

1, write a C program to simulate a deterministic finite automata [DFA] for the given language.

The screenshot shows a C program in Dev-C++ for simulating a Deterministic Finite Automata (DFA). The code is in a file named `DFA.c` and is being compiled with TDM-GCC 4.9.2 64-bit Release. The program takes an input string and checks if it is accepted by the DFA. The output window shows the program's execution for the input string "abaaab", which is accepted. The compilation results show 0 errors and 0 warnings, with the output file named `DFA.exe`.

```

14 int l=strlen(input_string);
15 for(i=0;i<l;i++)
16 {
17     if(input_string[i]!='a')
18         next_state=trans_table[present_state][0];
19     else if(input_string[i]!='b')
20         next_state=trans_table[present_state][1];
21     else
22         invalid=1;
23     present_state=next_state;
24 }
25 if(invalid==1)
26 {
27     printf("Invalid input");
28 }
29 else if(present_state==final_state)
30     printf("Accept\n");
31 else
32     printf("Don't Accept\n");
33 }

```

Compilation results...

```

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\hp\OneDrive\Documents\DFA.exe
- Output Size: 128.2705078125 KiB
- Compilation Time: 1.00s

```

Line: 19 Col: 30 Sel: 0 Lines: 33 Length: 662 Insert: Done parsing in 0.031 seconds

2, write a C program to simulate a non deterministic finite automata [NFA] for the given language.

The screenshot shows a C program in Dev-C++ for simulating a Non-deterministic Finite Automata (NFA). The code is in a file named `NFA.c` and is being compiled with TDM-GCC 4.9.2 64-bit Release. The program takes the number of states and symbols in the input alphabet as input, and then checks if the input string is accepted by the NFA. The output window shows the program's execution for 3 states and 3 symbols, with the input string "abaaab" being accepted. The compilation results show 0 errors and 0 warnings, with the output file named `NFA.exe`.

```

106 flag=0;
107 for(i=0;i<prev_trans;i++)
108 {
109     for(j=0;j<num_final;j++)
110     {
111         if(present_state[i]==final_state[j])
112             flag=1;
113         break;
114     }
115 }
116 if(flag==1)
117     printf("Accepted\n");
118 else
119     printf("Not accepted\n");
120 printf("Try with another input\n");
121 }
122 }
123 }
124 }
125 }

```

Compilation results...

```

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\hp\OneDrive\Documents\NFA.exe
- Output Size: 128.2705078125 KiB
- Compilation Time: 1.00s

```

Line: 125 Col: 1 Sel: 0 Lines: 125 Length: 2231 Insert: Done parsing in 0 seconds

3, write a C program to find e-closure for all the state in a non- deterministic finite automata [NFA] with e-moves.

```

69 | printf("\n");
70 | }
71 |
72 | void find_e_closure(int x)
73 | {
74 |     int i,j,y[10],num_trans;
75 |     i=0;
76 |     while(trans_table[x][0][i]!=-1)
77 |     {
78 |         y[i]=trans_table[x][0][i];
79 |         i=i+1;
80 |     }
81 |     num_trans=i;
82 |     for(j=0;j<num_trans;j++)
83 |     {
84 |         e_closure[state][ptr]=y[j];
85 |         ptr++;
86 |         find_e_closure(y[j]);
87 |     }
88 | }

```

```

How may states in the NFA with e-moves:3
How many symbols in the input alphabet including e :3
Enter the symbols without space. Give 'e' first:01
How many transitions from state 0 for the input 0:1
Enter the transitions 1 from state 0 for the input 0 :1
How many transitions from state 0 for the input 1:0
How many transitions from state 0 for the input :1
Enter the transitions 1 from state 0 for the input :1
How many transitions from state 1 for the input 0:1
Enter the transitions 1 from state 1 for the input 0 :2
How many transitions from state 1 for the input 1:2
Enter the transitions 1 from state 1 for the input 1 :0
Enter the transitions 2 from state 1 for the input 1 :1
How many transitions from state 1 for the input :0
How many transitions from state 2 for the input 0:0
How many transitions from state 2 for the input 1:0
How many transitions from state 2 for the input :0
e-closure(0)= {0, 1, 2, }
e-closure(1)= {1, 2, }
e-closure(2)= {2, }

-----
Process exited after 50.31 seconds with return value 3
Press any key to continue . . .

```

4. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG).

```

1 | #include<stdio.h>
2 | #include<string.h>
3 | int main(){
4 |     char s[100];
5 |     int i,flag;
6 |     int l;
7 |     printf("enter a string to check:");
8 |     scanf("%s",s);
9 |     l=strlen(s);
10 |    flag=1;
11 |    for(i=0;i<l;i++)
12 |    {
13 |        if(s[i]!='0' && s[i]!='1')
14 |        {
15 |            flag=0;
16 |        }
17 |    }
18 |    if(flag!=1)
19 |        printf("string is Not Valid\n");
20 |    if(flag==1)
21 |        printf("string is Valid\n");

```

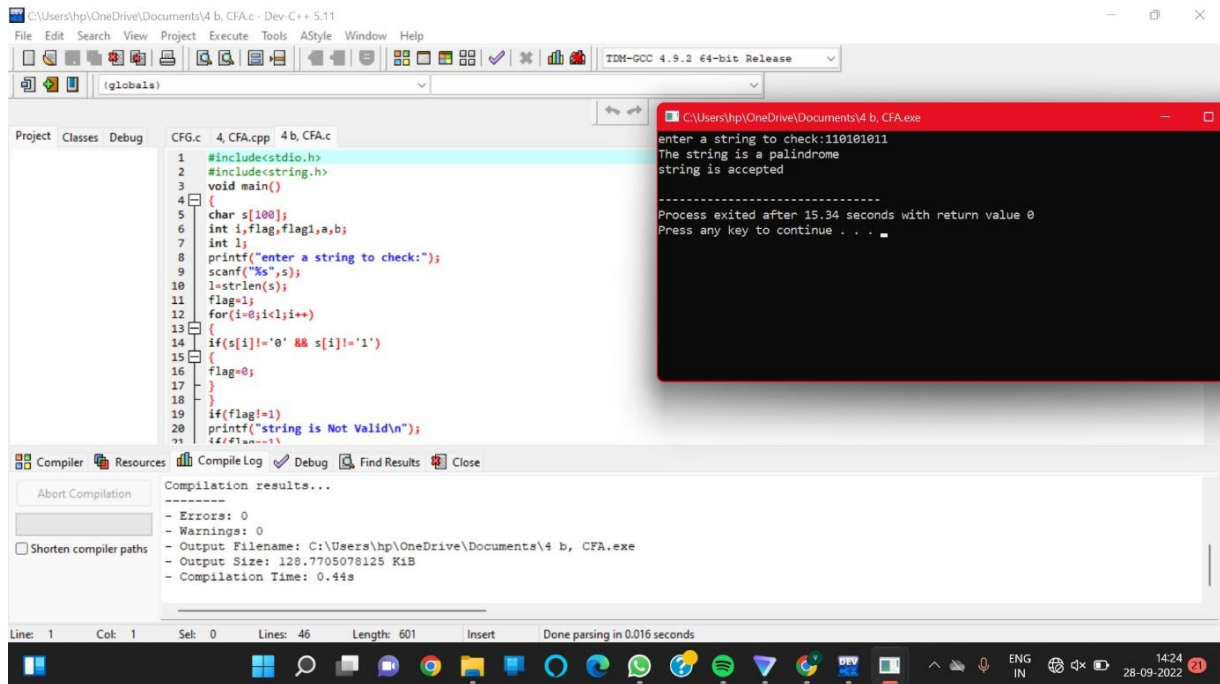
```

enter a string to check:0101011101
string is accepted

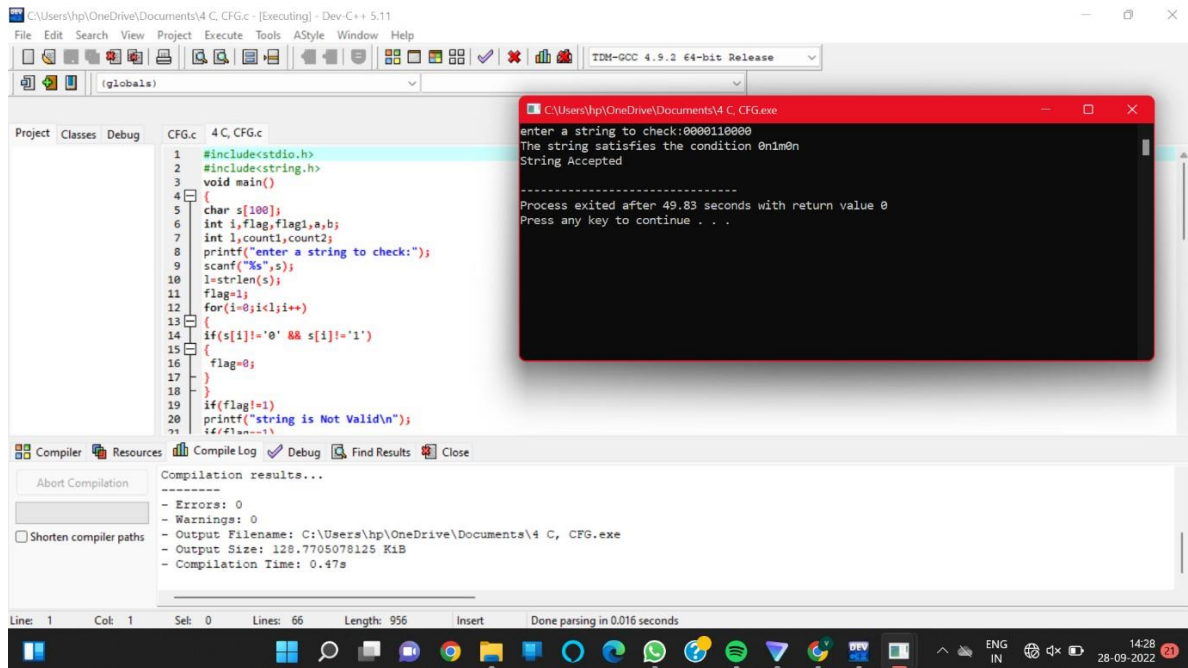
-----
Process exited after 119.1 seconds with return value 0
Press any key to continue . . .

```

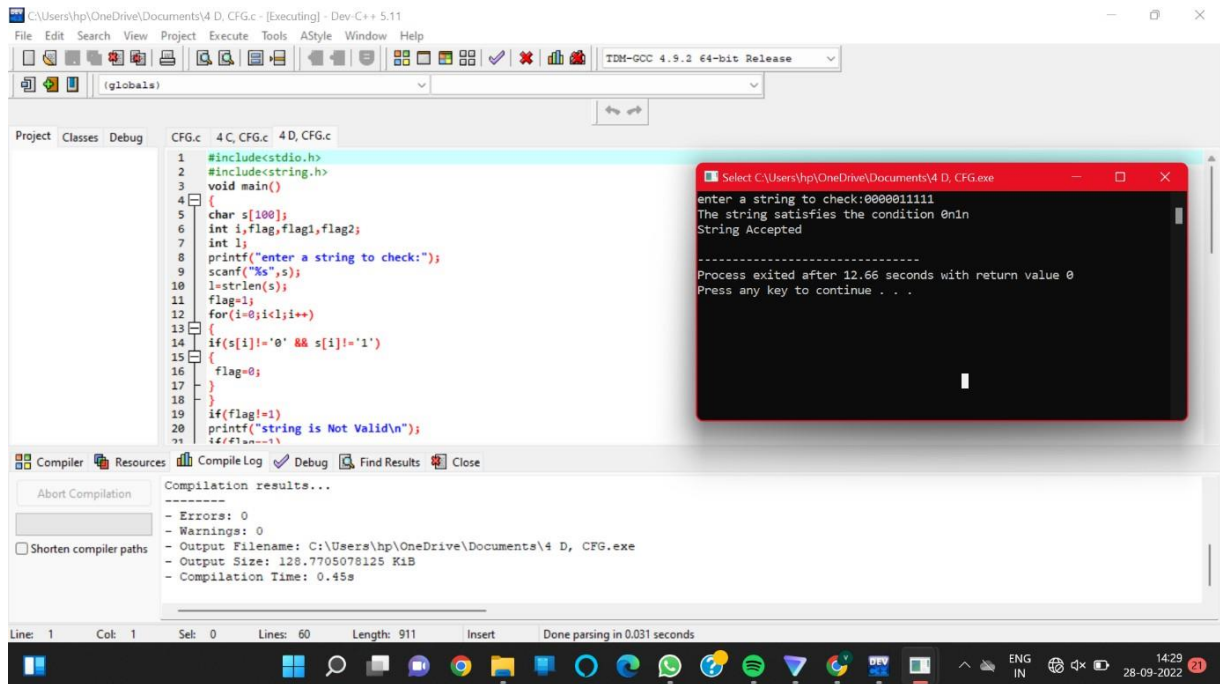
B,



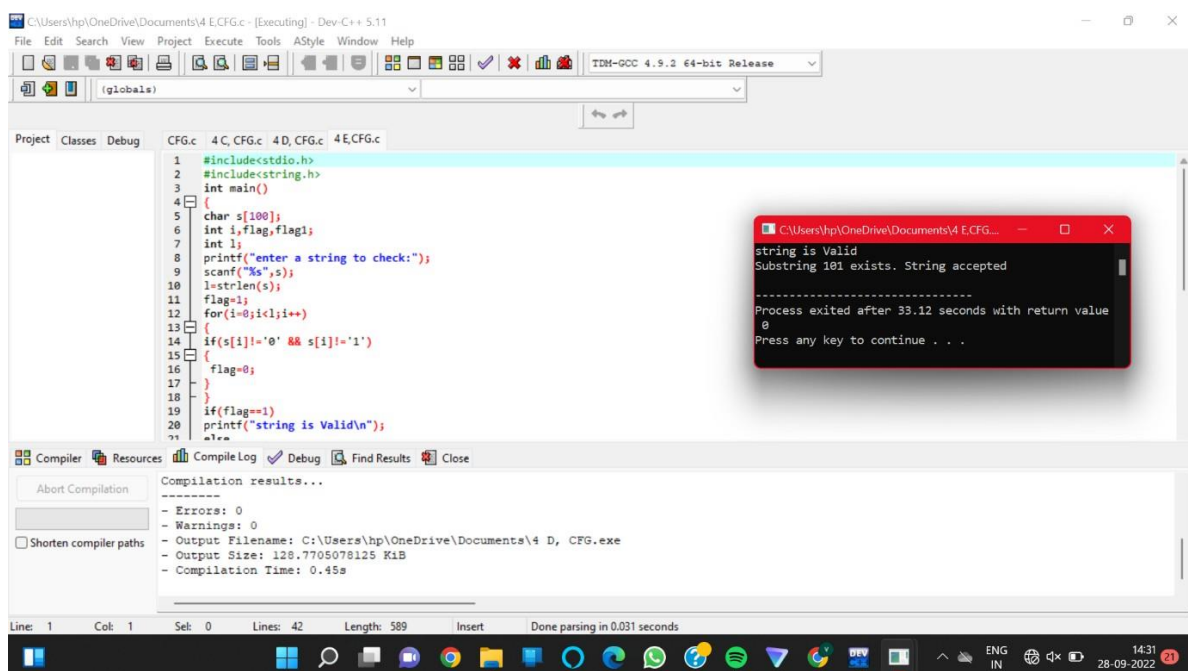
C,



D,



E,



5. Write a C program to simulate a Push Down Automata (PDA) for the language 0^n1^n .

The screenshot shows a C program in Dev-C++ 5.11. The program simulates a Push Down Automata (PDA) for the language 0^n1^n . The code is as follows:

```

1 #include<stdio.h>
2 #include<string.h>
3 char stack[20];
4 int top;
5 void push()
6 {
7     top=top+1;
8     stack[top]='0';
9     stack[top+1]='\0';
10 }
11 int pop()
12 {
13     if(top<1)
14         return(0);
15     else
16     {
17         stack[top]='\0';
18         top=top-1;
19         return(1);
20     }
21 }

```

The console window shows the simulation results for the input string "000011111". The stack is initially empty (Z). The simulation pushes four '0's onto the stack. Then, it pops four '0's from the stack. The string is accepted, and the process exits after 9.59 seconds with a return value of 13.

6. Write a C program to simulate a Push Down Automata (PDA) for the language $anb2n$

The screenshot shows a C program in Dev-C++ 5.11. The program simulates a Push Down Automata (PDA) for the language $anb2n$. The code is as follows:

```

1 #include<stdio.h>
2 #include<string.h>
3 char stack[20];
4 int top, count=0;
5 void push()
6 {
7     top=top+1;
8     stack[top]='0';
9     stack[top+1]='\0';
10 }
11 int pop()
12 {
13     if(top<1)
14         return(0);
15     else
16     {
17         stack[top]='\0';
18         top=top-1;
19         return(1);
20     }
21 }

```

The console window shows the simulation results for the input string "000011111111". The stack is initially empty (Z). The simulation pushes four '0's onto the stack. Then, it pops four '0's from the stack. The string is accepted, and the process exits after 20.47 seconds with a return value of 13.

7. Write a C program to simulate a Turing Machine (TM) for the given language.

