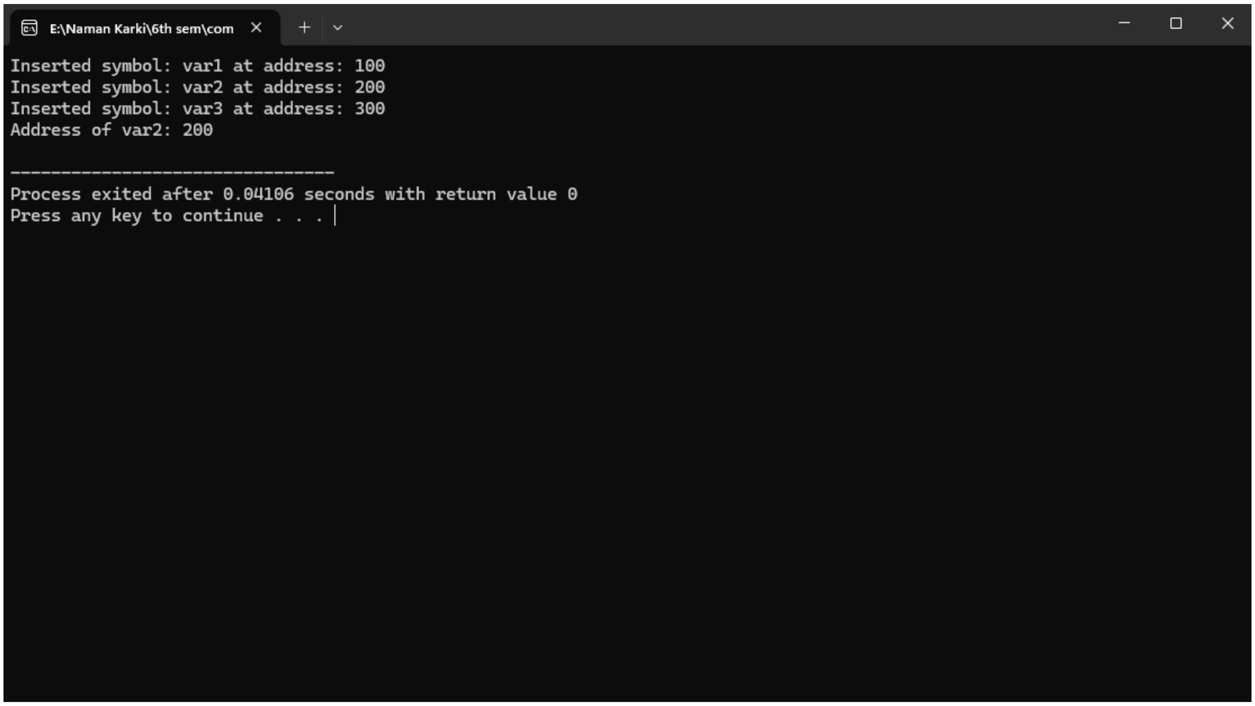
```
Enter a string:abb

abb is accepted under rule 'abb'
```

13) Write a program to implement symbol table.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_SYMBOLS 100
// Define a structure for each entry in the symbol table
typedef struct {
char name[50];
int address;
} SymbolEntry;
// Define the symbol table structure
typedef struct {
SymbolEntry entries[MAX_SYMBOLS];
int count;
} SymbolTable;

// Function to initialize the symbol table
void initializeSymbolTable(SymbolTable *table) {
table->count = 0;
}
// Function to insert a symbol into the symbol table
void insertSymbol(SymbolTable *table, char *name, int address) {
if (table->count < MAX_SYMBOLS) {
SymbolEntry *entry = &table->entries[table->count++];
strncpy(entry->name, name, sizeof(entry->name));
entry->address = address;
printf("Inserted symbol: %s at address: %d\n", name, address);
} else {
printf("Symbol table full. Cannot insert symbol: %s\n", name);
}
}
// Function to search for a symbol in the symbol table
```

```
int searchSymbol(SymbolTable *table, char *name) {
int i; // Declare 'i' outside the loop to conform with C89 standard
for (i = 0; i < table->count; i++) {
if (strcmp(table->entries[i].name, name) == 0) {
return table->entries[i].address;
}
}
return -1; // Symbol not found
}
int main() {
SymbolTable symbolTable;
initializeSymbolTable(&symbolTable);

// Insert some symbols into the table
insertSymbol(&symbolTable, "var1", 100);
insertSymbol(&symbolTable, "var2", 200);
insertSymbol(&symbolTable, "var3", 300);

// Search for a symbol
int address = searchSymbol(&symbolTable, "var2");
if (address != -1) {
printf("Address of var2: %d\n", address);
} else {
printf("Symbol not found\n");
}
return 0;
}
```

Outout:



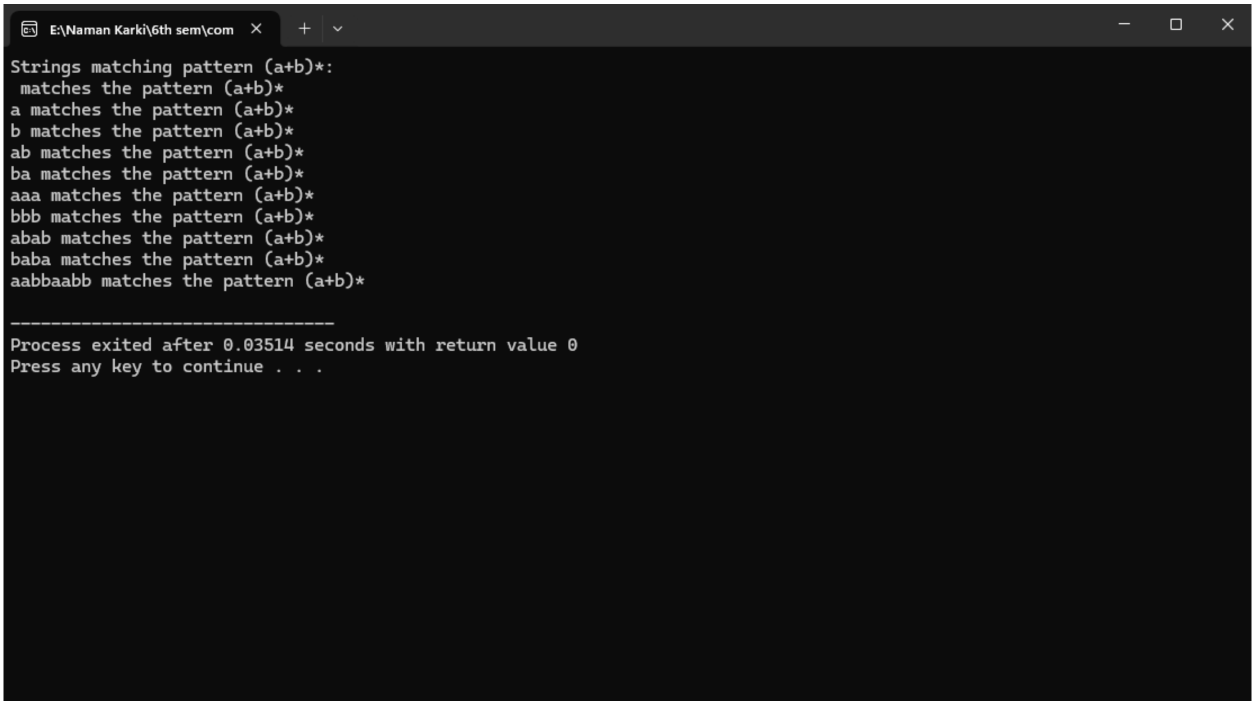
```
Inserted symbol: var1 at address: 100
Inserted symbol: var2 at address: 200
Inserted symbol: var3 at address: 300
Address of var2: 200

--------------------------------
Process exited after 0.04106 seconds with return value 0
Press any key to continue . . .
```

.

3) WAP to check the pattern (a+b)*

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

// Function to check if a string matches the pattern (a+b)*
int matches_pattern(const char *string) {
int i;
for (i = 0; i < strlen(string); i++) {
if (string[i] != 'a' && string[i] != 'b') {
return 0; // If any character is not 'a' or 'b', the string does not match
}
}
return 1; // The string matches if all characters are 'a' or 'b'
}
int main() {
const char *language[] = {"", "a", "b", "ab", "ba", "aaa", "bbb", "abab", "baba", "aabbaabb"};
int num_strings = sizeof(language) / sizeof(language[0]);
int i;
printf("Strings matching pattern (a+b)*:\n");
for (i = 0; i < num_strings; i++) {
if (matches_pattern(language[i])) {
printf("%s matches the pattern (a+b)*\n", language[i]);
} else {
printf("%s does not match the pattern (a+b)*\n", language[i]);
}
}
return 0;
}
```

Output:

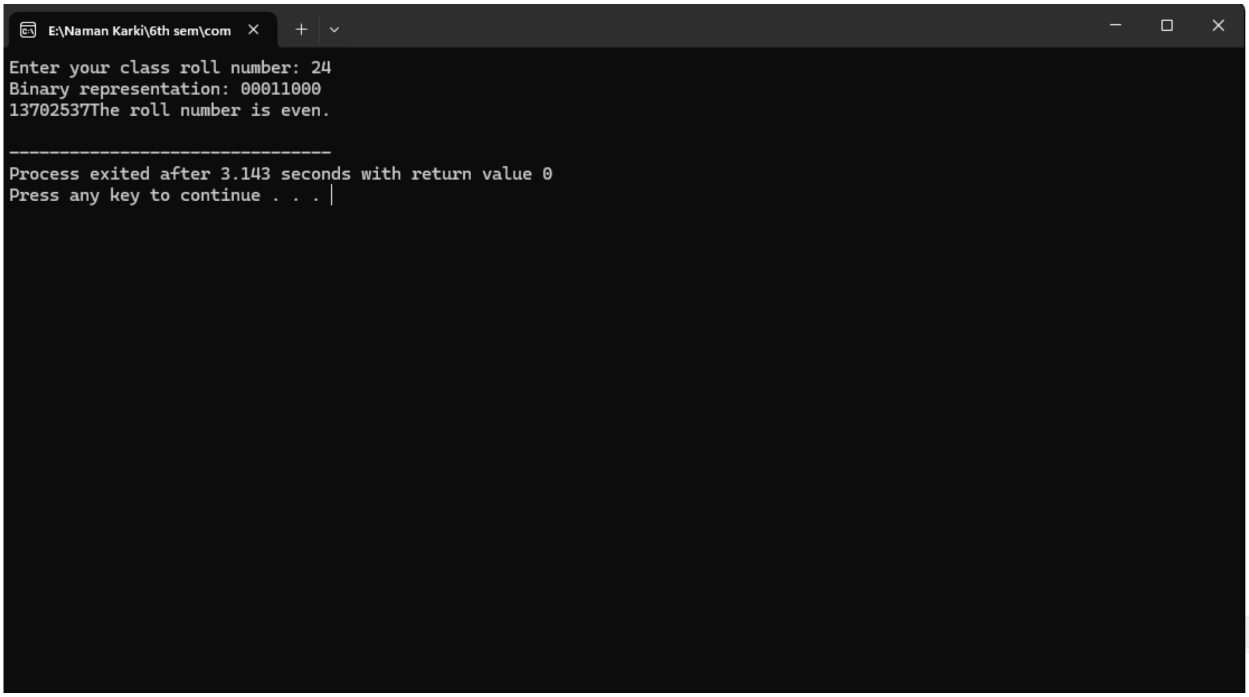


```
Strings matching pattern (a+b)*:
 matches the pattern (a+b)*
a matches the pattern (a+b)*
b matches the pattern (a+b)*
ab matches the pattern (a+b)*
ba matches the pattern (a+b)*
aaa matches the pattern (a+b)*
bbb matches the pattern (a+b)*
abab matches the pattern (a+b)*
baba matches the pattern (a+b)*
aabbaabb matches the pattern (a+b)*

--------------------------------
Process exited after 0.03514 seconds with return value 0
Press any key to continue . . .
```

4) WAP for DFA function to simulate the automaton

```
#include <stdio.h>

// DFA function to simulate the automaton
int dfa(int state, int input) {
// Transition table
int transition[8][2] = {
{1, 2}, {3, 4}, {5, 6}, {7, 0}, {1, 2}, {3, 4}, {5, 6}, {7, 0}
};
return transition[state][input];
}
int main() {
int roll_number;
printf("Enter your class roll number: ");
scanf("%d", &roll_number);
// Convert roll number to binary and display it
int binary[8];
int i, state = 0;
printf("Binary representation: ");
for (i = 7; i >= 0; i--) {
binary[i] = roll_number % 2;
printf("%d", binary[i]);
roll_number /= 2;
}
printf("\n");
// Implement DFA
for (i = 0; i < 8; i++) {
state = dfa(state, binary[i]);
printf("%d",state);
}
// Check if the final state is accepting (even)
if (state == 0 || state == 4 || state == 6) {
printf("The roll number is odd .\n");
} else {
printf("The roll number is even.\n");
}

return 0;
}
```

Output:

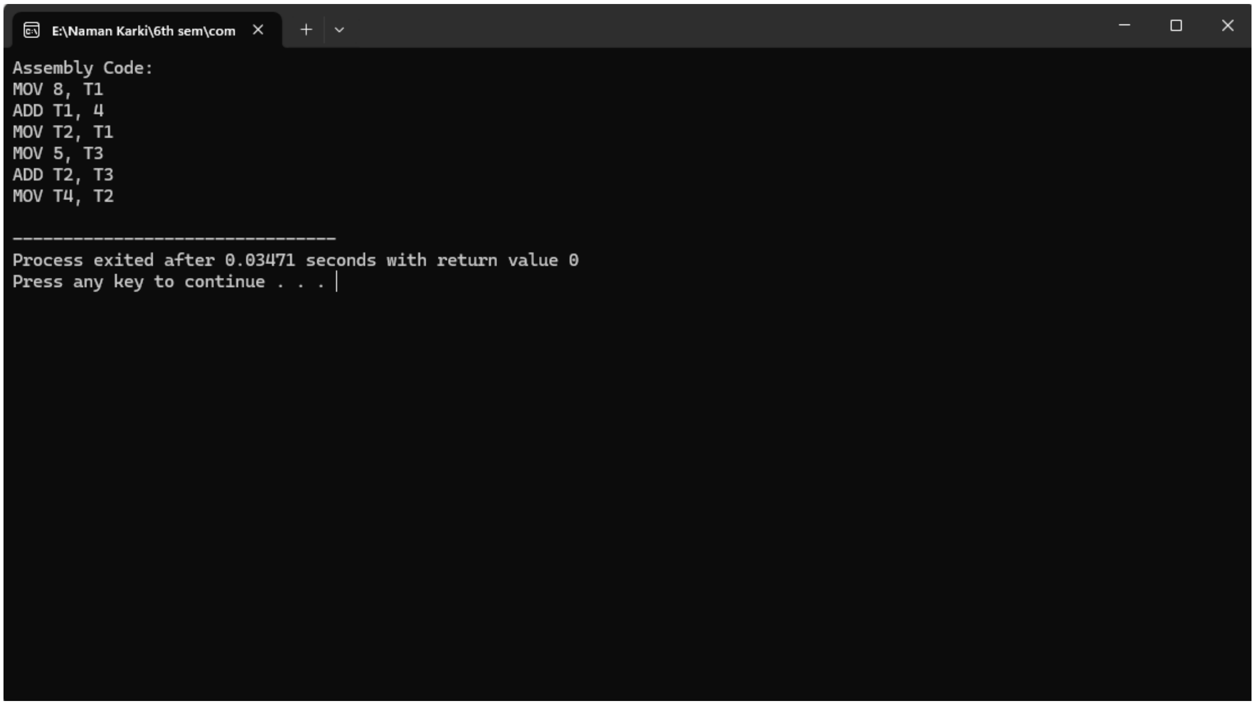


```
Enter your class roll number: 24
Binary representation: 00011000
13702537The roll number is even.

--------------------------------
Process exited after 3.143 seconds with return value 0
Press any key to continue . . .
```

12) Write a program to implement final code (Assembly code) of given intermediate code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_CODE_SIZE 100
// Define the structure for intermediate code instructions
struct Instruction {
char opcode[10];
char operand1[10];
char operand2[10];
char result[10];
};
void generateAssemblyCode(struct Instruction* code, int codeSize) {
printf("Assembly Code:\n");
int i;
for (i = 0; i < codeSize; i++) {
if (strcmp(code[i].opcode, "LOAD") == 0) {
printf("MOV %s, %s\n", code[i].operand1, code[i].result);
} else if (strcmp(code[i].opcode, "ADD") == 0) {
printf("ADD %s, %s\n", code[i].operand1, code[i].operand2);
printf("MOV %s, %s\n", code[i].result, code[i].operand1);
}
}
}
int main() {
struct Instruction code[MAX_CODE_SIZE];
int codeSize = 0;
strcpy(code[codeSize].opcode, "LOAD");
strcpy(code[codeSize].operand1, "8");
strcpy(code[codeSize].operand2, "-");
strcpy(code[codeSize].result, "T1");
codeSize++;
strcpy(code[codeSize].opcode, "ADD");
strcpy(code[codeSize].operand1, "T1");
strcpy(code[codeSize].operand2, "4");
strcpy(code[codeSize].result, "T2");
codeSize++;
strcpy(code[codeSize].opcode, "LOAD");
strcpy(code[codeSize].operand1, "5");
strcpy(code[codeSize].operand2, "-");
strcpy(code[codeSize].result, "T3");
codeSize++;
strcpy(code[codeSize].opcode, "ADD");
strcpy(code[codeSize].operand1, "T2");
strcpy(code[codeSize].operand2, "T3");
strcpy(code[codeSize].result, "T4");
codeSize++;
// Generate assembly code
generateAssemblyCode(code, codeSize);
return 0;
}
```

Output:



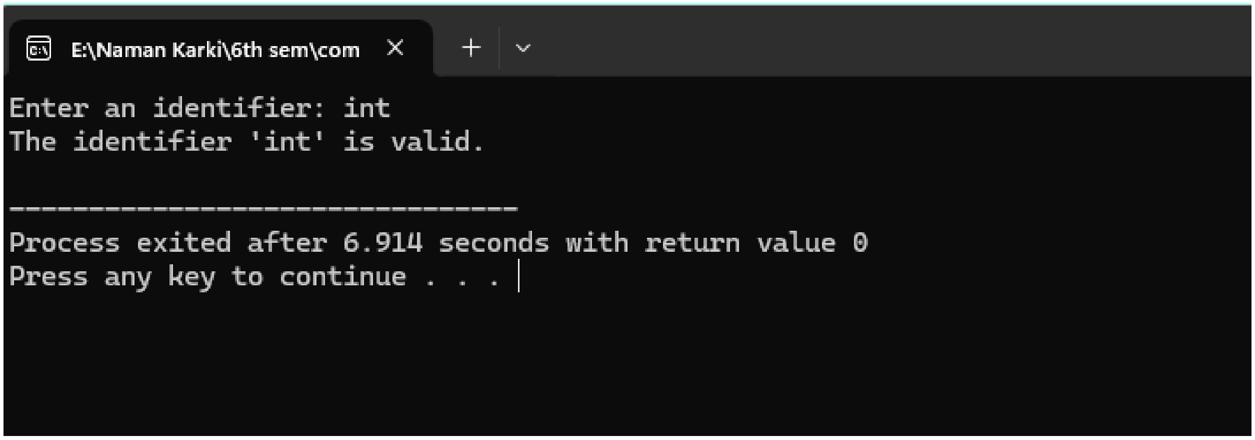
```
Assembly Code:
MOV 8, T1
ADD T1, 4
MOV T2, T1
MOV 5, T3
ADD T2, T3
MOV T4, T2

--------------------------------
Process exited after 0.03471 seconds with return value 0
Press any key to continue . . .
```

5) Write a programto test the given identidfier is valid or not?

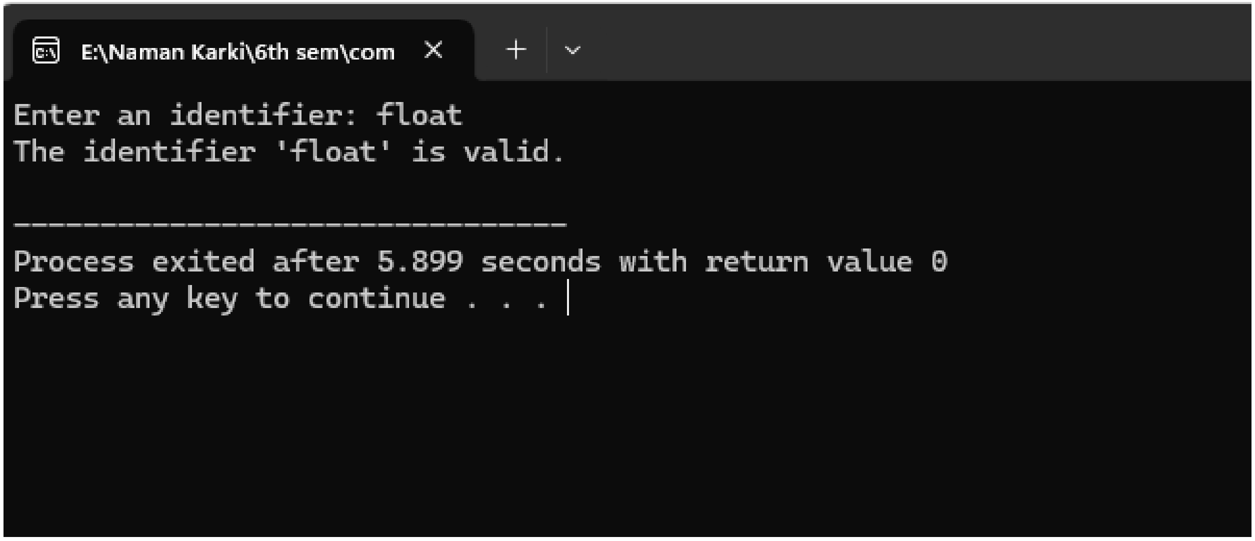
```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int isValidIdentifier(const char *identifier) {
// Check if the identifier is empty
if (strlen(identifier) == 0) {
return 0; // Invalid if empty
}

// Check the first character
if (!(isalpha(identifier[0]) || identifier[0] == '_')) {
return 0; // Invalid if not a letter or underscore
}

// Check subsequent characters
int length = strlen(identifier);
int i;
for (i = 1; i < length; i++) {
if (!(isalpha(identifier[i]) || isdigit(identifier[i]) || identifier[i] == '_')) {
return 0; // Invalid if not a letter, digit, or underscore
}
}

// If all checks pass, the identifier is valid
return 1;
}

int main() {
```

```
char identifier[100];
int i;

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

if (isValidIdentifier(identifier)) {
printf("The identifier '%s' is valid.\n", identifier);
} else {
printf("The identifier '%s' is invalid.\n", identifier);
}    return 0;
}
```

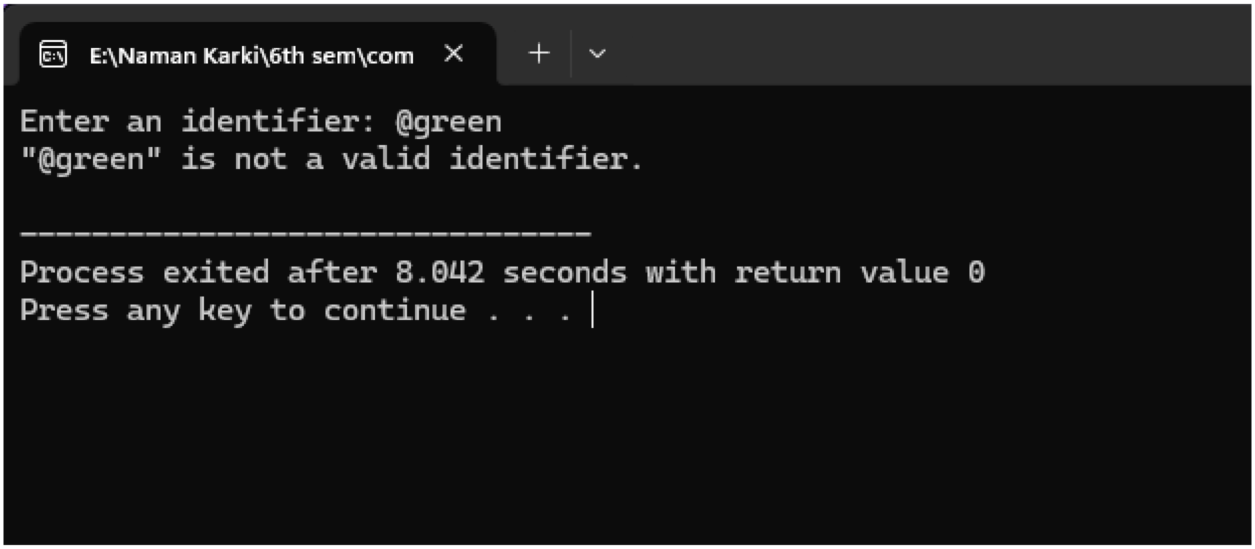
Output:



```
Enter an identifier: int
The identifier 'int' is valid.

--------------------------------
Process exited after 6.914 seconds with return value 0
Press any key to continue . . .
```



```
Enter an identifier: float
The identifier 'float' is valid.

--------------------------------
Process exited after 5.899 seconds with return value 0
Press any key to continue . . .
```



```
Enter an identifier: @green
"@green" is not a valid identifier.

--------------------------------
Process exited after 8.042 seconds with return value 0
Press any key to continue . . .
```

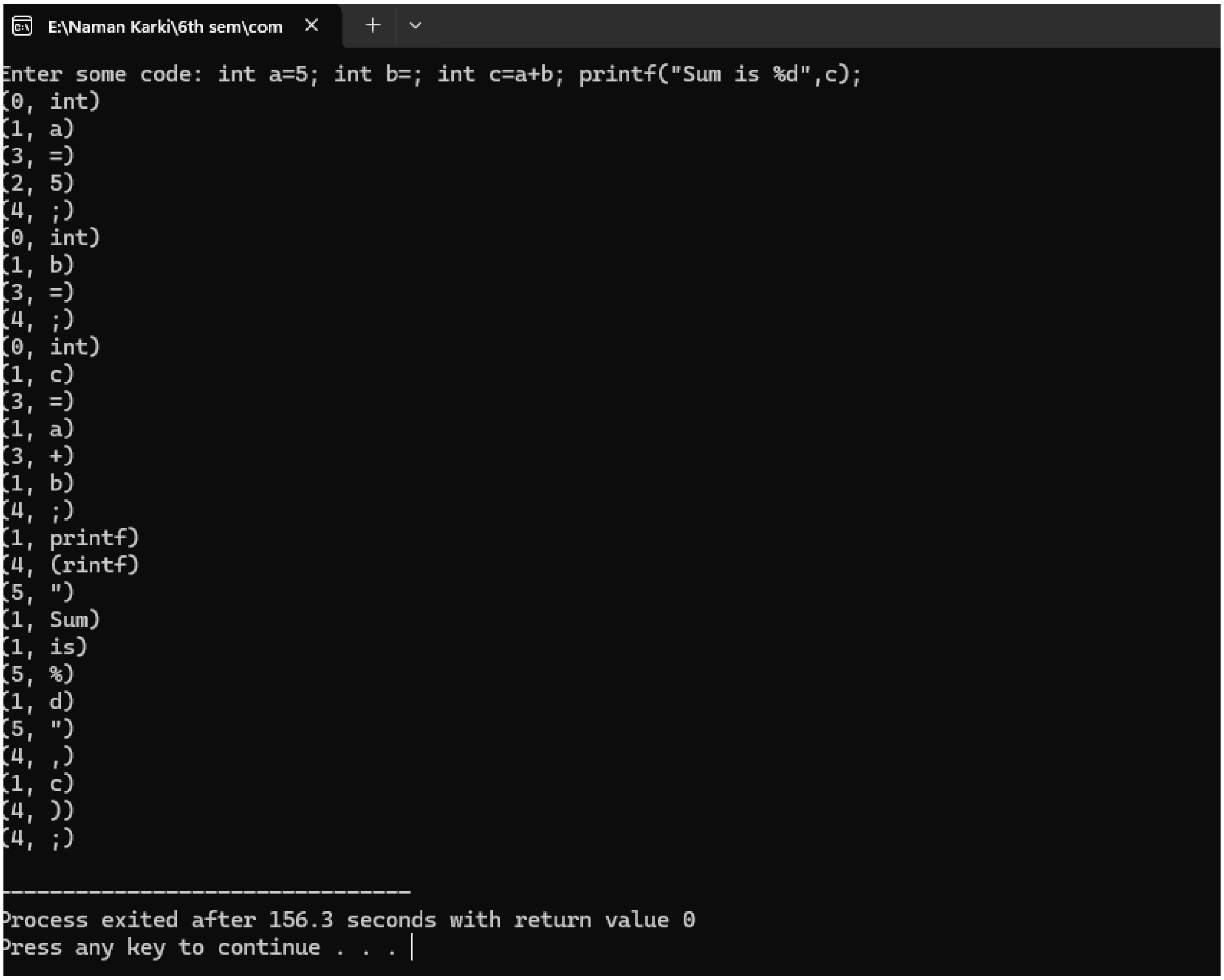
6)Write a program for lexical analyzer

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

// Token types
enum TokenType {
KEYWORD,
IDENTIFIER,
NUMBER,
OPERATOR,
SEPARATOR,
INVALID
};

// Function to check if a string is a keyword
int isKeyword(char* word) {
char keywords[6][10] = {"int", "float", "if", "else", "while", "for"};
int i;
for (i = 0; i < 6; i++) {
if (strcmp(keywords[i], word) == 0) {
return 1;
}
}
return 0;
}

// Function to check if a character is an operator
int isOperator(char c) {
char operators[] = "+-*/=!<>";
```

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

// Function to check if a character is a separator
int isSeparator(char c) {
char separators[] = ";,(){}";
int i;
for (i = 0; separators[i]; i++) {
if (separators[i] == c) {
return 1;
}
}
return 0;
}

// Function to tokenize the input string
void tokenize(char* input) {
char buffer[50];
int bufferIndex = 0;
int i;

for (i = 0; input[i]; i++) {
if (isalpha(input[i])) { // Identifier or keyword
buffer[bufferIndex++] = input[i];
while (isalnum(input[i + 1])) {
buffer[bufferIndex++] = input[++i];
}
buffer[bufferIndex] = '\0';
if (isKeyword(buffer)) {
printf("(%d, %s)\n", KEYWORD, buffer);
} else {
printf("(%d, %s)\n", IDENTIFIER, buffer);
}
bufferIndex = 0;
} else if (isdigit(input[i])) { // Number
buffer[bufferIndex++] = input[i];
while (isdigit(input[i + 1]) || input[i + 1] == '.') {
buffer[bufferIndex++] = input[++i];
}
buffer[bufferIndex] = '\0';
printf("(%d, %s)\n", NUMBER, buffer);
bufferIndex = 0;
} else if (isOperator(input[i])) { // Operator
buffer[bufferIndex++] = input[i];
printf("(%d, %s)\n", OPERATOR, buffer);
bufferIndex = 0;
} else if (isSeparator(input[i])) { // Separator
buffer[bufferIndex++] = input[i];
printf("(%d, %s)\n", SEPARATOR, buffer);
bufferIndex = 0;
} else if (!isspace(input[i])) { // Invalid character
printf("(%d, %c)\n", INVALID, input[i]);
}
}
}

// Main function
int main() {
```

```
char input[100];

printf("Enter some code: ");

fgets(input, sizeof(input), stdin);

tokenize(input);

return 0;

}
```

Output:

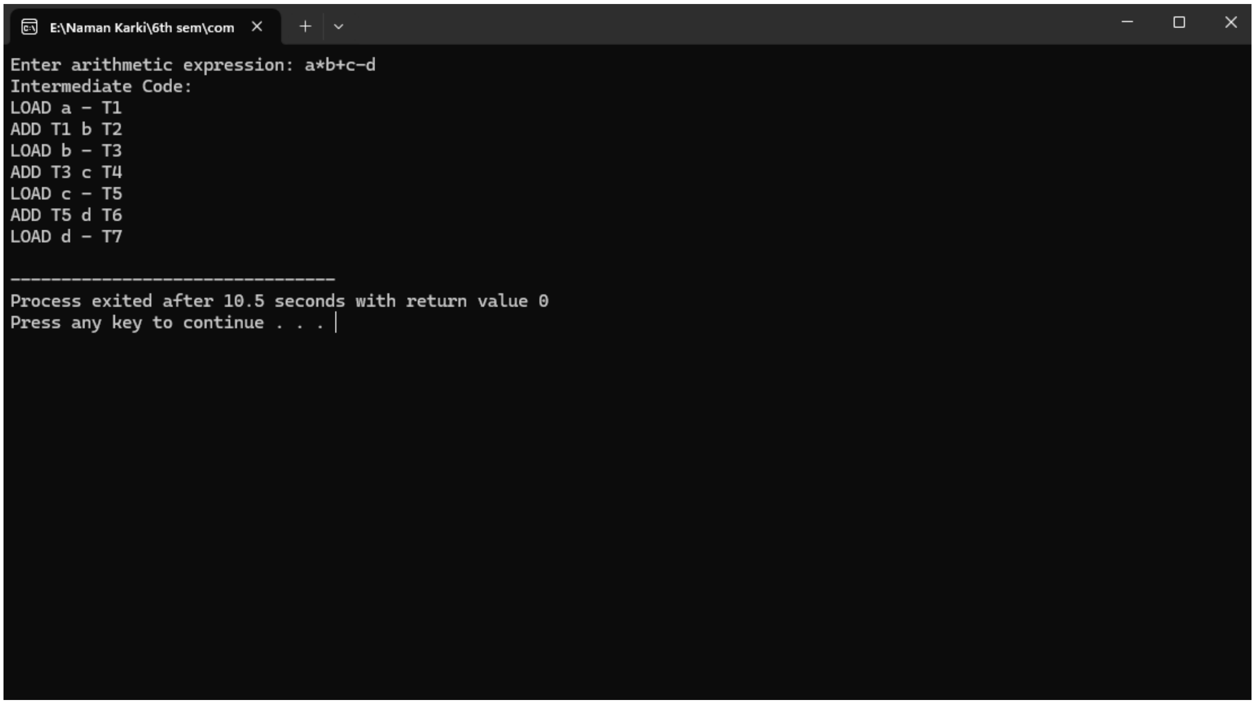


```
Enter some code: int a=5; int b=; int c=a+b; printf("Sum is %d",c);
(0, int)
(1, a)
(3, =)
(2, 5)
(4, ;)
(0, int)
(1, b)
(3, =)
(4, ;)
(0, int)
(1, c)
(3, =)
(1, a)
(3, +)
(1, b)
(4, ;)
(1, printf)
(4, (rintf)
(5, ")
(1, Sum)
(1, is)
(5, %)
(1, d)
(5, ")
(4, ,)
(1, c)
(4, ))
(4, ;)

--------------------------------
Process exited after 156.3 seconds with return value 0
Press any key to continue . . .
```

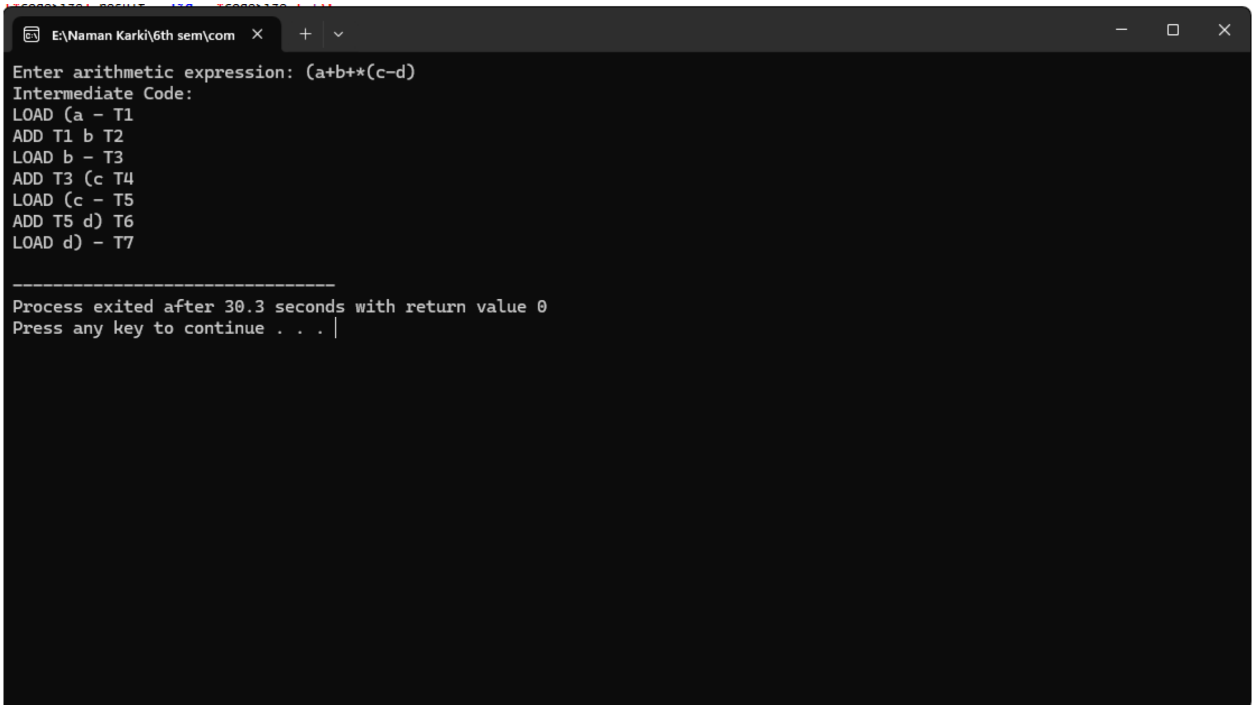
11) Write a program to implement immediate code generator.

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

#define MAX_CODE_SIZE 100

// Define the structure for intermediate code instructions
struct Instruction {
char opcode[10];
char operand1[10];
char operand2[10];
char result[10];
};

// Function to generate intermediate code for arithmetic expressions
void generateIntermediateCode(char* expression, struct Instruction* code, int* codeSize) {
char* token = strtok(expression, "+-*/");

while (token != NULL) {
strcpy(code[*codeSize].opcode, "LOAD");
strcpy(code[*codeSize].operand1, token);
strcpy(code[*codeSize].operand2, "-");
sprintf(code[*codeSize].result, "T%d", *codeSize + 1);
(*codeSize)++;

token = strtok(NULL, "+-*/");
if (token != NULL) {
strcpy(code[*codeSize].opcode, "ADD");
strcpy(code[*codeSize].operand1, code[*codeSize - 1].result);
strcpy(code[*codeSize].operand2, token);
sprintf(code[*codeSize].result, "T%d", *codeSize + 1);
(*codeSize)++;
}
}
}

// Function to print intermediate code
void printIntermediateCode(struct Instruction* code, int codeSize) {
printf("Intermediate Code:\n");
int i; // Variable 'i' declaration moved inside the function
for (i = 0; i < codeSize; i++) { // Adjusted loop structure
printf("%s %s %s %s\n", code[i].opcode, code[i].operand1, code[i].operand2, code[i].result);
}
}

int main() {
char expression[100];
struct Instruction code[MAX_CODE_SIZE];
int codeSize = 0;

printf("Enter arithmetic expression: ");
scanf("%s", expression);

generateIntermediateCode(expression, code, &codeSize);
printIntermediateCode(code, codeSize);

return 0;
}
```

Output:



```
Enter arithmetic expression: a*b+c-d
Intermediate Code:
LOAD a - T1
ADD T1 b T2
LOAD b - T3
ADD T3 c T4
LOAD c - T5
ADD T5 d T6
LOAD d - T7

--------------------------------
Process exited after 10.5 seconds with return value 0
Press any key to continue . . .
```



```
Enter arithmetic expression: (a+b+*(c-d)
Intermediate Code:
LOAD (a - T1
ADD T1 b T2
LOAD b - T3
ADD T3 (c T4
LOAD (c - T5
ADD T5 d) T6
LOAD d) - T7

--------------------------------
Process exited after 30.3 seconds with return value 0
Press any key to continue . . .
```

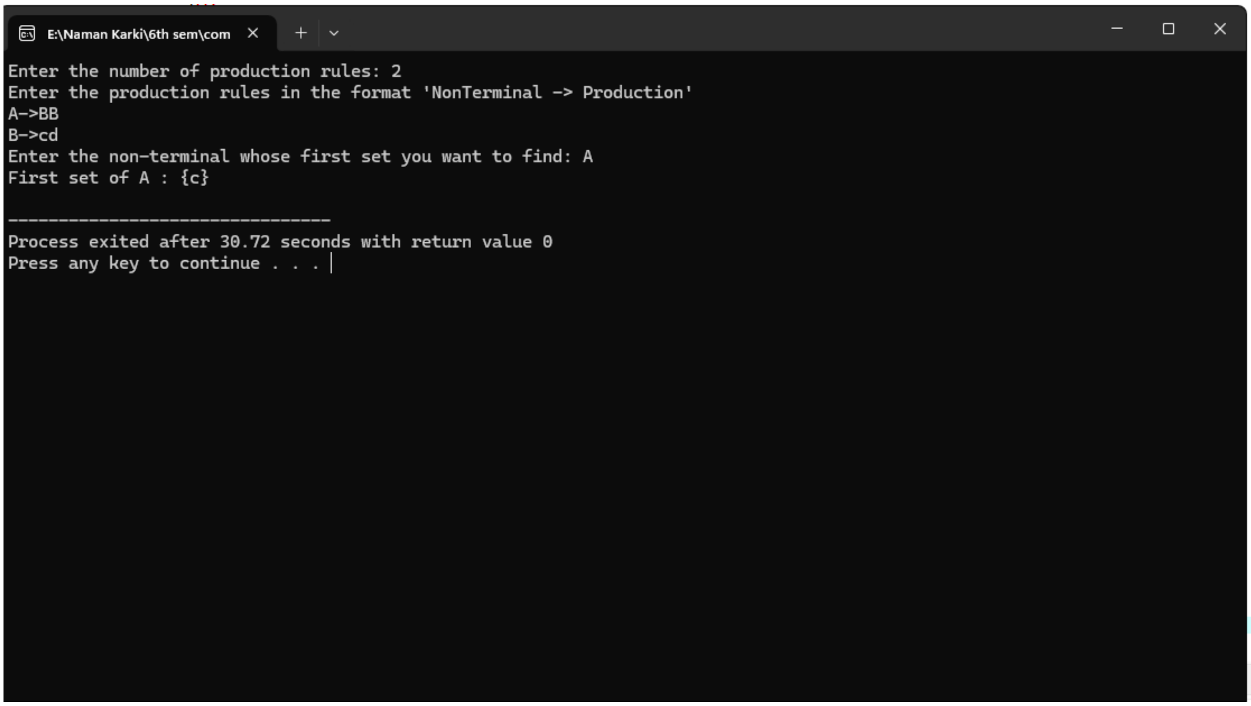
7) Write a program to find first of given grammar.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>


#define MAX_RULES 10

#define MAX_LENGTH 10


// Structure to represent a production rule

struct Rule {

char nonTerminal;

char production[MAX_LENGTH];

};


// Function to check if a symbol is terminal

int isTerminal(char symbol) {

return islower(symbol) || symbol == '$';

}


// Function to check if a symbol is non-terminal

int isNonTerminal(char symbol) {

return isupper(symbol);

}


// Function to add a symbol to a set

void addToSet(char set[], char symbol) {

if (!strchr(set, symbol)) {

strncat(set, &symbol, 1);

}

}
```

```
// Function to find the first set for a given grammar
void findFirstSet(struct Rule rules[], int ruleCount, char nonTerminal, char firstSet[]) {
int i;
for (i = 0; i < ruleCount; i++) {
if (rules[i].nonTerminal == nonTerminal) {
char symbol = rules[i].production[0];
if (isTerminal(symbol) && symbol != '$') {
addToSet(firstSet, symbol);
} else if (isNonTerminal(symbol)) {
findFirstSet(rules, ruleCount, symbol, firstSet);
} else if (symbol == '$' && strlen(rules[i].production) == 1) {
addToSet(firstSet, '$');
} else {
int j = 0;
while (symbol != '\0') {
findFirstSet(rules, ruleCount, symbol, firstSet);
if (strchr(firstSet, '$')) {
j++;
symbol = rules[i].production[j];
} else {
break;
}
}
}
}
}
}

int main() {
struct Rule rules[MAX_RULES];
int ruleCount;
```

```c
char nonTerminal;
char firstSet[MAX_LENGTH] = "";

printf("Enter the number of production rules: ");
scanf("%d", &ruleCount);
getchar(); // Clear newline character from buffer

printf("Enter the production rules in the format 'NonTerminal -> Production'\n");
int i;
for (i = 0; i < ruleCount; i++) {
scanf("%c -> %[^\n]s", &rules[i].nonTerminal, rules[i].production);
getchar(); // Clear newline character from buffer
}

printf("Enter the non-terminal whose first set you want to find: ");
scanf("%c", &nonTerminal);

findFirstSet(rules, ruleCount, nonTerminal, firstSet);

printf("First set of %c : {%s}\n", nonTerminal, firstSet);

return 0;
}
```

Output:



```
Enter the number of production rules: 2
Enter the production rules in the format 'NonTerminal -> Production'
A->BB
B->cd
Enter the non-terminal whose first set you want to find: A
First set of A : {c}

--------------------------------
Process exited after 30.72 seconds with return value 0
Press any key to continue . . .
```

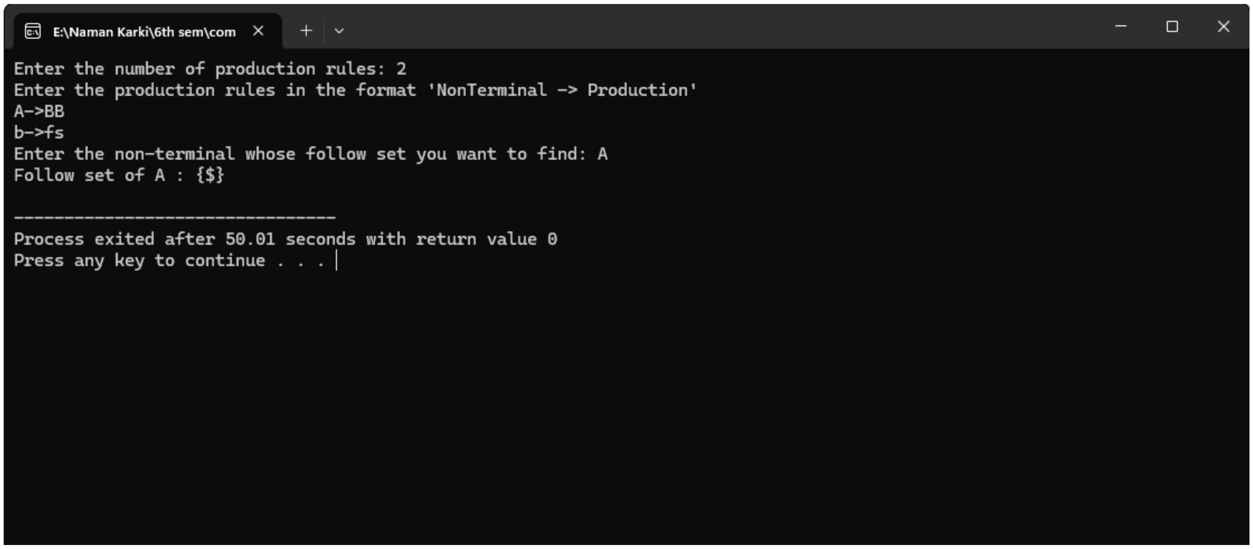
8) Write a program to find follow of given grammar.

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

#define MAX_RULES 10
#define MAX_LENGTH 10

// Structure to represent a production rule
struct Rule {
char nonTerminal;
char production[MAX_LENGTH];
};

// Structure to represent a follow set
struct FollowSet {
char nonTerminal;
char follow[MAX_LENGTH];
};

// Function to check if a symbol is terminal
int isTerminal(char symbol) {
return islower(symbol) || symbol == '$';
}

// Function to check if a symbol is non-terminal
int isNonTerminal(char symbol) {
return isupper(symbol);
}
```

```
// Function to add a symbol to a set
void addToSet(char set[], char symbol) {
if (!strchr(set, symbol)) {
strncat(set, &symbol, 1);
}
}

// Function to find the follow set for a given grammar
void findFollowSet(struct Rule rules[], int ruleCount, struct FollowSet followSets[], int
followSetCount, char nonTerminal) {
int i, j;
for (i = 0; i < ruleCount; i++) {
char* ptr = strchr(rules[i].production, nonTerminal);
if (ptr) {
while (*(ptr + 1)) {
char symbol = *(ptr + 1);
if (isTerminal(symbol)) {
addToSet(followSets[followSetCount].follow, symbol);
break;
} else if (isNonTerminal(symbol)) {
char firstSet[MAX_LENGTH] = "";
int foundEpsilon = 0;
for (j = 0; j < ruleCount; j++) {
if (rules[j].nonTerminal == symbol) {
if (rules[j].production[0] == '$' || isTerminal(rules[j].production[0])) {
addToSet(firstSet, rules[j].production[0]);
} else {
findFollowSet(rules, ruleCount, followSets, followSetCount, rules[j].production[0]);
strcat(firstSet, followSets[followSetCount].follow);
}
if (strchr(rules[j].production, '$')) {
foundEpsilon = 1;
} else {
foundEpsilon = 0;
break;
}
}
}
if (foundEpsilon) {
ptr++;
}
strcat(followSets[followSetCount].follow, firstSet);
} else {
break;
}
}
if (!*(ptr + 1)) {
for (j = 0; j < followSetCount; j++) {
if (followSets[j].nonTerminal == rules[i].nonTerminal) {
strcat(followSets[j].follow, followSets[followSetCount].follow);
break;
}
}
}
}
}
}

int main() {
struct Rule rules[MAX_RULES];
struct FollowSet followSets[MAX_RULES];
int ruleCount, followSetCount;
char nonTerminal;
```

```
printf("Enter the number of production rules: ");
scanf("%d", &ruleCount);
getchar(); // Clear newline character from buffer

printf("Enter the production rules in the format 'NonTerminal -> Production'\n");
int i;
for (i = 0; i < ruleCount; i++) {
scanf("%c -> %[^\n]s", &rules[i].nonTerminal, rules[i].production);
getchar(); // Clear newline character from buffer
}

printf("Enter the non-terminal whose follow set you want to find: ");
scanf("%c", &nonTerminal);

followSetCount = 0;
for (i = 0; i < ruleCount; i++) {
if (rules[i].nonTerminal == nonTerminal) {
strcpy(followSets[followSetCount].follow, "$");
followSets[followSetCount].nonTerminal = nonTerminal;
followSetCount++;
break;
}
}

findFollowSet(rules, ruleCount, followSets, followSetCount, nonTerminal);

printf("Follow set of %c : {%s}\n", nonTerminal, followSets[0].follow);

return 0;
}
```

Output:



```
Enter the number of production rules: 2
Enter the production rules in the format 'NonTerminal -> Production'
A->BB
b->fs
Enter the non-terminal whose follow set you want to find: A
Follow set of A : {$}

--------------------------------
Process exited after 50.01 seconds with return value 0
Press any key to continue . . .
```

9) WAP to construct LL(1) table of given grammer.

```
#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 = ");
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)){
```

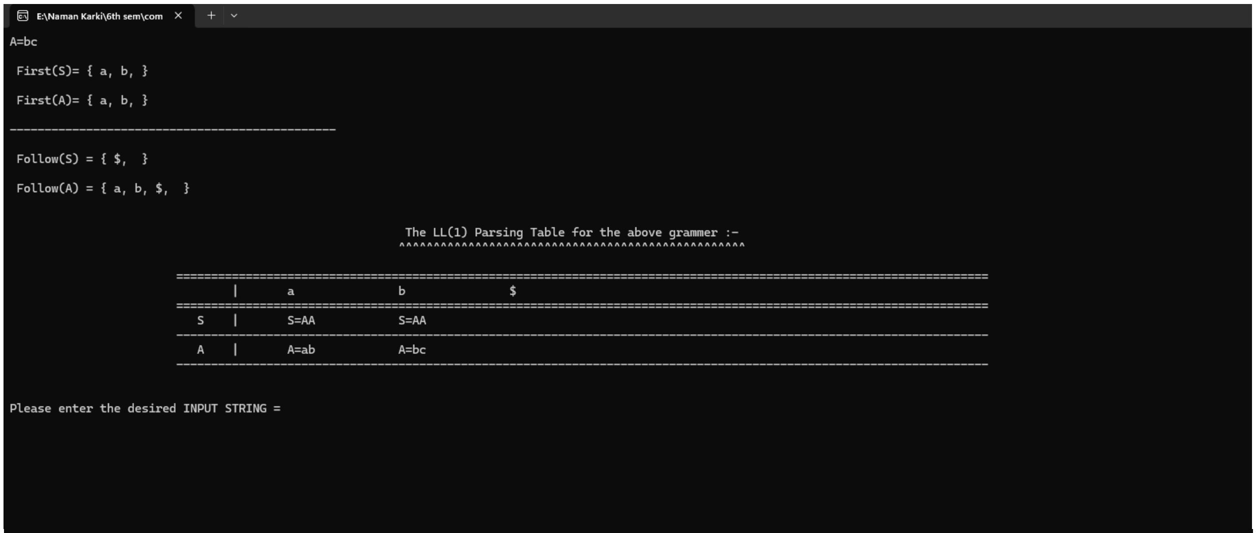
```
if(her == him){
i_ptr++;
printf("POP ACTION\n");
}
else{
printf("\nString Not Accepted by LL(1) Parser !!\n");
exit(0);
}
}
else{
for(i=0;i<sid;i++){
if(ter[i] == her)
break;
}
char produ[100];
for(j=0;j<land;j++){
if(him == table[j][0]){
if (table[j][i+1] == '#'){
printf("%c=#\n",table[j][0]);
produ[0] = '#';
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));
}
```

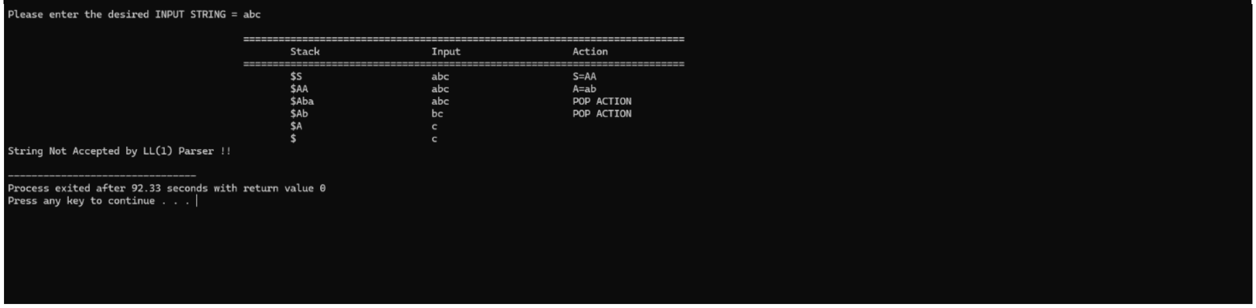
```
else
first[n++]='#';
}
else if(!isupper(production[j][2])){
first[n++]=production[j][2];
}
else {
findfirst(production[j][2], j, 3);
}
}
}
}

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++;
}
}
}
```

Output:



```
A=bc

 First(S)= { a, b, }

 First(A)= { a, b, }

-----------------------------------------------

 Follow(S) = { $,  }

 Follow(A) = { a, b, $,  }


                                                       The LL(1) Parsing Table for the above grammer :-
                                                      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

                        ===================================================================================================================
                                |       a               b               $
                        ===================================================================================================================
                           S    |       S=AA            S=AA
                        -------------------------------------------------------------------------------------------------------------------
                           A    |       A=ab            A=bc
                        -------------------------------------------------------------------------------------------------------------------


Please enter the desired INPUT STRING =
```



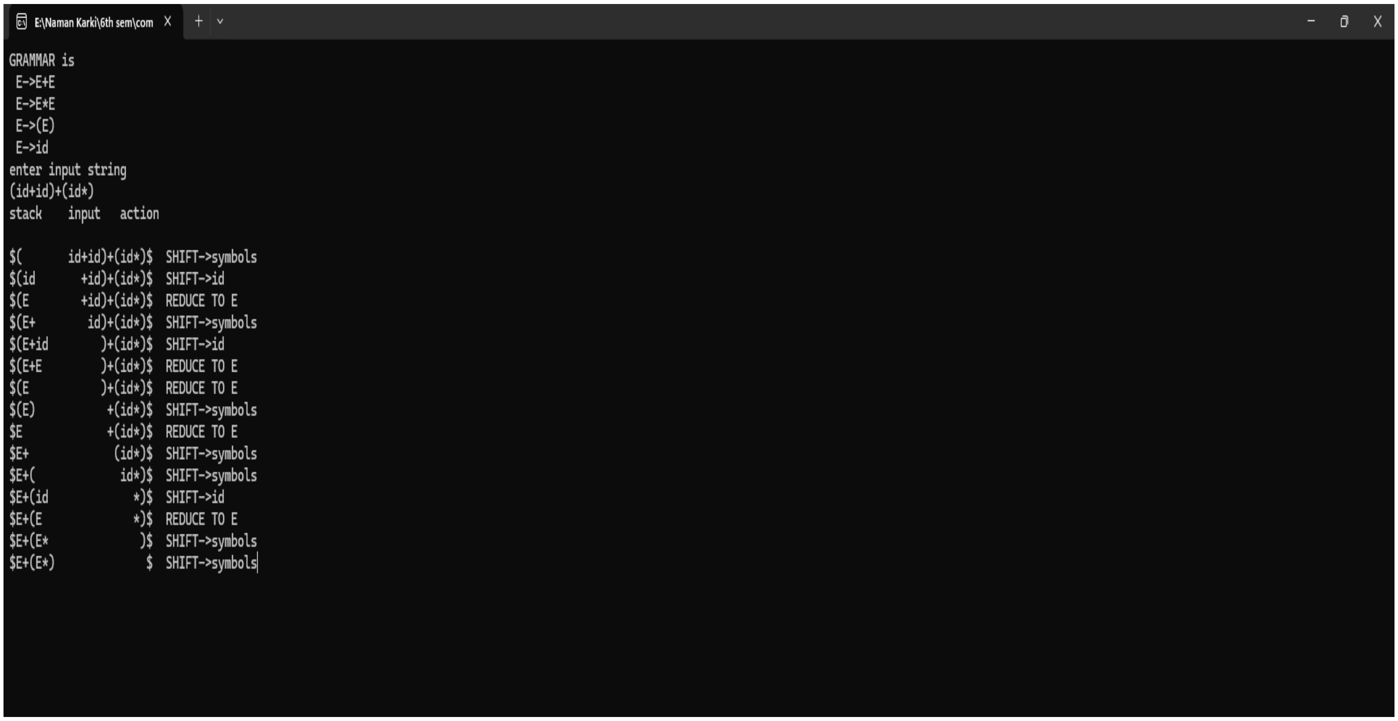
```
Please enter the desired INPUT STRING = abc

                                        ===========================================================================
                                                Stack                   Input                   Action
                                        ===========================================================================
                                                $S                      abc                     S=AA
                                                $AA                     abc                     A=ab
                                                $Aba                    abc                     POP ACTION
                                                $Ab                     bc                      POP ACTION
                                                $A                      c
                                                $                       c
String Not Accepted by LL(1) Parser !!

--------------------------------
Process exited after 92.33 seconds with return value 0
Press any key to continue . . .
```

10) WAP to implement shift/reduce parsing.

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

int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
void main()
{

puts("GRAMMAR is\n E->E+E \n E->E*E \n E->(E) \n E->id");
puts("enter input string ");
gets(a);
c=strlen(a);
strcpy(act,"SHIFT->");
puts("stack \t input \t action");
for(k=0,i=0; j<c; k++,i++,j++)
{
if(a[j]=='i' && a[j+1]=='d')
{
stk[i]=a[j];
stk[i+1]=a[j+1];
stk[i+2]='\0';
a[j]=' ';
a[j+1]=' ';
printf("\n$%s\t%s$\t%sid",stk,a,act);
check();
}
else
{
stk[i]=a[j];
stk[i+1]='\0';
a[j]=' ';
printf("\n$%s\t%s$\t%ssymbols",stk,a,act);
check();
}
}
getch();
}
void check()
{
strcpy(ac,"REDUCE TO E");
for(z=0; z<c; z++)
if(stk[z]=='i' && stk[z+1]=='d')
{
stk[z]='E';
stk[z+1]='\0';
printf("\n$%s\t%s$\t%s",stk,a,ac);
j++;
}
for(z=0; z<c; z++)
if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
{
stk[z]='E';
stk[z+1]='\0';
stk[z+2]='\0';
printf("\n$%s\t%s$\t%s",stk,a,ac);
i=i-2;
}
```

```
for(z=0; z<c; z++)
if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
{
stk[z]='E';
stk[z+1]='\0';
stk[z+1]='\0';
printf("\n$%s\t%s$\t%s",stk,a,ac);
i=i-2;
}
for(z=0; z<c; z++)
if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
{
stk[z]='E';
stk[z+1]='\0';
stk[z+1]='\0';
printf("\n$%s\t%s$\t%s",stk,a,ac);
i=i-2;
}
}
```

Output:



```
GRAMMAR is
 E->E+E
 E->E*E
 E->(E)
 E->id
enter input string
(id+id)+(id*)
stack    input   action

$(       id+id)+(id*)$  SHIFT->symbols
$(id       +id)+(id*)$  SHIFT->id
$(E        +id)+(id*)$  REDUCE TO E
$(E+        id)+(id*)$  SHIFT->symbols
$(E+id        )+(id*)$  SHIFT->id
$(E+E         )+(id*)$  REDUCE TO E
$(E           )+(id*)$  REDUCE TO E
$(E)           +(id*)$  SHIFT->symbols
$E             +(id*)$  REDUCE TO E
$E+             (id*)$  SHIFT->symbols
$E+(             id*)$  SHIFT->symbols
$E+(id             *)$  SHIFT->id
$E+(E              *)$  REDUCE TO E
$E+(E*              )$  SHIFT->symbols
$E+(E*)              $  SHIFT->symbols
```