본 프로그램은 C언어로 작성했습니다.

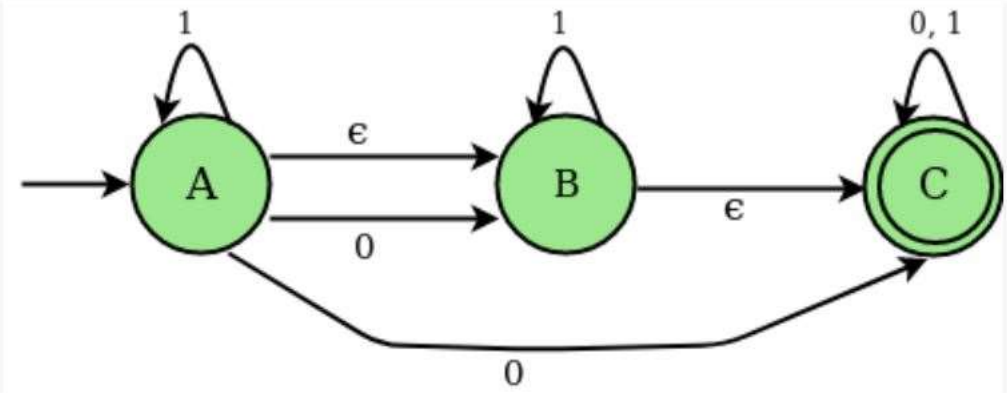
자료구조 정의 :

```
typedef char string[5];

typedef struct _NFA
{
        //NFA_Table[x][0] : state
        string NFA_Table[NFA_TABLE_SIZE_ROW][NFA_TABLE_SIZE_COL];
        string Epsilon_Table[NFA_TABLE_SIZE_ROW][2];//e-closure
}NFA;

typedef struct _DFA
{
        //DFA_Table[x][0] : DFA state
        string DFA_Table[NFA_TABLE_SIZE_ROW][NFA_TABLE_SIZE_COL - 1];
        char Check_Table[ALPHABET_LENGTH]; //중복 문자 확인 테이블
}DFA;
```

NFA와 DFA를 표현하기 위해서 구조체 안에 2차원 배열을 사용했습니다. 본 프로그램에서는 상태 3개와 입력 심볼 2개의 e-NFA를 입력으로 받아서 e-closure를 구한 후 DFA를 구하고 있습니다. NFA 구조체 안에는 e-closure를 저장하기 위해서 Epsilon_Table 배열을 가지고 있습니다. DFA 구조체 안에는 NFA에서 DFA로 전환할 때 발생하는 중복 문자를 제거하기 위해서 Check_Table 배열을 가지고 있습니다. 해당 테이블은 비트맵 방식으로 동작해서 중복되는 문자를 제거하는데 사용됩니다. 아래 그림은 입력 받은 NFA의 테이블입니다.

STATES	0	1	EPSILON
A	BC	A	B
B	-	B	C
C	C	C	-

입력 받은 NFA의 상태 전이도는 아래와 같습니다.



NFA 테이블 4번째 열에 epsilon을 저장했습니다. 문자와 epsilon 입력을 원하지 않을 때는 '-' 를 입력합니다. 다음으로 epsilon-closure 함수의 핵심 부분을 설명하겠습니다.

Epsilon-closure :

```
void Epsilon_Closure_Func(NFA* ptrNFA, int* ptrNumOfStates)
{
	//NFA_Table[x][3] : epsilon
	int i, j, colTraverseNum = 0;
	char tempString[10];
	char noInputEpsilon[2];
	noInputEpsilon[0] = '-';
	noInputEpsilon[1] = '₩0';
	memset(tempString, 0, sizeof(tempString));

	for (i = 0; i < *(ptrNumOfStates); i++)
	{
		strcpy_s(tempString, sizeof(tempString), &(ptrNFA->NFA_Table[i][0]));
		for (j = i; j<*(ptrNumOfStates); j++)
		{
			//epilon col에 '-'가 있는지 확인
			if (strcmp(&ptrNFA->NFA_Table[j][3], &noInputEpsilon) != 0)
			{
				strcat_s(tempString, sizeof(tempString), &(ptrNFA-
>NFA_Table[j][3]));
				colTraverseNum++;
			}

		}
		tempString[colTraverseNum+1] = '₩0';

		//epsilon table 생성
		//ptrNFA->Epilson_Table[i][0] : e-closure(x)
		//ptrNFA->Epilson_Table[i][1] : e-closure(x) = result value
		strcpy_s(&(ptrNFA->Epsilon_Table[i][0]), sizeof(ptrNFA->Epsilon_Table[i][0]),
&(ptrNFA->NFA_Table[i][0]));
		strcpy_s(&(ptrNFA->Epsilon_Table[i][1]), sizeof(ptrNFA->Epsilon_Table[i][1]),
&tempString);

	}
```

epsilon closure는 주어진 상태 x에서 x로부터 도달할 수 있는 e-move와 x를 포함한 상태들의 집합입니다. E-move가 포함되지 않는 상태들을 나타내기 위해서는 '-' 를 입력했습니다. '-'를 확인하기 위해서 if (strcmp(&ptrNFA->NFA_Table[j][3], &noInputEpsilon) != 0) 조건문을 사용했습니다. ptrNFA->NFA_Table[j][3]은 epsilon이 저장된 col이고, 해당 col에 '-' 이 저장되었는지 strcmp로 비교합니다. '-' 이 저장되어 있으면 if문을 넘어가고 문자를 복사하지 않습니다. '-' 이 아닌 알파벳이 저장되어 있으면 tempString에 strcat_s로 문자를 저장합니다. 내부의 for문이 실행 완료되면 e-closure(x)의 결과 값을 tempString이 가지고 있습니다. 그런 다음 strcpy_s를 사용해서 ptrNFA->Epsilon_Table에 e-closure(x)의 결과 값을 저장합니다. ptrNFA->Epsilon_Table[i][0]은 상태를 저장하고 ptrNFA->Epsilon_Table[i][1]에는 e-closure(x)의 결과 값이 저장됩니다.

Convert NFA To DFA :

```
void Convert_NFA_To_DFA_Func(NFA* ptrNFA, DFA* ptrDFA, int* ptrNumOfStates)
{
	int i, j, k;
	char tempString[10];
	char noInputEpsilon[2];
	noInputEpsilon[0] = '-';
	noInputEpsilon[1] = '₩0';
	memset(tempString, 0, sizeof(tempString));

	for(i=0; i<*(ptrNumOfStates); i++)
		strcpy_s(&(ptrDFA->DFA_Table[i][0]), sizeof(ptrDFA->DFA_Table[i][0]),
&(ptrNFA->Epsilon_Table[i][1]));

	for (k = 1; k < NFA_TABLE_SIZE_COL - 1; k++)
	{
		for (i = 0; i < *(ptrNumOfStates); i++)
		{
			for (j = i; j < *(ptrNumOfStates); j++)
			{
				if (strcmp(&ptrNFA->NFA_Table[j][k], &noInputEpsilon) != 0)
					strcat_s(tempString, sizeof(tempString), &(ptrNFA-
>NFA_Table[j][k]));


			}
			strcpy_s(&(ptrDFA->DFA_Table[i][k]), sizeof(ptrDFA->DFA_Table[i][k]),
tempString);
			memset(tempString, 0, sizeof(tempString));
		}
	}

	//NFA에서 DFA로 바꿀 때 발생한 중복 문자 제거
	Check_Duplicate_Character(ptrDFA, ptrNumOfStates);
```

Epsilon table의 2열인 ptrNFA->Epsilon_Table[i][1]은 e-closure(x)의 결과 값을 가지고 있습니다. E-closure(x)의 결과 값이 DFA의 상태가 되기 때문에 처음 for문을 사용해서 DFA 테이블 상태열에 e-closure(x)의 결과 값을 저장해줍니다. 이제 e-closure(x)의 상태에서 접근할 수 있는 상태를 파악해야 하므로 3중 for문을 사용해서 전이 결과를 가져옵니다. 처음 NFA 표 상에서 ABC의 0으로 전이를 알기 위해서는 NFA 표 1열의 세로 상태들을 모두 합치면 ABC가 0으로 전이일 때 접근할 수 있는 상태들을 얻을 수 있습니다. 그래서 결과값은 BC가 나옵니다. 다른 예로 ABC가 1로 전이를 하면 2열의 세로 상태들을 다 합치면 ABC로 전이하는 것을 알 수 있습니다. 이와 같은 방법으로 NFA에서 DFA로 전환할 수 있습니다. 그런데 상태들을 합치다 보면 중복되는 문제가 삽입되는 것을 확인할 수 있었습니다. 중복되는 문제를 제거하기 위해 Check_Duplicate_character(ptrDFA, ptrNumOfStates); 함수를 사용했습니다.

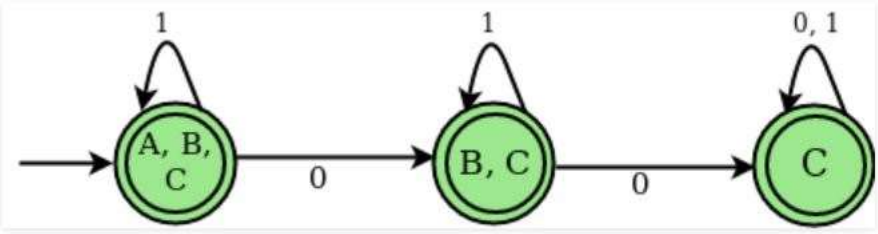
Check Dupilcate Character :

```
if (ptrDFA != NULL)
	{
		for (k = 1; k < NFA_TABLE_SIZE_COL - 1; k++)
```

```
		{
			for (i = 0; i < *(ptrNumOfStates); i++)
			{
				memset(tempString, 0, sizeof(tempString));
				memset(ptrDFA->Check_Table, 0, sizeof(ptrDFA->Check_Table));
				for (j = 0; j < strlen(ptrDFA->DFA_Table[i][k]); j++)
				{
					//중복 문자를 제거하기 위해 비트맵 사용
					//3차원 배열로 접근
					//i : row
					//k : col
					//j : string의 포인터
					if (ptrDFA->Check_Table[ptrDFA->DFA_Table[i][k][j] -
65] == 0)
					{
						ptrDFA->Check_Table[ptrDFA-
>DFA_Table[i][k][j] - 65] = 1;
						tempString[j] = ptrDFA->DFA_Table[i][k][j];
					}
				}
				tempString[strlen(tempString)] = '₩0';
				strcpy_s(&(ptrDFA->DFA_Table[i][k]), sizeof(ptrDFA-
>DFA_Table[i][k]), tempString);
			}
		}
```

DFA 표에서 중복 문자를 제거하기 위해 3중 for문을 사용했습니다. 3중 for문을 사용하는 이유는 테이블의 상태를 저장할 때 `typedef char string[5];`를 사용해서 하나의 셀당 string 자료형을 주었기 때문입니다. 코드를 보면 중복 문자를 확인하기 위해 alphabet bitmap을 사용하는 것을 확인할 수 있습니다. 해당되는 문자가 있으면 비트맵 상에 1로 표시하고, 중복되는 문자가 나타나면 이미 비트맵에는 1로 표시가 되어 있어서 중복 문자를 제거하는 것을 확인할 수 있었습니다. 중복 문자가 제거된 결과값은 다시 DFA 표의 상태 셀에 저장되고 상태 셀당 이 과정을 반복하고 더 이상 실행할 셀이 없으면 최종 DFA를 얻을 수 있었습니다.

DFA STATE DIAGRAM



결과 화면 :

STATES	0	1	EPSILON
A	BC	A	B
B	-	B	C
C	C	C	-

```
Input Number Of States : 3
Input NFA Table :
STATES    0    1     EPSILON
A BC A B
B - B C
C C C -

e-closure(A) : ABC

e-closure(B) : BC

e-closure(C) : C

DFA Table :
STATES    0    1
ABC    BC    ABC
BC    C    BC
C    C    C
계속하려면 아무 키나 누르십시오 . . .
```

STATES	0	1	EPSILON
A	B	A	B
B	C	BC	C
C	C	C	-

```
Input Number Of States : 3
Input NFA Table :
STATES    0    1     EPSILON
A B A B
B C BC C
C C C -

e-closure(A) : ABC

e-closure(B) : BC

e-closure(C) : C

DFA Table :
STATES    0    1
ABC    BC    ABC
BC    C    BC
C    C    C
계속하려면 아무 키나 누르십시오 . . .
```