1.DFA implementation for (a+b)*.

A DFA (Deterministic Finite Automaton) for the regular expression (a+b)∗(a+b)^* needs to recognize any combination of 'a' and 'b', including the empty string.

Implementation:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
// Function to check if the string is accepted by the DFA
bool isAccepted(char *str) {
    for (int i = 0; i < strlen(str); i++) {
        if (str[i] != 'a' && str[i] != 'b') {
            return false;  // Reject if any other character is found } }
    return true;  // Accept if only 'a' and 'b' are present
int main() {
    char input[100];
    printf("Enter a string: ");
    scanf("%s", input);
    if (isAccepted(input)) {
        printf("Accepted: The string belongs to (a+b)*\n");
    } else {
        printf("Rejected: The string contains invalid characters.\n");  }
    return 0;}
```

Output:

Enter a string: abba

Accepted: The string belongs to (a+b)*

Enter a string: abc

Rejected: The string contains invalid characters**.**

2. DFA implementation for (a.b)*.

To construct a DFA for the regular expression (a.b)∗(a.b)^*, we need to recognize strings that contain zero or more repetitions of "a" followed by "b" (e.g., "", "ab", "abab", "ababab", etc.).

Implementation:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
// Function to check if the string is accepted by DFA
bool isAccepted(char *str) {
    int state = 0;  // Initial state q0
    for (int i = 0; i < strlen(str); i++) {
        if (state == 0 && str[i] == 'a') state = 1;
        else if (state == 1 && str[i] == 'b') state = 2;
        else if (state == 2 && str[i] == 'a') state = 1;
        else return false;  // Invalid transition, reject  }
    return (state == 0 || state == 2);  // Accepting states}
int main() {
    char input[100];
    printf("Enter a string: ");
    scanf("%s", input);
    if (isAccepted(input)) {
        printf("Accepted: The string belongs to (a.b)*\n");
    } else {
        printf("Rejected: The string does not match the pattern.\n");}
    return 0;}
```

Output:

Enter a string: abab

Accepted: The string belongs to (a.b)*

Enter a string: abba

Rejected: The string does not match the pattern.

3. DFA implementation for (a+b)*abb.

To construct a **DFA** for the regular expression (a+b)∗abb(a+b)^*abb, we need to recognize strings that consist of any combination of 'a' and 'b' followed by "abb" at the end.

Implementation:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
// Function to check if the string is accepted by DFA
bool isAccepted(char *str) {
    int state = 0;  // Initial state q0
    for (int i = 0; i < strlen(str); i++) {
        if (state == 0 && (str[i] == 'a' || str[i] == 'b')) state = 0;  // Loop in (a+b)*
        if (state == 0 && str[i] == 'a') state = 1;
        else if (state == 1 && str[i] == 'b') state = 2;
        else if (state == 2 && str[i] == 'b') state = 3;
        else return false;  // Invalid transition, reject }
    return (state == 3);  // Accept only if final state q3 is reached}
int main() {
    char input[100];
     printf("Enter a string: ");
    scanf("%s", input);
     if (isAccepted(input)) {
        printf("Accepted: The string belongs to (a+b)*abb\n");
    } else {
        printf("Rejected: The string does not match the pattern.\n");
    return 0;
}
```

Output:

Enter a string: aabb

Accepted: The string belongs to (a+b)*abb

Enter a string: aba

Rejected: The string does not match the pattern.

4. DFA implementation for (a.b)*abb.

To construct a **DFA** for the regular expression (a.b)∗abb(a.b)^*abb, we need to recognize strings that contain zero or more occurrences of "ab" followed by "abb" at the end.

Implementation:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
// Function to check if the string is accepted by DFA
bool isAccepted(char *str) {
    int state = 0;  // Initial state q0
    for (int i = 0; i < strlen(str); i++) {
        if (state == 0 && str[i] == 'a') state = 1;
        else if (state == 1 && str[i] == 'b') state = 2;
        else if (state == 2 && str[i] == 'a') state = 3;
        else if (state == 3 && str[i] == 'b') state = 4;
        else if (state == 4 && str[i] == 'b') state = 5;
        else return false;  // Invalid transition, reject }
    return (state == 5);  // Accept only if final state q5 is reached}
int main() {
    char input[100];
    printf("Enter a string: ");
    scanf("%s", input)
    if (isAccepted(input)) {
        printf("Accepted: The string belongs to (a.b)*abb\n");
    } else {
        printf("Rejected: The string does not match the pattern.\n"); }
    return 0;
}
```

Output:

Enter a string: abababb

Accepted: The string belongs to (a.b)*abb

Enter a string: aba

Rejected: The string does not match the pattern.

5.DFA implementation for even numbers of 1’s.

To construct a **DFA** for recognizing strings with an even number of 1’s, we need to track whether the count of 1’s in the input is **even or odd**.

Implementation:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
// Function to check if the string has an even number of 1's
bool isAccepted(char *str) {
    int state = 0;  // Initial state q0 (even number of 1's)
    for (int i = 0; i < strlen(str); i++) {
        if (str[i] == '1') state = (state == 0) ? 1 : 0;  // Toggle state on '1'
        else if (str[i] != '0') return false;  // Reject if character is not '0' or '1'}
     return (state == 0);  // Accept only if final state is q0}
int main() {
    char input[100];
    printf("Enter a binary string: ");
    scanf("%s", input);
     if (isAccepted(input)) {
        printf("Accepted: The string has an even number of 1’s.\n");
    } else {
        printf("Rejected: The string does not have an even number of 1’s.\n");}
    return 0;}
```

Output:

Enter a binary string: 1010

Accepted: The string has an even number of 1’s.

Enter a binary string: 1101

Rejected: The string does not have an even number of 1’s.

6. DFA implementation for odd numbers of 0’s.

to construct a **DFA** for recognizing strings with an odd number of 0’s, we need to track whether the count of 0’s is **odd or even**, while allowing any number of 1’s.

Implementation:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
// Function to check if the string has an odd number of 0's
bool isAccepted(char *str) {
    int state = 0;  // Initial state q0 (even number of 0's)
    for (int i = 0; i < strlen(str); i++) {
        if (str[i] == '0') state = (state == 0) ? 1 : 0;  // Toggle state on '0'
        else if (str[i] != '1') return false;  // Reject if character is not '0' or '1'}
     return (state == 1);  // Accept only if final state is q1 (odd number of 0's)}
int main() {
    char input[100];
      printf("Enter a binary string: ");
    scanf("%s", input);
     if (isAccepted(input)) {
        printf("Accepted: The string has an odd number of 0’s.\n");
    } else {
        printf("Rejected: The string does not have an odd number of 0’s.\n");
    }

    return 0;
}
```

Output:

Enter a binary string: 1001

Accepted: The string has an odd number of 0’s.

Enter a binary string: 10100

Rejected: The string does not have an odd number of 0’s.

7. DFA implementation for (a^n .b^n).

Implementation:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include<process.h>
Void main()
{
Char str [15]
Int i:
Printf(“this program is accepts the strings (a^n.b^n) over ‘a’ & ‘b’ “);
Ptintf(“\n enter a string : “ );
Gets(str);
For(i=0;i<strlen(str)/2;i++)
If (str[i]!=’a’|| str[i+strlen(str/2)]!=’b’)
{
Printf(‘string is not accepted’);
Exit(0);
} printf(“string is accepted”);
}
```

Output:

Enter a string: ab

String is accepted

Enter a string: aabb

String is accepted

Enter a string: aaabbb

String is accepted

Enter a string: aab

String is not accepted

Enter a string: abb

String is not accepted

Enter a string: aaabb

String is not accepted

8.implementation of lexical analyzer for some keyword- printf ,scanf, while, float.

A **lexical analyzer (lexer)** is responsible for **tokenizing** source code into meaningful tokens such as **keywords, identifiers, operators, numbers, and punctuation**. Below is a simple **C implementation of a lexical analyzer** that detects the keywords printf, scanf, while, and float.

Implementation:

```
#include<conio.h>
#include<string.h>
#include<process.h>
#include<ctype.h>
void main()
{
char str[10];
printf("This program identifies some keyword that are:\t");
printf("\nprintf\tscanf\tfloat\twhile");
printf("\n Press 'enter' after each keyword and when finish enter '$' \t");
while(1);
{gets(str);
if(str[0]=='$')
exit(1);
else if(str[0]=='p'&&str[1]=='r'&&str[3]=='n'&&str[4]=='t'&&str[5]=='f'&&isalpha
(str[6])==0)
 printf("\n You have entered 'printf'\t");
else if (str[0]=='s'&&str[1]=='c'&&str[2]=='a'&&str[3]=='n'&&str[4]=='f'&&isalpha
(str[5])==0)
printf("\n You have entered 'scanf'\t");
 else if (str[0]=='f'&&str[1]=='I'&&str[2]=='o'&&str[3]=='a'&&str[4]=='t'&&isalpha
(str[5])==0)
printf("\n You have entered 'float'\t");
else if (str[0]=='w'&&str[1]=='h'&&str[2]=='i'&&str[3]=='I'&&str[4]=='e'&&isalpha
(str[5])==0)
printf("\n You have entered 'while'\t");
else
printf("\n You have entered wrong keyword\t");
}}
```

Output:

This program identifies some keyword that are:

printf scanf float while

Press ‘enter’ after each keyword and when finish enter ‘$’ printf

You have entered ‘printf’ scanf

You have entered ‘scanf’ sdef

You have entered wrong keyword float

You have entered ‘float’ $

9.implementation of lexical analyzer fo all identifiers.

A **lexical analyzer (lexer)** scans source code and identifies different tokens, including **identifiers**, keywords, operators, numbers, and symbols. Below is a **C implementation** of a simple lexical analyzer that detects **identifiers**.

Implementation:

```
#include<conio.h>
#include<string.h>
#include<process.h>
#include<ctype.h>
void main(){
char str[10];
int i ;
printf("This program identifies all the identifiers\t");
printf("\n \n Press 'enter' after each identifier and when finish enter '$' \t");
while(1);
{
gets(str);
if(str[0]=='$')
exit(1);
if(isdigit(str[0]) || ispunct(str[0]))
 printf("Illegal identifier\t");
for(i=0;i<strlen(str);)
 if(isalnum(str[i]) && islower(str[i]))
 i++;
if(i==strlen(str))
 printf("Legal identifier\t");
else
 printf("Illegal identifier\t"); }
```

Output:

This program identifies all the identifiers

Press ‘enter’ after each identifier and when finish enter ‘$’ dfrg

Legal identifier 2dfr

Illegal identifier (sde

Illegal identifier Asde

Illegal identifier $

Index