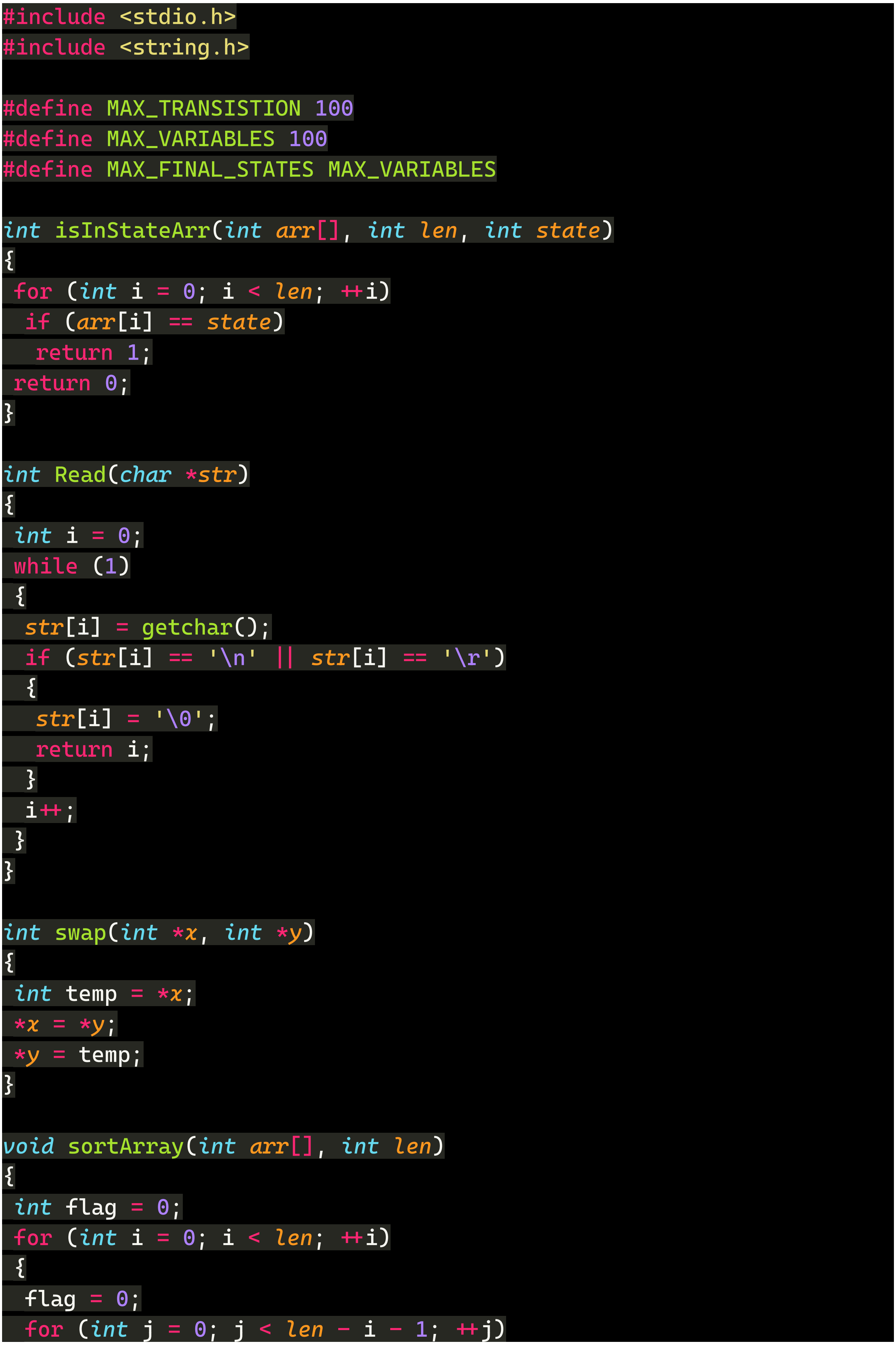
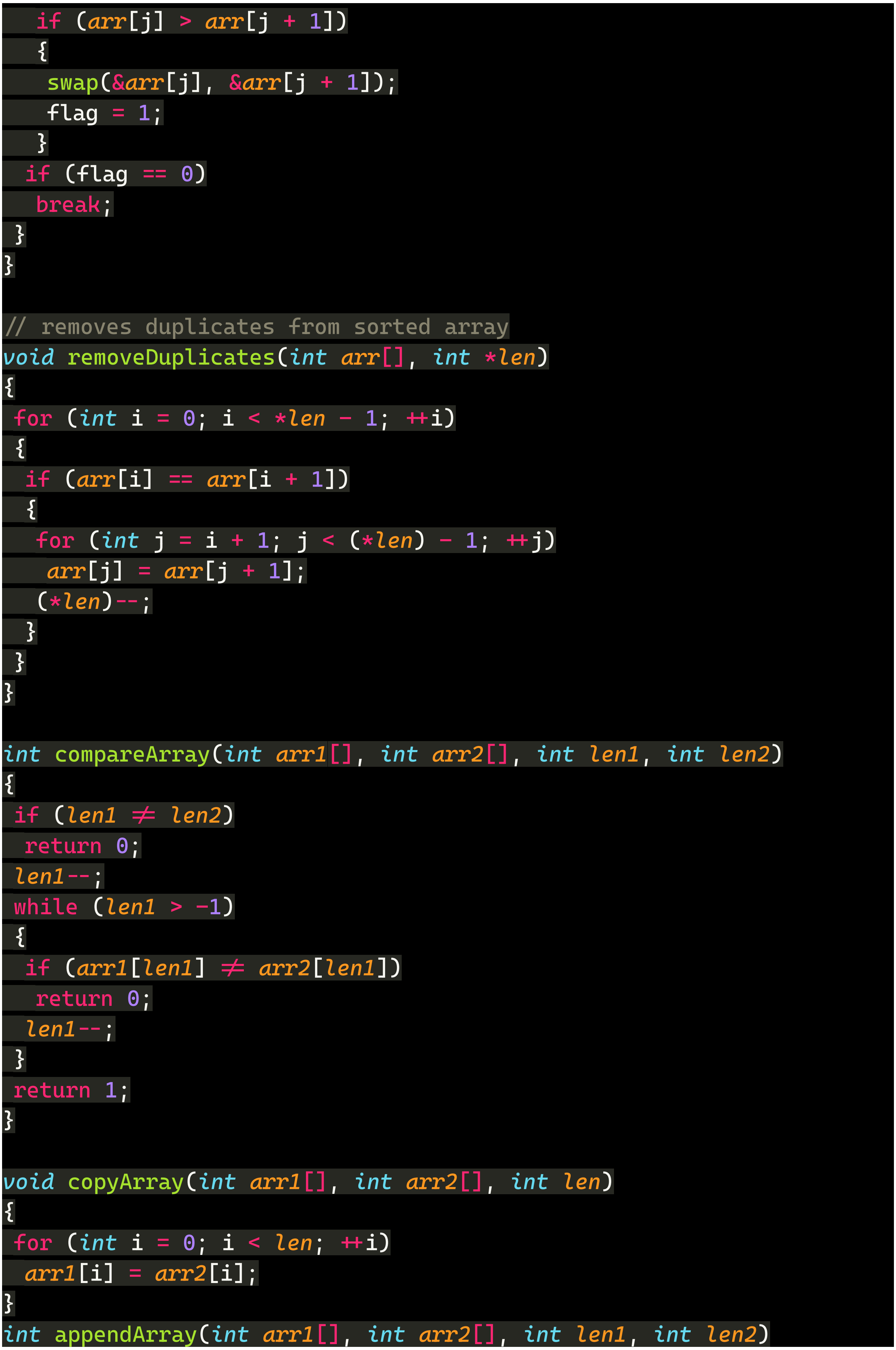
Program 4:

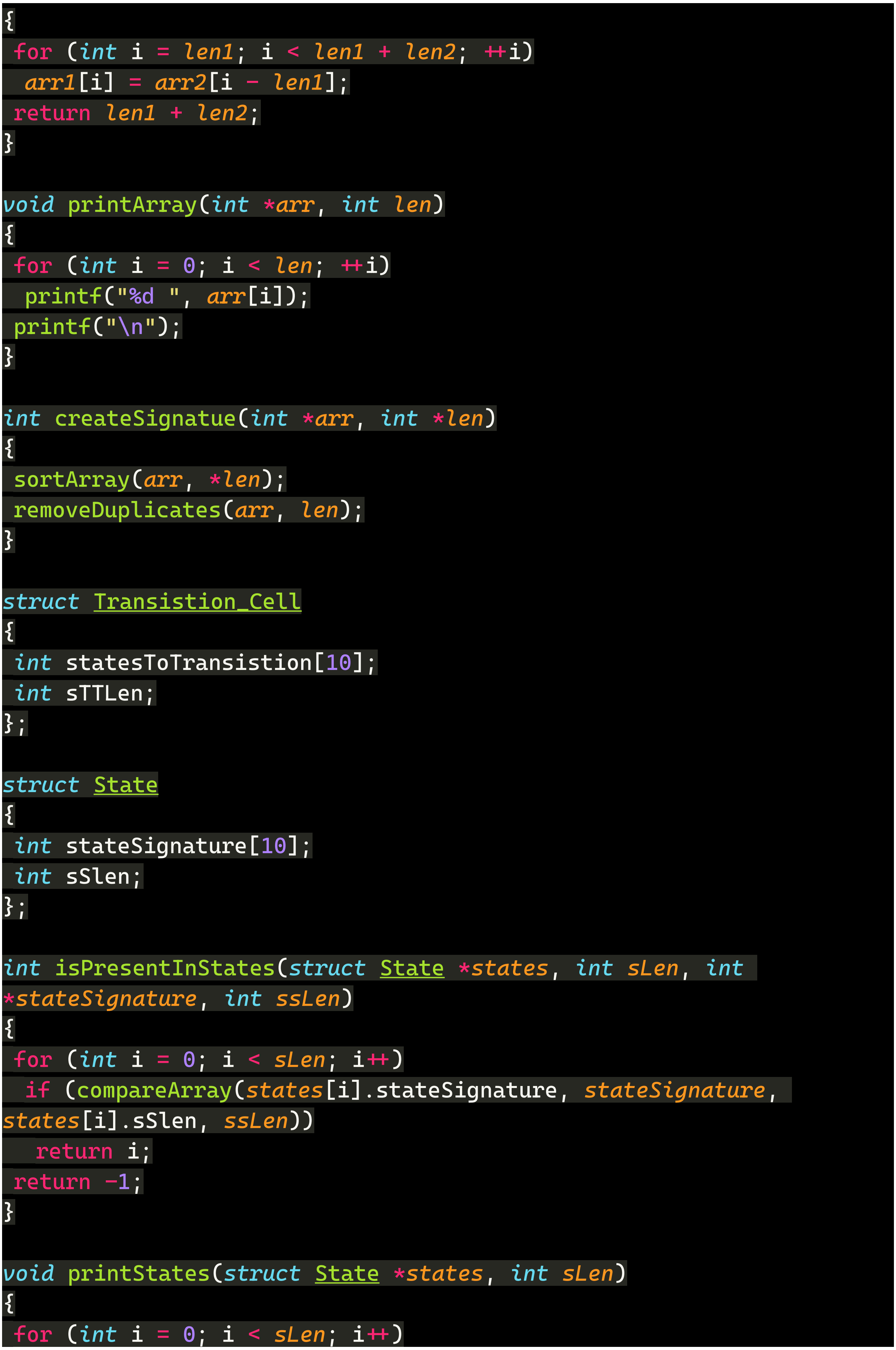
WAP to implement the conversion of a NFA to a corresponding DFA. The NFA must be given through a separate file.



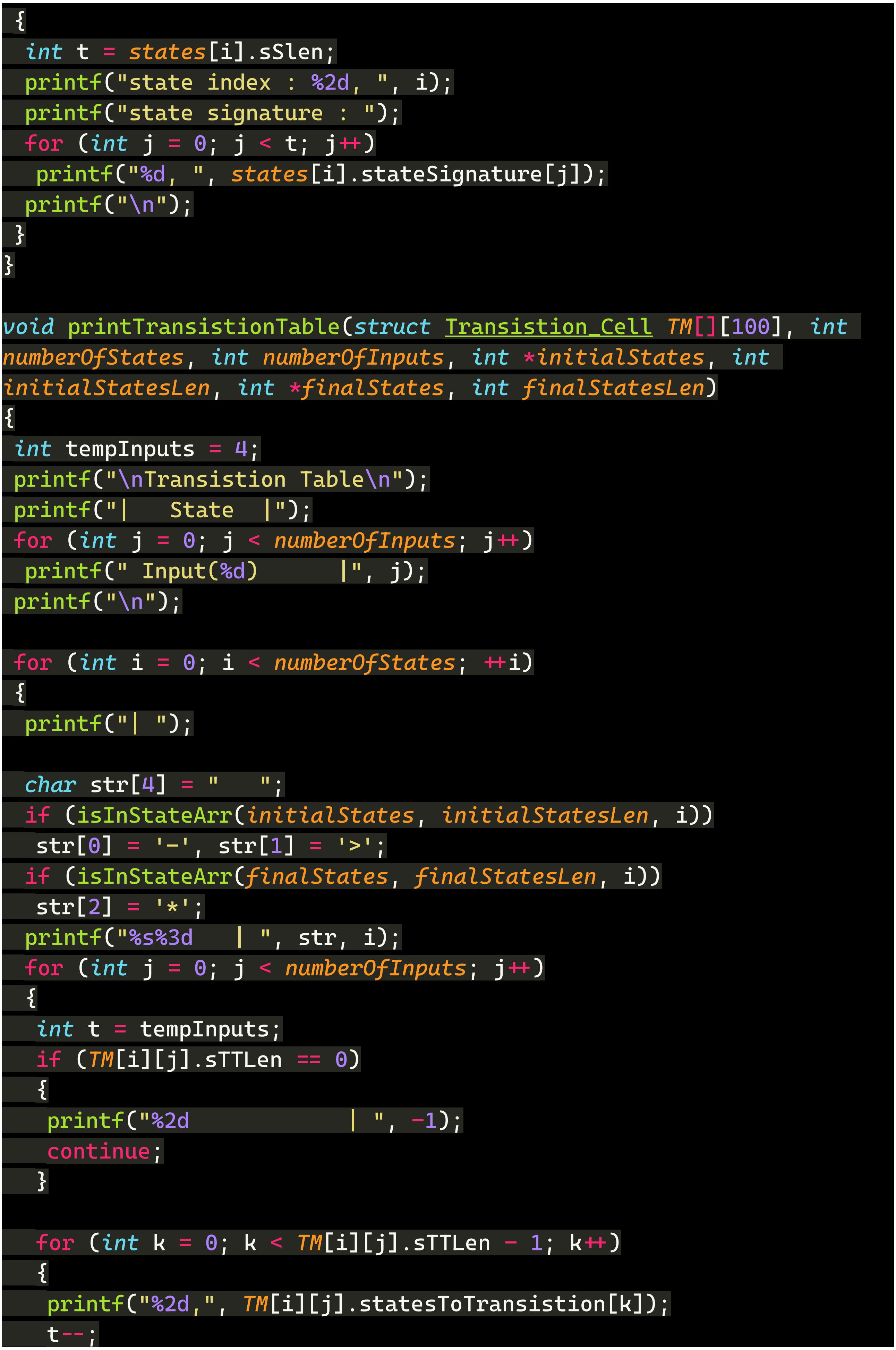
```c
#include <stdio.h>
#include <string.h>

#define MAX_TRANSISTION 100
#define MAX_VARIABLES 100
#define MAX_FINAL_STATES MAX_VARIABLES

int isInStateArr(int arr[], int len, int state)
{
 for (int i = 0; i < len; ++i)
  if (arr[i] == state)
   return 1;
 return 0;
}

int Read(char *str)
{
 int i = 0;
 while (1)
 {
  str[i] = getchar();
  if (str[i] == '\n' || str[i] == '\r')
  {
   str[i] = '\0';
   return i;
  }
  i++;
 }
}

int swap(int *x, int *y)
{
 int temp = *x;
 *x = *y;
 *y = temp;
}

void sortArray(int arr[], int len)
{
 int flag = 0;
 for (int i = 0; i < len; ++i)
 {
  flag = 0;
  for (int j = 0; j < len - i - 1; ++j)
```



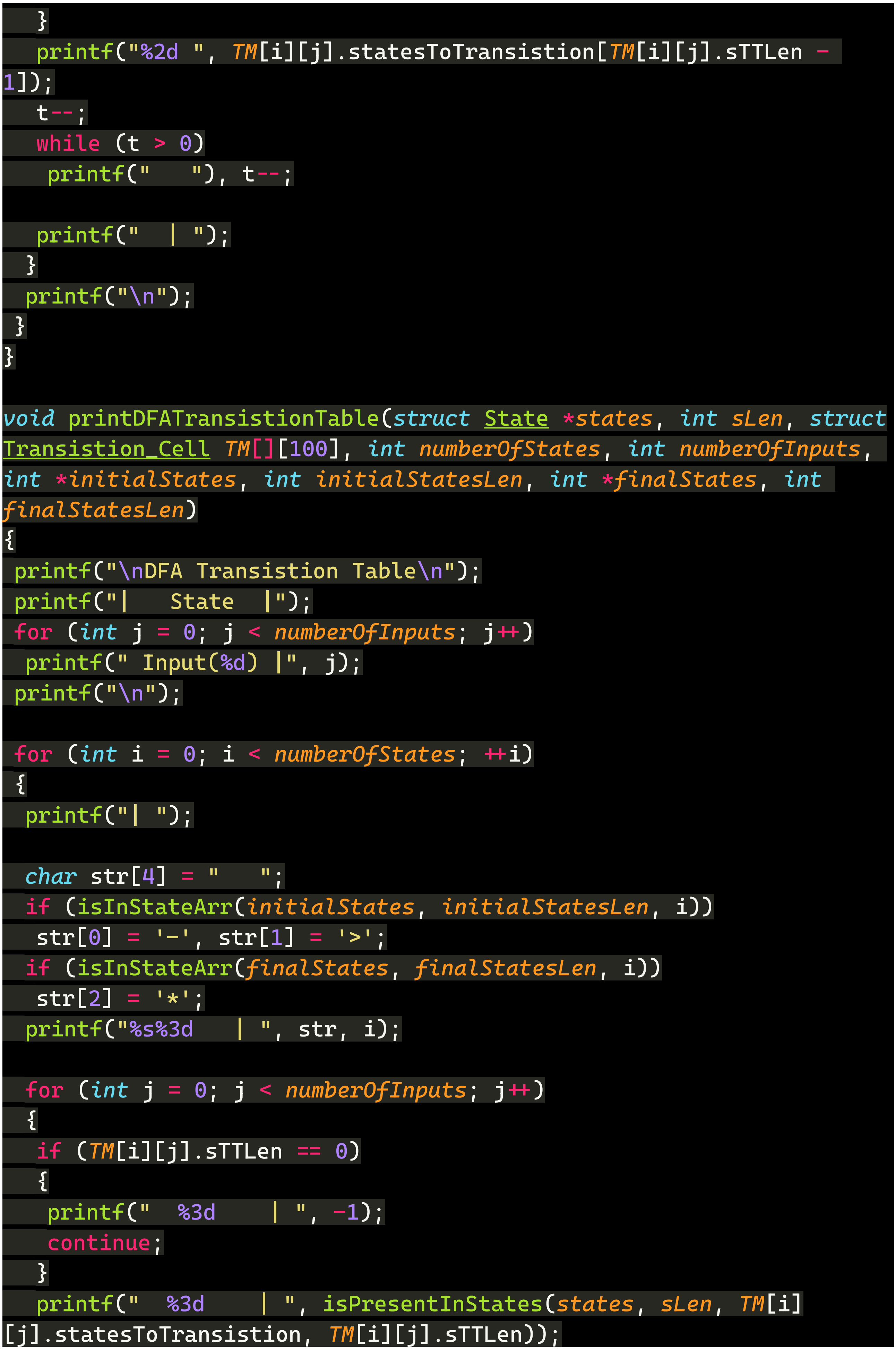
```
   if (arr[j] > arr[j + 1])
   {
    swap(&arr[j], &arr[j + 1]);
    flag = 1;
   }
  if (flag == 0)
   break;
 }
}

// removes duplicates from sorted array
void removeDuplicates(int arr[], int *len)
{
 for (int i = 0; i < *len - 1; ++i)
 {
  if (arr[i] == arr[i + 1])
  {
   for (int j = i + 1; j < (*len) - 1; ++j)
    arr[j] = arr[j + 1];
   (*len)--;
  }
 }
}

int compareArray(int arr1[], int arr2[], int len1, int len2)
{
 if (len1 != len2)
  return 0;
 len1--;
 while (len1 > -1)
 {
  if (arr1[len1] != arr2[len1])
   return 0;
  len1--;
 }
 return 1;
}

void copyArray(int arr1[], int arr2[], int len)
{
 for (int i = 0; i < len; ++i)
  arr1[i] = arr2[i];
}
int appendArray(int arr1[], int arr2[], int len1, int len2)
```



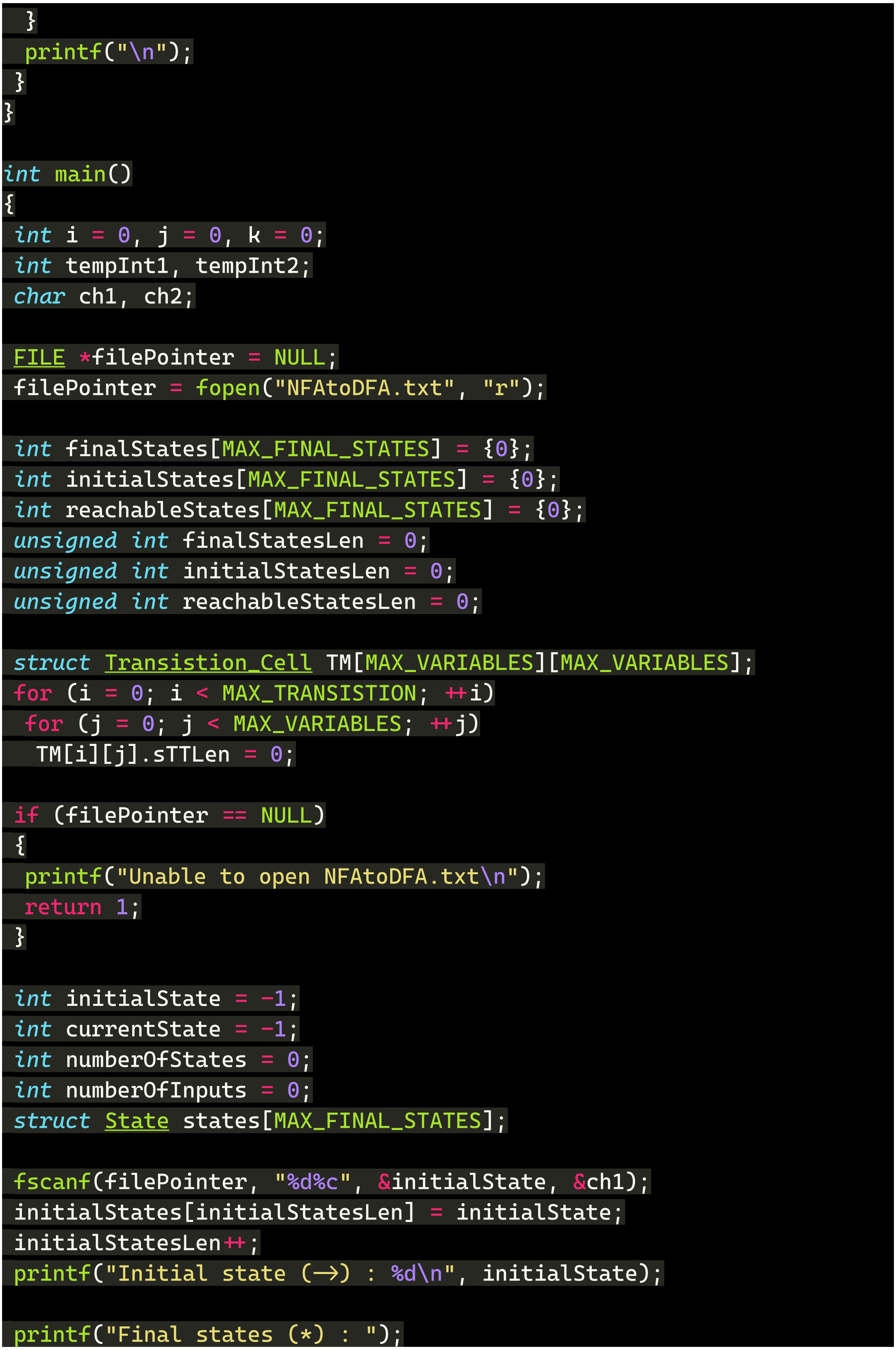
```
{
 for (int i = len1; i < len1 + len2; ++i)
  arr1[i] = arr2[i - len1];
 return len1 + len2;
}

void printArray(int *arr, int len)
{
 for (int i = 0; i < len; ++i)
  printf("%d ", arr[i]);
 printf("\n");
}

int createSignatue(int *arr, int *len)
{
 sortArray(arr, *len);
 removeDuplicates(arr, len);
}

struct Transistion_Cell
{
 int statesToTransistion[10];
 int sTTLen;
};

struct State
{
 int stateSignature[10];
 int sSlen;
};

int isPresentInStates(struct State *states, int sLen, int 
*stateSignature, int ssLen)
{
 for (int i = 0; i < sLen; i++)
  if (compareArray(states[i].stateSignature, stateSignature, 
states[i].sSlen, ssLen))
   return i;
 return -1;
}

void printStates(struct State *states, int sLen)
{
 for (int i = 0; i < sLen; i++)
```



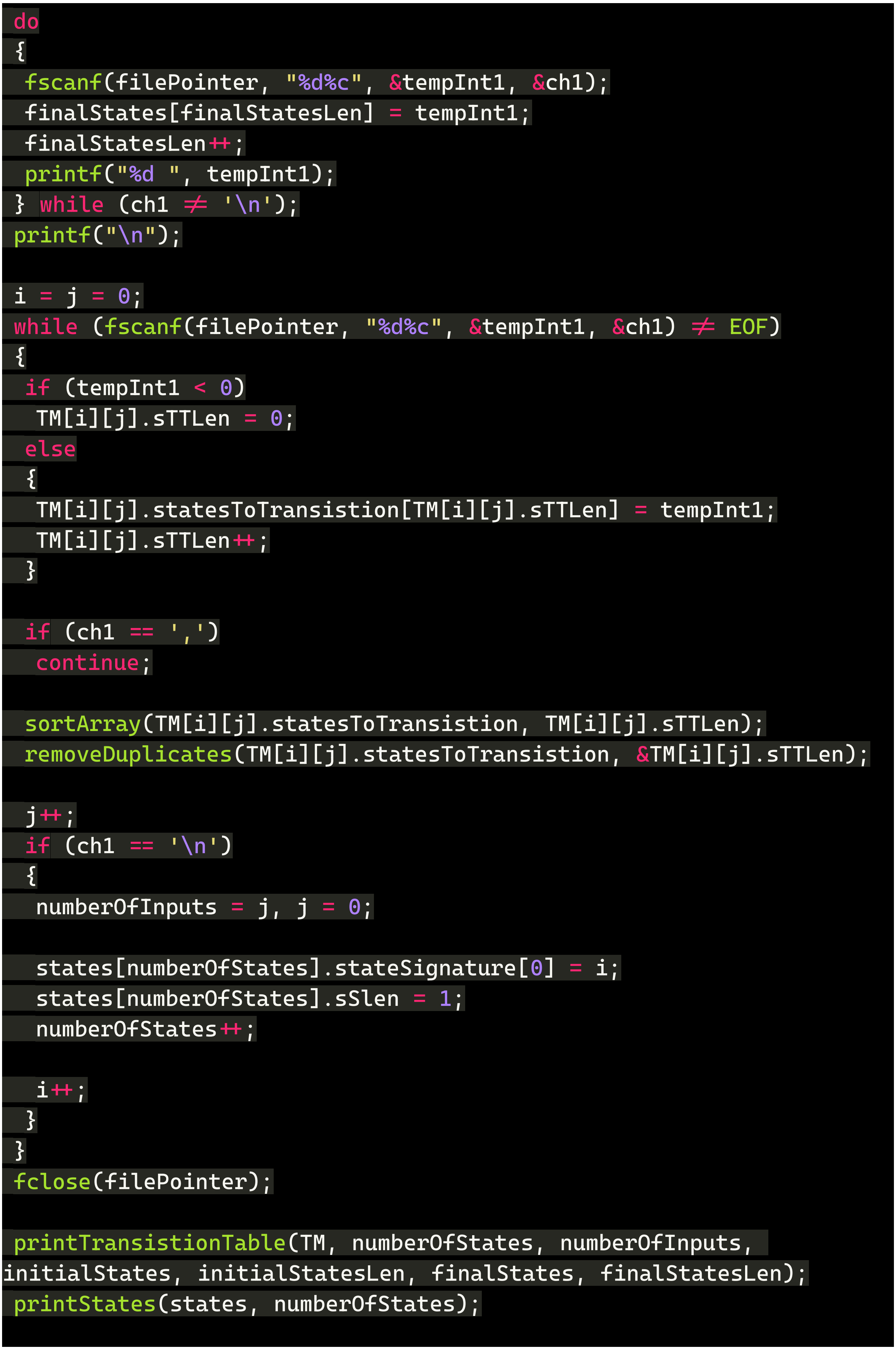
```
 {
  int t = states[i].sSlen;
  printf("state index : %2d, ", i);
  printf("state signature : ");
  for (int j = 0; j < t; j++)
   printf("%d, ", states[i].stateSignature[j]);
  printf("\n");
 }
}

void printTransistionTable(struct Transistion_Cell TM[][100], int 
numberOfStates, int numberOfInputs, int *initialStates, int 
initialStatesLen, int *finalStates, int finalStatesLen)
{
 int tempInputs = 4;
 printf("\nTransistion Table\n");
 printf("|    State   |");
 for (int j = 0; j < numberOfInputs; j++)
  printf(" Input(%d)       |", j);
 printf("\n");

 for (int i = 0; i < numberOfStates; ++i)
 {
  printf("| ");

  char str[4] = "   ";
  if (isInStateArr(initialStates, initialStatesLen, i))
   str[0] = '-', str[1] = '>';
  if (isInStateArr(finalStates, finalStatesLen, i))
   str[2] = '*';
  printf("%s%3d    | ", str, i);
  for (int j = 0; j < numberOfInputs; j++)
  {
   int t = tempInputs;
   if (TM[i][j].sTTLen == 0)
   {
    printf("%2d              | ", -1);
    continue;
   }

   for (int k = 0; k < TM[i][j].sTTLen - 1; k++)
   {
    printf("%2d,", TM[i][j].statesToTransistion[k]);
    t--;
```



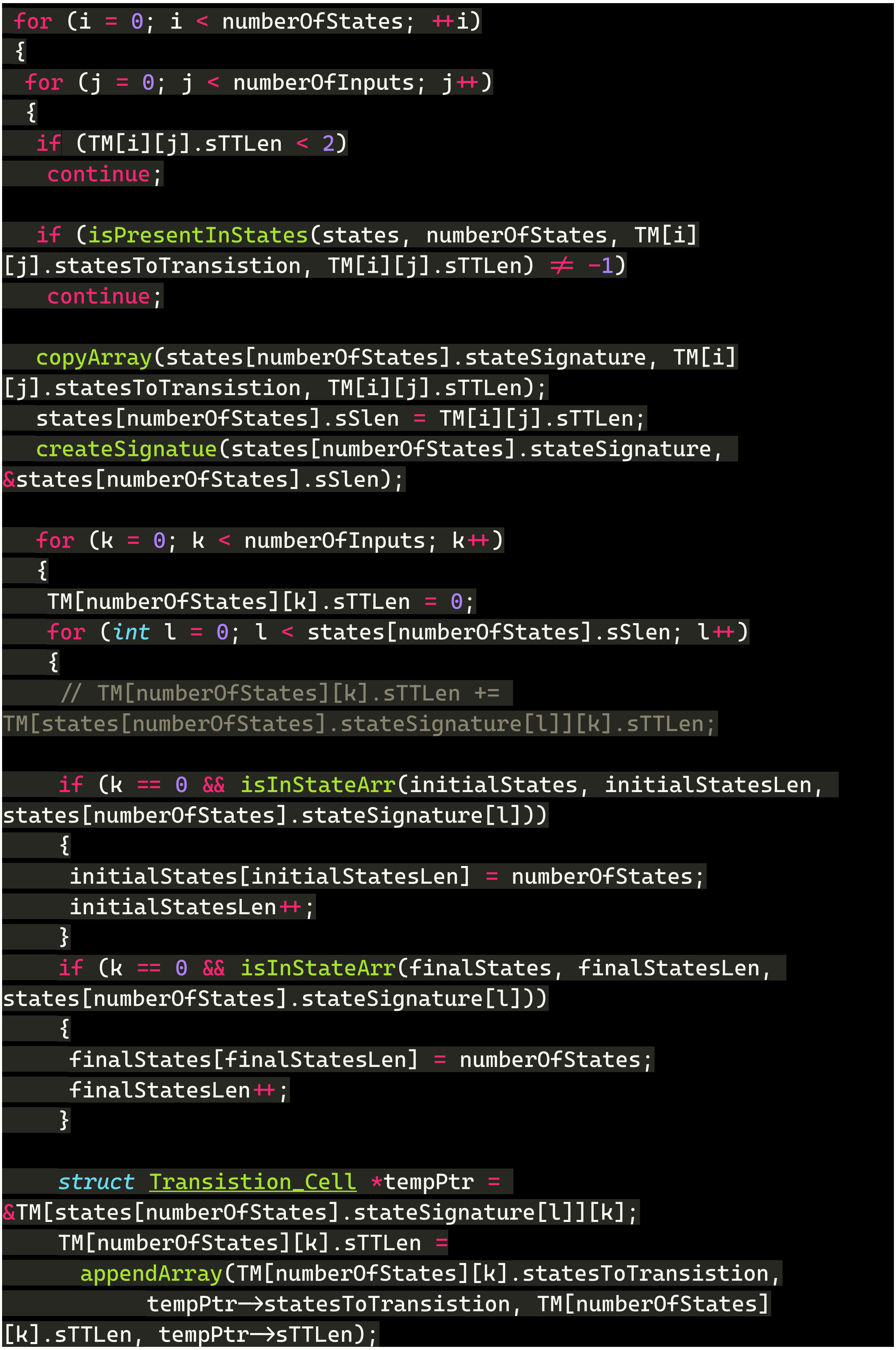
```
   }
   printf("%2d ", TM[i][j].statesToTransistion[TM[i][j].sTTLen -
1]);
   t--;
   while (t > 0)
    printf("   "), t--;

   printf("  | ");
  }
  printf("\n");
 }
}

void printDFATransistionTable(struct State *states, int sLen, struct
Transistion_Cell TM[][100], int numberOfStates, int numberOfInputs,
int *initialStates, int initialStatesLen, int *finalStates, int
finalStatesLen)
{
 printf("\nDFA Transistion Table\n");
 printf("|   State  |");
 for (int j = 0; j < numberOfInputs; j++)
  printf(" Input(%d) |", j);
 printf("\n");

 for (int i = 0; i < numberOfStates; ++i)
 {
  printf("| ");

  char str[4] = "   ";
  if (isInStateArr(initialStates, initialStatesLen, i))
   str[0] = '-', str[1] = '>';
  if (isInStateArr(finalStates, finalStatesLen, i))
   str[2] = '*';
  printf("%s%3d   | ", str, i);

  for (int j = 0; j < numberOfInputs; j++)
  {
   if (TM[i][j].sTTLen == 0)
   {
    printf("  %3d    | ", -1);
    continue;
   }
   printf("  %3d    | ", isPresentInStates(states, sLen, TM[i]
[j].statesToTransistion, TM[i][j].sTTLen));
```



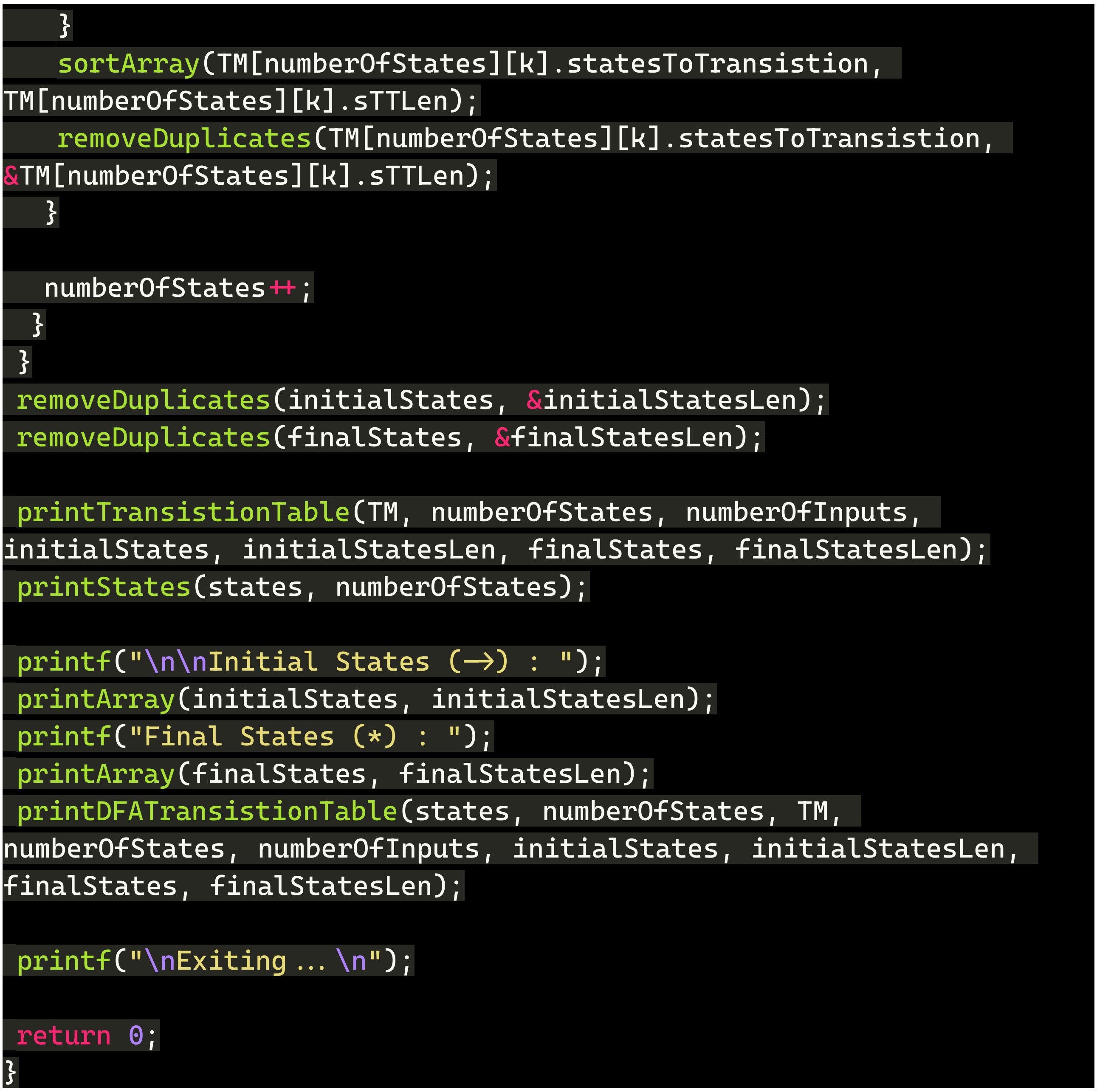
```c
  }
  printf("\n");
 }
}

int main()
{
 int i = 0, j = 0, k = 0;
 int tempInt1, tempInt2;
 char ch1, ch2;

 FILE *filePointer = NULL;
 filePointer = fopen("NFAtoDFA.txt", "r");

 int finalStates[MAX_FINAL_STATES] = {0};
 int initialStates[MAX_FINAL_STATES] = {0};
 int reachableStates[MAX_FINAL_STATES] = {0};
 unsigned int finalStatesLen = 0;
 unsigned int initialStatesLen = 0;
 unsigned int reachableStatesLen = 0;

 struct Transistion_Cell TM[MAX_VARIABLES][MAX_VARIABLES];
 for (i = 0; i < MAX_TRANSISTION; ++i)
  for (j = 0; j < MAX_VARIABLES; ++j)
   TM[i][j].sTTLen = 0;

 if (filePointer == NULL)
 {
  printf("Unable to open NFAtoDFA.txt\n");
  return 1;
 }

 int initialState = -1;
 int currentState = -1;
 int numberOfStates = 0;
 int numberOfInputs = 0;
 struct State states[MAX_FINAL_STATES];

 fscanf(filePointer, "%d%c", &initialState, &ch1);
 initialStates[initialStatesLen] = initialState;
 initialStatesLen++;
 printf("Initial state (->) : %d\n", initialState);

 printf("Final states (*) : ");
```



```
 do
 {
  fscanf(filePointer, "%d%c", &tempInt1, &ch1);
  finalStates[finalStatesLen] = tempInt1;
  finalStatesLen++;
  printf("%d ", tempInt1);
 } while (ch1 != '\n');
 printf("\n");

 i = j = 0;
 while (fscanf(filePointer, "%d%c", &tempInt1, &ch1) != EOF)
 {
  if (tempInt1 < 0)
   TM[i][j].sTTLen = 0;
  else
  {
   TM[i][j].statesToTransistion[TM[i][j].sTTLen] = tempInt1;
   TM[i][j].sTTLen++;
  }

  if (ch1 == ',')
   continue;

  sortArray(TM[i][j].statesToTransistion, TM[i][j].sTTLen);
  removeDuplicates(TM[i][j].statesToTransistion, &TM[i][j].sTTLen);

  j++;
  if (ch1 == '\n')
  {
   numberOfInputs = j, j = 0;

   states[numberOfStates].stateSignature[0] = i;
   states[numberOfStates].sSlen = 1;
   numberOfStates++;

   i++;
  }
 }
 fclose(filePointer);

 printTransistionTable(TM, numberOfStates, numberOfInputs,
initialStates, initialStatesLen, finalStates, finalStatesLen);
 printStates(states, numberOfStates);
```

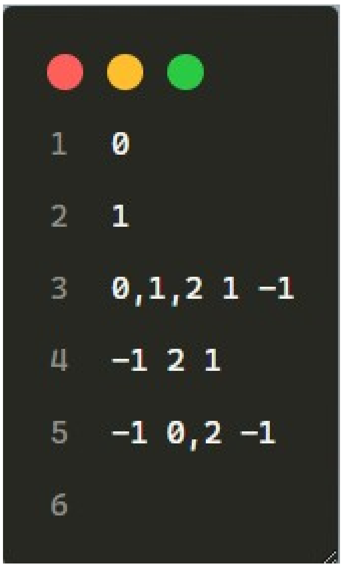


```
 for (i = 0; i < numberOfStates; ++i)
 {
  for (j = 0; j < numberOfInputs; j++)
  {
   if (TM[i][j].sTTLen < 2)
    continue;

   if (isPresentInStates(states, numberOfStates, TM[i]
[j].statesToTransistion, TM[i][j].sTTLen) != -1)
    continue;

   copyArray(states[numberOfStates].stateSignature, TM[i]
[j].statesToTransistion, TM[i][j].sTTLen);
   states[numberOfStates].sSlen = TM[i][j].sTTLen;
   createSignatue(states[numberOfStates].stateSignature,
&states[numberOfStates].sSlen);

   for (k = 0; k < numberOfInputs; k++)
   {
    TM[numberOfStates][k].sTTLen = 0;
    for (int l = 0; l < states[numberOfStates].sSlen; l++)
    {
     // TM[numberOfStates][k].sTTLen +=
TM[states[numberOfStates].stateSignature[l]][k].sTTLen;

     if (k == 0 && isInStateArr(initialStates, initialStatesLen,
states[numberOfStates].stateSignature[l]))
     {
      initialStates[initialStatesLen] = numberOfStates;
      initialStatesLen++;
     }
     if (k == 0 && isInStateArr(finalStates, finalStatesLen,
states[numberOfStates].stateSignature[l]))
     {
      finalStates[finalStatesLen] = numberOfStates;
      finalStatesLen++;
     }

     struct Transistion_Cell *tempPtr =
&TM[states[numberOfStates].stateSignature[l]][k];
     TM[numberOfStates][k].sTTLen =
      appendArray(TM[numberOfStates][k].statesToTransistion,
           tempPtr->statesToTransistion, TM[numberOfStates]
[k].sTTLen, tempPtr->sTTLen);
```



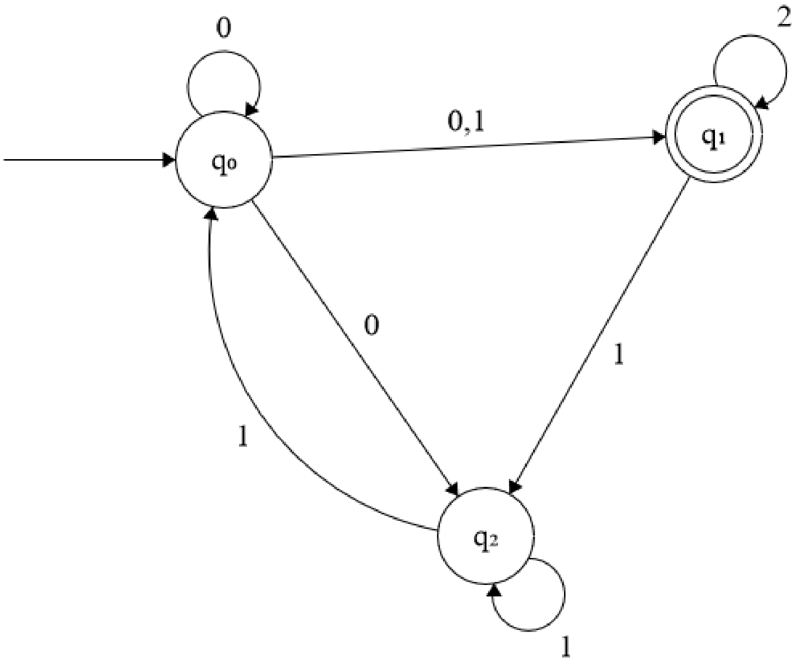
```
    }
    sortArray(TM[numberOfStates][k].statesToTransistion, 
TM[numberOfStates][k].sTTLen);
    removeDuplicates(TM[numberOfStates][k].statesToTransistion, 
&TM[numberOfStates][k].sTTLen);
   }

   numberOfStates++;
  }
 }
 removeDuplicates(initialStates, &initialStatesLen);
 removeDuplicates(finalStates, &finalStatesLen);

 printTransistionTable(TM, numberOfStates, numberOfInputs, 
initialStates, initialStatesLen, finalStates, finalStatesLen);
 printStates(states, numberOfStates);

 printf("\n\nInitial States (->) : ");
 printArray(initialStates, initialStatesLen);
 printf("Final States (*) : ");
 printArray(finalStates, finalStatesLen);
 printDFATransistionTable(states, numberOfStates, TM, 
numberOfStates, numberOfInputs, initialStates, initialStatesLen, 
finalStates, finalStatesLen);

 printf("\nExiting...\n");

 return 0;
}
```

NFAtoDFA.txt

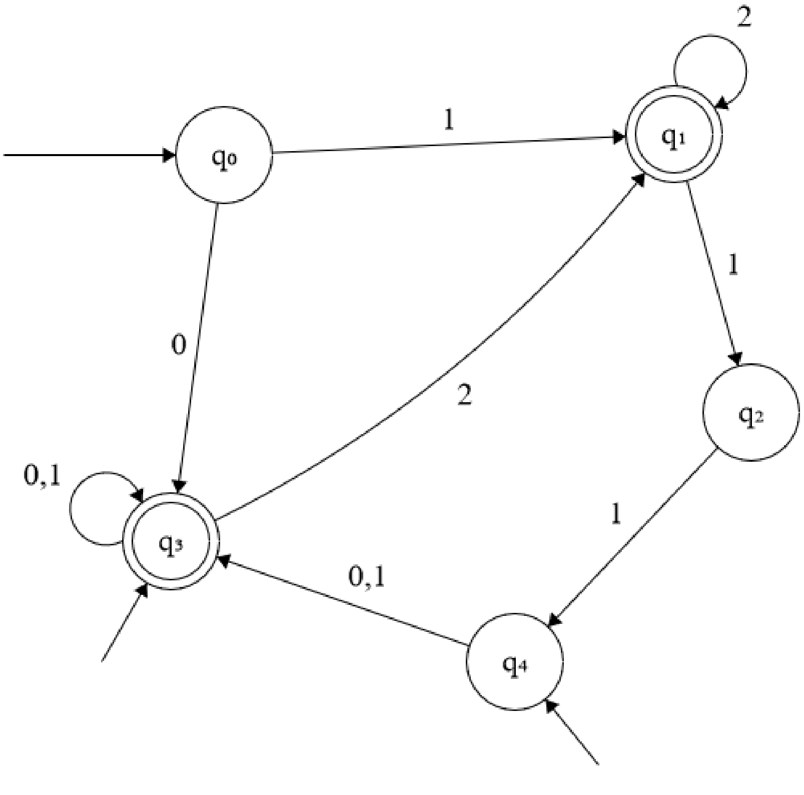


```
0
1
0,1,2 1 -1
-1 2 1
-1 0,2 -1

```

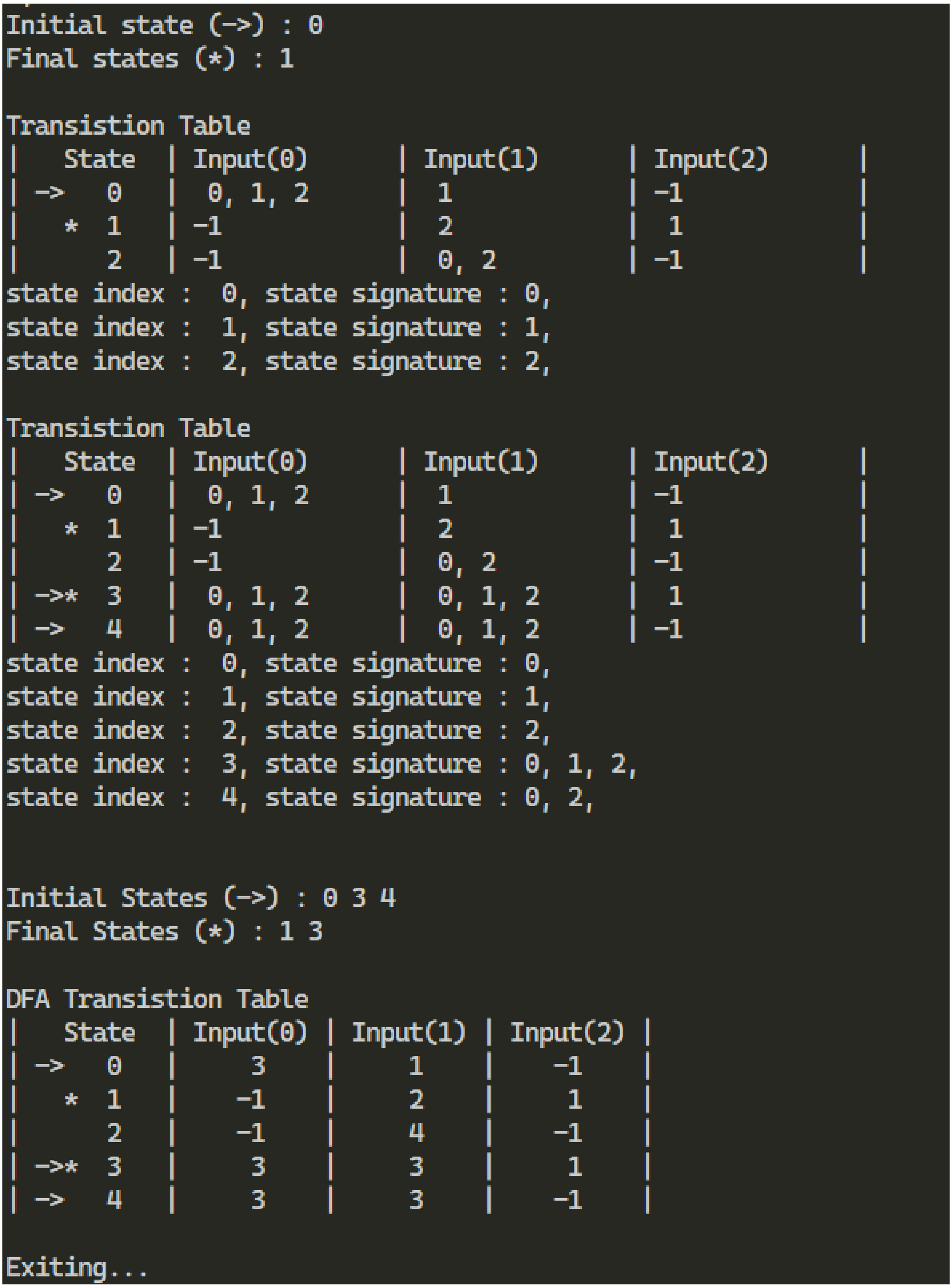
NFA:



DFA:



Output:



```
Initial state (->) : 0
Final states (*) : 1

Transistion Table
|   State   | Input(0)        | Input(1)        | Input(2)        |
| ->   0    |  0, 1, 2        |  1              | -1              |
|    * 1    | -1              |  2              |  1              |
|      2    | -1              |  0, 2           | -1              |
state index :  0, state signature : 0,
state index :  1, state signature : 1,
state index :  2, state signature : 2,

Transistion Table
|   State   | Input(0)        | Input(1)        | Input(2)        |
| ->   0    |  0, 1, 2        |  1              | -1              |
|    * 1    | -1              |  2              |  1              |
|      2    | -1              |  0, 2           | -1              |
| ->*  3    |  0, 1, 2        |  0, 1, 2        |  1              |
| ->   4    |  0, 1, 2        |  0, 1, 2        | -1              |
state index :  0, state signature : 0,
state index :  1, state signature : 1,
state index :  2, state signature : 2,
state index :  3, state signature : 0, 1, 2,
state index :  4, state signature : 0, 2,


Initial States (->) : 0 3 4
Final States (*) : 1 3

DFA Transistion Table
|   State   | Input(0) | Input(1) | Input(2) |
| ->   0    |    3     |    1     |    -1    |
|    * 1    |   -1     |    2     |     1    |
|      2    |   -1     |    4     |    -1    |
| ->*  3    |    3     |    3     |     1    |
| ->   4    |    3     |    3     |    -1    |

Exiting...
```