```
#define SIZE 10000

struct{
    char str;
    int iIndex;
    int jIndex;
    //LCS백 트래킹해서 스택에 넣을 데이터 구조체
} typedef data;

int top = -1; //스택의 top 초기화

int list[SIZE+1][SIZE+1];

int is_stack_empty();
int is_stack_full();
void push(data);
data pop();

int max(int, int);
void makeLCSArray(int [SIZE+1][SIZE+1], char [SIZE+1], char [SIZE+1]);

int max(int a, int b)   //a, b중 큰값을 반환하는 함수
{
    if(a>b)
    {
        return a;
    }
    else return b;
}

void makeLCSArray(int LCS[SIZE+1][SIZE+1], char AAA[SIZE+2], char BBB[SIZE+2])
{
    //LCS알고리즘을 위한 배열을 생성하는 함수

    for(int i=0; i<SIZE+1; i++)
    {
        for(int j=0; j<SIZE+1; j++)
        {
            LCS[i][j] = 0;
        }
    }   //배열을 모두 0으로 초기화

    for(int i=1; i<SIZE+1; i++)
    {
        for(int j=1; j<SIZE+1; j++)
        {
            if(BBB[i-1]-AAA[j-1] == 0)
            {
                LCS[i][j] = LCS[i-1][j-1] + 1;
                //AAA와 BBB의 값이 같은 경우 좌측상단에서 +1한 값을 대입
            }
            else{
                LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1]);
                //두개의 값이 다른경우 좌측과 상측 중에 큰값을 대입
            }
        }
    }
}

data stack[SIZE];
```

```
void test_main(char A[SIZE+1],char B[SIZE+1],char resultStr[2][SIZE*2]){

    int k=0;
    int tailI=0, tailJ=0;

    for(int i=0; i<SIZE+1; i++)
    {
        if(A[i] == '\0')
        {
            tailJ = i;
            break;
            //A 문자열의 끝을 tailJ에 대입
        }
    }
    for(int i=0; i<SIZE+1; i++)
    {
        if(B[i] == '\0')
        {
            tailI = i;
            break;
            //B 문자열의 끝을 tailI에 대입
        }
    }

    makeLCSArray(list, A, B);   //LCS알고리즘을 위한 배열 구성

    int iIndex = SIZE, jIndex = SIZE;
    k = list[iIndex][jIndex];
    //k는 LCS배열의 최우측하단의 값을 대입. (최대 공통 수열의 길이)

    while(1)
    {
        if(A[jIndex-1] != B[iIndex-1])
        {
            //A와 B의 값이 다른경우

            if(list[iIndex][jIndex-1] > list[iIndex-1][jIndex])
            {
                jIndex--;
            }
            else if(list[iIndex][jIndex-1] < list[iIndex-1][jIndex])
            {
                iIndex--;
            }
            //위와 좌측중에 값이 큰쪽으로 이동
            else if(list[iIndex][jIndex-1] == list[iIndex-1][jIndex])
            {
                jIndex--;
                //위와 좌측의 값이 같다면 좌측으로 이동
            }
        }
        else if(A[jIndex-1] == B[iIndex-1] && (A[jIndex-1] != '\0'))
        {
            data temp;
            temp.str = A[jIndex-1];
            temp.jIndex = jIndex -1;
            temp.iIndex = iIndex -1;
            //값이 같다면 스택에 데이터를 넣는다.
            //jIndex와 iIndex에서 1을 빼는 이유는 LCSArray의 크기는 [SIZE+1], 즉, 행과
```

```
                열이 원래 데이터보다 1개씩 더 많기 때문

            push(temp);

            jIndex--;
            iIndex--;
            //데이터를 푸시 하고 좌측상단으로 이동
            k--;
        }
        else{

            jIndex--;
            iIndex--;
            k--;
        }

        if(k==0) break;
        //k==0인 경우는 최장공통수열 만큼 모두 스택에 푸시 된경우.
    }

    int lastI=0;
    //데이터를 넣고 난뒤 마지막 i인덱스가 저장될 변수
    int lastJ=0;
    //데이터를 넣고 난뒤 마지막 j인덱스가 저장될 변수
    int firstK=0, secondK=0;
    //firstK : resultStr[0]의 인덱스
    //secondK : resultStr[1]의 인덱스

    k = 0;

    while(!is_stack_empty())    //스택이 empty될때까지 반복
    {
        data temp;
        temp = pop();//스택에서 데이터를 pop
        int distI = temp.iIndex - lastI, distJ = temp.jIndex - lastJ;

        if(distI > distJ)
        {
            for(;lastI<=temp.iIndex; lastI++)
            {
                resultStr[1][secondK] = B[lastI];
                secondK++;
                //i인덱스의 길이가 더 길다면, B의 데이터를 lasti부터 temp.iIndex(stack)까
                    지 모두 집어 넣는다.
            }

            for(;firstK<secondK; firstK++)
            {
                //firstK가 secondK가 될때까지 A의 lastJ부분의 데이터를 집어 넣는다.

                if(lastJ != temp.jIndex || firstK == secondK-1)
                {
                    resultStr[0][firstK] = A[lastJ];
                    lastJ++;
                }
                //lastJ가 temp.jIndex와 같은데 firstK가 secondK-1과 다르다면 공백을 넣
                    어서 맞춘다.
                else resultStr[0][firstK] = ' ';
            }
        }
```

```
    else if(distI < distJ)
    {
        for(;lastJ<=temp.jIndex; lastJ++)
        {
            resultStr[0][firstK] = A[lastJ];
            firstK++;
        }

        for(;secondK<firstK; secondK++)
        {
            if(lastI != temp.iIndex || secondK == firstK-1)
            {
                resultStr[1][secondK] = B[lastI];
                lastI++;
            }
            else resultStr[1][secondK] = ' ';
        }
    }

    else
    {
        for(;lastJ<=temp.jIndex; lastJ++)
        {
            resultStr[0][firstK] = A[lastJ];
            firstK++;
        }

        for(;lastI<=temp.iIndex; lastI++)
        {
            resultStr[1][secondK] = B[lastI];
            secondK++;
        }
    }
}


//A혹은 B한쪽이 끝에 다다랐을 경우
if(lastI == tailI)
{
    for(;lastJ<tailJ;lastJ++)
    {
        resultStr[0][firstK] = A[lastJ];
        firstK++;
        resultStr[1][secondK] = ' ';
        secondK++;
    }
}
else
{
    for(;lastI<tailI;lastI++)
    {
        resultStr[1][secondK] = B[lastI];
        secondK++;
        resultStr[0][firstK] = ' ';
        firstK++;
    }
}
//resultStr끝부분에 '\0'을 넣어준다.
resultStr[0][firstK] = '\0';
```

```
    resultStr[1][secondK] = '\0';
}

int is_stack_empty()
{
    return top == -1;
    //스택이 비었는지 확인
}

int is_stack_full()
{
    return top == SIZE -1;
    //스택이 꽉찬경우
}

void push(data item)
{
    if(is_stack_full())
    {
        //stack full
    }
    else {
        stack[++top] = item;
        //stack에 데이터를 push
    }
}

data pop()
{
    if(is_stack_empty())
    {
        //stack empty
    }
    return stack[top--];
    //stack의 데이터를 pop
}
```