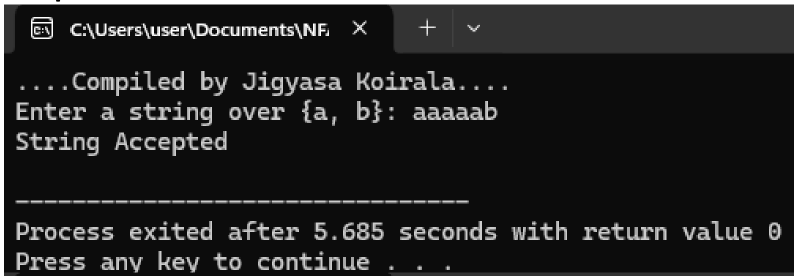
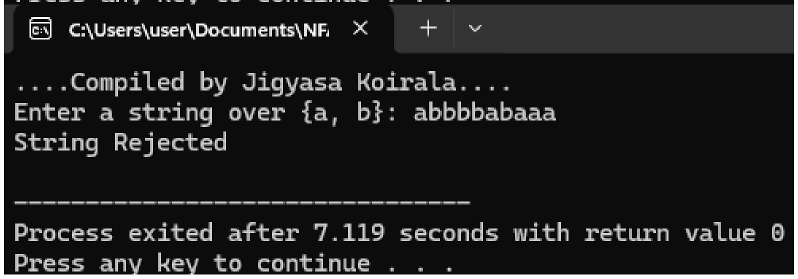
1. C program to simulate Nondeterministic Finite Automata (NFA ending with ab).

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

int main() {
    char input[100];
    int current[10], next[10];
    int nCurrent, nNext;

    // Define accepting state
    int finalState = 2;

    printf("....Compiled by Jigyasa Koirala....\n");
    printf("Enter a string over {a, b}: ");
    scanf("%s", input);

    // Start state
    current[0] = 0;
    nCurrent = 1;

    for (int i = 0; i < strlen(input); i++) {
        char ch = input[i];
        nNext = 0;

        for (int j = 0; j < nCurrent; j++) {
            int st = current[j];

            // NFA transitions
            if (st == 0 && ch == 'a') {
                next[nNext++] = 0;  // loop
                next[nNext++] = 1;  // go to 1
            }
            if (st == 1 && ch == 'b') {
                next[nNext++] = 2;  // final
            }
        }

        // copy next ? current
        for (int k = 0; k < nNext; k++)
            current[k] = next[k];

        nCurrent = nNext;
    }
    // Check if any current state is final
    int accepted = 0;
    for (int i = 0; i < nCurrent; i++) {
        if (current[i] == finalState)
            accepted = 1;
    }

    if (accepted)
        printf("String Accepted\n");
    else
        printf("String Rejected\n");

    return 0;
}
```

Output:



```
....Compiled by Jigyasa Koirala....
Enter a string over {a, b}: aaaaab
String Accepted

--------------------------------
Process exited after 5.685 seconds with return value 0
Press any key to continue . . .
```



```
....Compiled by Jigyasa Koirala....
Enter a string over {a, b}: abbbbabaaa
String Rejected

--------------------------------
Process exited after 7.119 seconds with return value 0
Press any key to continue . . .
```

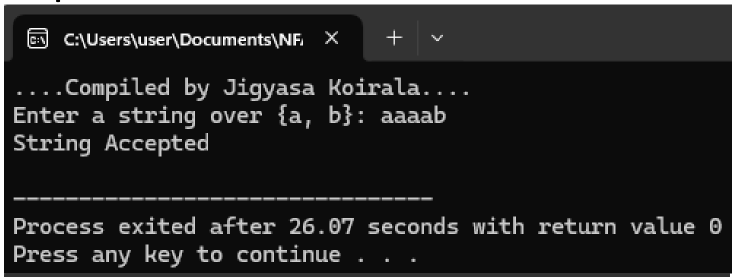
2. Program to implement NFA from Regular Expression

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

// NFA for regex a*b
// States:
// 0 = start
// 1 = loop for a
// 2 = final (after b)

int main() {
```

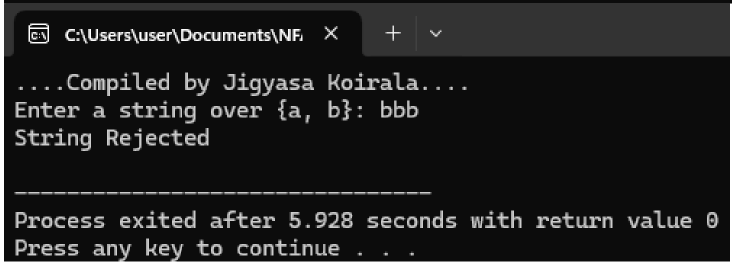
```
char input[100];
int current[10], next[10];
int nCurrent, nNext;

printf("....Compiled by Jigyasa Koirala....\n");
printf("Enter a string over {a, b}: ");
scanf("%s", input);

// start state
current[0] = 0;
nCurrent = 1;

int finalState = 2;

for (int i = 0; i < strlen(input); i++) {
   char ch = input[i];
   nNext = 0;

   for (int j = 0; j < nCurrent; j++) {
      int st = current[j];

      // transitions for regex a*b

      // from state 0 ? on 'a' ? stay in 0
      if (st == 0 && ch == 'a')
         next[nNext++] = 0;

      // from state 0 ? on 'b' ? go to final state 2
      if (st == 0 && ch == 'b')
         next[nNext++] = 2;

      // from final state, no more transitions
   }

   // update current states
   for (int k = 0; k < nNext; k++)
      current[k] = next[k];

   nCurrent = nNext;

   if (nCurrent == 0)
      break;  // dead end
}
```

```
    // check acceptance
    int accepted = 0;
    for (int i = 0; i < nCurrent; i++) {
        if (current[i] == finalState)
            accepted = 1;
    }

    if (accepted)
        printf("String Accepted\n");
    else
        printf("String Rejected\n");

    return 0;
}
```

Output:



```
....Compiled by Jigyasa Koirala....
Enter a string over {a, b}: aaaab
String Accepted

--------------------------------
Process exited after 26.07 seconds with return value 0
Press any key to continue . . .
```

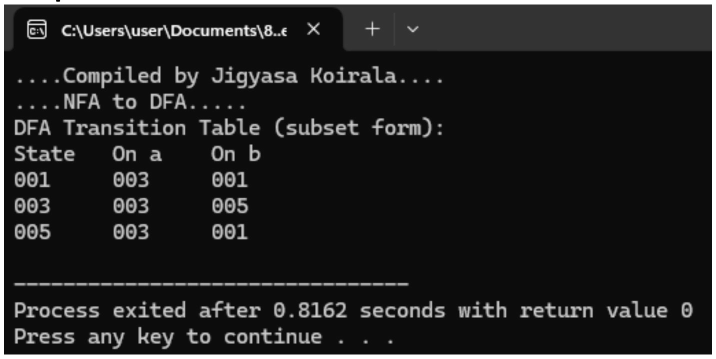


```
....Compiled by Jigyasa Koirala....
Enter a string over {a, b}: bbb
String Rejected

--------------------------------
Process exited after 5.928 seconds with return value 0
Press any key to continue . . .
```

3. A simple and basic program in C to convert NFA to DFA (does not handle null moves)

```
#include <stdio.h>

int main() {
    // NFA transition table (bitmasks)
    // nfa[state][symbol] = bitmask of next states
    // symbol: 0 = 'a', 1 = 'b'

    int nfa[3][2] = {
        { 1<<0 | 1<<1, 1<<0 },   // state 0
        { 0,       1<<2 },       // state 1
    { 0,      0 }          // state 2
};

// DFA will have at most 2^3 = 8 states
int dfa[8][2] = {0};
int used[8] = {0};

int queue[8], front = 0, rear = 0;

// start DFA state = {0} ? bitmask = 001 = 1
queue[rear++] = 1;
used[1] = 1;

while (front < rear) {
    int set = queue[front++];

    // for each symbol a (0) and b (1)
    for (int sym = 0; sym < 2; sym++) {

        int nextSet = 0;

        // check each NFA state
        for (int s = 0; s < 3; s++) {
            if (set & (1<<s)) {       // if s in current set
                nextSet |= nfa[s][sym];
            }
        }

        dfa[set][sym] = nextSet;

        // if new subset, add to queue
        if (!used[nextSet]) {
            used[nextSet] = 1;
            queue[rear++] = nextSet;
        }
    }
}

// print result
printf("....Compiled by Jigyasa Koirala....\n");
printf("....NFA to DFA.....\n");
printf("DFA Transition Table (subset form):\n");
printf("State\tOn a\tOn b\n");
for (int i = 0; i < 8; i++) {
        if (used[i]) {
            printf("%03d\t%03d\t%03d\n", i, dfa[i][0], dfa[i][1]);
        }
    }

    return 0;
}
```

Output:



```
....Compiled by Jigyasa Koirala....
....NFA to DFA.....
DFA Transition Table (subset form):
State   On a    On b
001     003     001
003     003     005
005     003     001

--------------------------------
Process exited after 0.8162 seconds with return value 0
Press any key to continue . . .
```

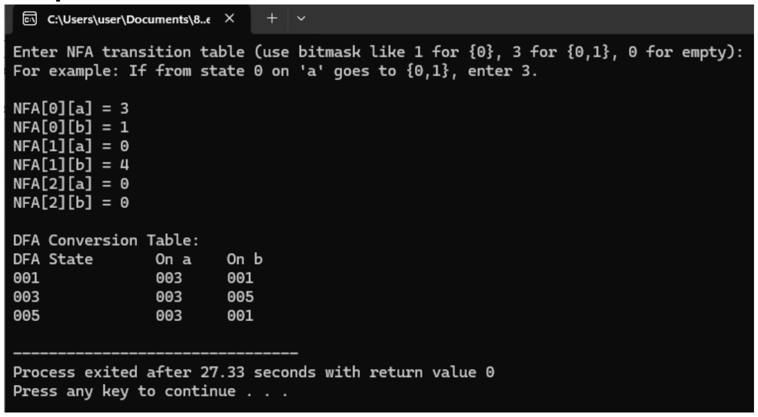
NOTE :-
1. If your states are q0, q1, q2 they will be represented as follows (in the table)
q0 = 2^0 = 1
q1 = 2^1 = 2
q2 = 2^2 = 4
2. Similarly union of states will be represented as -
q0,q1 = 2^0 + 2^1 = 3
q1, q2 = 2^1 + 2^2 = 6
q0,q1,q2 = 2^0 + 2^1 + 2^2 = 7
3. Do not give any condition for "phi"...
That case is not handled... (Coz I m Lazy :P)
4. Follow zero based indexing everywhere
5. Program assumes that if "Number of states are = n", then they are numbered as q0, q1, q2 ... q(n-1)
6. If you find any bug, msg me and forgive me for the errors

4. A program to convert NFA to DFA using conversion table.

```
#include <stdio.h>
#define STATES 3      // number of NFA states
#define ALPHABET 2    // a, b
#define MAX_DFA 8     // 2^3 possible subsets
```

```
int main() {
    int nfa[STATES][ALPHABET];
    int dfa[MAX_DFA][ALPHABET] = {0};
    int used[MAX_DFA] = {0};
    int queue[MAX_DFA];
    int front = 0, rear = 0;

    printf("Enter NFA transition table (use bitmask like 1 for {0}, 3 for {0,1}, 0 for empty):\n");
    printf("For example: If from state 0 on 'a' goes to {0,1}, enter 3.\n\n");

    // Input NFA transitions
    for (int s = 0; s < STATES; s++) {
        for (int a = 0; a < ALPHABET; a++) {
            printf("NFA[%d][%c] = ", s, 'a' + a);
            scanf("%d", &nfa[s][a]);
        }
    }

    // Start DFA state = {0} = 1<<0 = 1
    int start = 1;
    queue[rear++] = start;
    used[start] = 1;

    // Subset Construction
    while (front < rear) {
        int set = queue[front++];

        // For each input symbol (a, b)
        for (int sym = 0; sym < ALPHABET; sym++) {
            int nextSet = 0;

            // For each NFA state
            for (int s = 0; s < STATES; s++) {
                if (set & (1 << s)) {
                    nextSet |= nfa[s][sym];
                }
            }

            dfa[set][sym] = nextSet;

            if (!used[nextSet]) {
                used[nextSet] = 1;
                queue[rear++] = nextSet;
            }
        }
    }

    // Print DFA Conversion Table
    printf("\nDFA Conversion Table:\n");
    printf("DFA State\tOn a\tOn b\n");

    for (int s = 0; s < MAX_DFA; s++) {
        if (used[s]) {
            printf("%03d\t\t%03d\t%03d\n", s, dfa[s][0], dfa[s][1]);
        }
    }

    return 0;
}
```

Output:



```
C:\Users\user\Documents\8.e

Enter NFA transition table (use bitmask like 1 for {0}, 3 for {0,1}, 0 for empty):
For example: If from state 0 on 'a' goes to {0,1}, enter 3.

NFA[0][a] = 3
NFA[0][b] = 1
NFA[1][a] = 0
NFA[1][b] = 4
NFA[2][a] = 0
NFA[2][b] = 0

DFA Conversion Table:
DFA State       On a    On b
001             003     001
003             003     005
005             003     001

--------------------------------
Process exited after 27.33 seconds with return value 0
Press any key to continue . . .
```