〈로빛 보고서〉

5일차,6일차_최우진

1.

함수가 매개변수를 받을때 받을 수 있는 방식이 call by value, call by reference 두가지 입니다. 즉 값으로 받는것과 주소로 받는 방법이 있습니다.

우선 call by value 방식은 값을 전달해주는 것입니다. 메인함수 지역에서 쓰던 변수를 다른 함수에서 사용하기 위해 전달해주는 방식을 의미합니다. 리턴을 해주지 않는 이상 메인함수에서 다른 함수에 있는 변수값을 사용할 수 없습니다.

즉 다른 함수로 넘어가면 지역변수 전역변수 개념으로 그 지역에서만 사용할 수 있는 변수의 값이 되는 것입니다. 하지만 call by reference 방식은 다릅니다. 메모리에 접근하는 개념이기에 지역변수 처럼 사용하는 것이 아닌 전역변수 처럼 사용할 수 있습니다.

call by reference 방식은 주소에 접근하는 개념입니다. 메인함수에서 매개변수를 통해 주소를 보내고 받는 함수에서는 포인터로 받아 사용하는 개념입니다.

메인 함수에서 포인터 변수와 변수를 선언하고 포인터 변수에 선언한 변수의 주소를 넘기게 된다면 그 변수가 있는 공간(동적할당했으면 힙, 아니면 뭐 스택에 들어있는)에 그 주소를 보내기에 전역개념으로 사용할 수 있습니다.

scanf는 값을 입력받을때 선언한 변수에 접근해서 역참조하여 값을 저장해주는 함수입니다.

scanf 안에 매개변수를 넣어줄때 값을 입력해주는 함수이기에 메모리에 접근해야합니다. 따라서 변수의 주소를 넣어주어야합니다.(&사용해야함.)

예외가 있다면 %s로 문자열 변수를 받을때는 &를 사용하면 안됩니다. 문자열은 변수가 주소 자체이기 때문에 &없이 scanf로 받아줘야합니다. scanf에는 주소를 매개변수로 넣어줘야하고 문자열을 그 자체로 주소이기 때문에 &를 쓰면 안됩니다.

2.

<메인 함수>

```c
int main(int argc, const char * argv[]) {
    int i=0;
    char str1[100];
    char **str2 = (char **)malloc (sizeof(char*)*20);
    for (int i=0;i<20;i++){
        str2[i] = (char *)malloc (sizeof(char)*20);
    }
    char **out = (char **)malloc (sizeof(char*)*20);
    for (int i=0;i<20;i++){
        out[i] = (char *)malloc (sizeof(char)*20);
    }
    i=0;

    while(1)  {
        printf("연산을 선택하세요.\tadd : ~\tremove : ~\ttoggle : ~\tall : ~\tempty : ~\n\n");
        scanf("%s",str1);
        if(strcmp(str1, "quit")==0){
            for (int j = 0; j < i; j++) {
                free(str2[j]);
            }
            free(str2);
            break;
        }
        //empty
        if(strcmp(str1, "empty")==0){
            empty(i, str2);
//            free(str2);
//            char **str2 = (char **)malloc (sizeof(char*)*20);
//            for (int j=0;j<20;j++){
//                str2[j] = (char *)malloc (sizeof(char)*20);
//            }
            printf("연산을 선택하세요.\tadd : ~\tremove : ~\ttoggle : ~\tall : ~\tempty : ~\n\n");
            scanf("%s",str1);
        }
        //all
        if(strcmp(str1, "all")==0){
            all(i, str2);
            i=0;
            printf("연산을 선택하세요.\tadd : ~\tremove : ~\ttoggle : ~\tall : ~\tempty : ~\n\n");
            scanf("%s",str1);
        }
        scanf("%s",str2[i]);

        //add
        if(strcmp(str1, "add")==0){
            i=add(i,str2);
            i++;
        }
        //remove
        if(strcmp(str1, "remove")==0){
            f_remove(i, str2);
            i--;
        }
        //check
        if(strcmp(str1, "check")==0){
            check(i, str2);
        }
        //toggle
        if(strcmp(str1, "toggle")==0){
            i=toggle(i, str2);
            i++;
        }
    }

    return 0;
}
```

-> 변수 선언

i : 반복문에서 사용하기 위해 선언했습니다. i 값을 가지고 더하고 빼면서 구현했습니다.

str1 : 사용자가 메뉴 선택지를 입력하는데 사용하려고 선언한 배열입니다.

str2 : 문자열 2차원 배열을 만들기 위한 이중 포인터입니다. 최대 20개의 행과 각 행당 20개의 문자를 저장할 수 있도록 할당했습니다. 숫자를 문자로 받기 위하여 2차원 배열을 사용해서 이중 포인터로 선언했습니다.

코드 리뷰

우선 i를 갖고 배열을 움직이려고 했습니다. 따라서 무한 루프에 i를 넣고 돌렸습니다. 무한으로 str1값과 str2값을 입력받게 했고 str2값과 관계없는 조건들에 경우가 있기 때문에 str1값을 받은 후 조건문을 걸고 그 다음에 str2를 받게 했습니다. 그리고 받은 값들을 가지고 조건에 맞는 곳에 들어가서 코드를 수행하도록 구현 했습니다.

<add 함수>

```
int add(int i, char **str2) {
    for (int j=0; j<i; j++) {
        if(strcmp(str2[i], str2[j])==0){
            i--;
        }
    }
    printf("{");
    for (int j=0; j<i+1; j++) {
        printf(" %s,",str2[j]);
    }
    printf("}");
    printf("\n");
    return i;
}
```

-> 매개변수를 i와 이중 포인터로 받았습니다. 그 후 문제 조건에 맞게 저장하고 출력되게 하였습니다.

출력 결과

```
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~

add 3
{ 3,}
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~

add 9
{ 3, 9,}
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~
```

<f_remove 함수>

```
void f_remove(int i, char **str2) {
    for (int j=0; j<i+1; j++) {
        if(strcmp(str2[i], str2[j])==0){
            for (int k=j; k<i; k++) {
                strcpy(str2[k], str2[k+1]);
            }
        }
    }
    printf("{");
    for (int j=0; j<i-1; j++) {
        printf(" %s,",str2[j]);
    }
    printf("}");
    printf("\n");
}
```

-> 매개변수를 i와 이중 포인터로 받았습니다. 그 후 문제 조건에 맞게 저장하고 출력되게 하였습니다.

출력 결과

```
{ 3, 9,}
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~

remove 3
{ 9,}
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~

remove 9
{}
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~
```

〈check 함수〉

```
int check(int i, char **str2) {
    int n1=0;
    for (int j=0; j<i; j++) {
        if(strcmp(str2[i], str2[j])==0){
            n1 = 1;
        }
    }
    switch (n1) {
        case 1:
            for (int j=0; j<i+1; j++) {
                if(strcmp(str2[i], str2[j])==0){
                    printf("1\n");
                    break;
                }
            }
            break;
        default:
            printf("0\n");
            break;
    }
    return 0;
}
```

-〉 매개변수를 i와 이중 포인터로 받았습니다. 그 후 문제 조건에 맞게 저장하고 출력되게 하였습니다.

출력 결과

```
{ 3, 9,}
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~

check 3
1
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~

check 8
0
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~
```

〈toggle 함수〉

```c
int toggle(int i, char **str2) {
    int n1=0;
    for (int j=0; j<i; j++) {
        if(strcmp(str2[i], str2[j])==0){
            for (int k=j; k<i; k++) {
                strcpy(str2[k], str2[k+1]);
            }
            n1=1;
        }
    }
    switch (n1) {
        case 1:
            i-=2;
            printf("{");
            for (int j=0; j<i+1; j++) {
                printf(" %s,",str2[j]);
            }
            printf("}");
            printf("\n");
            break;
        default:
            printf("{");
            for (int j=0; j<i+1; j++) {
                printf(" %s,",str2[j]);
            }
            printf("}");
            printf("\n");
            break;
    }
    return i;
}
```

-> 매개변수를 i와 이중 포인터로 받았습니다. 그 후 문제 조건에 맞게 저장하고 출력되게 하였습니다.

출력 결과

```
{ 3, 9,}
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~

toggle 8
{ 3, 9, 8,}
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~

toggle 3
{ 9, 8,}
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~
```

<all 함수>

```
void all(int i, char **str2) {
    for (int i=0; i<9; i++) {
        sprintf(str2[i], "%x", i+1);
    }
    for (int i=15; i<20; i++) {
        sprintf(str2[i-6], "%x", i+1);
    }
    for (int i=20; i<25; i++) {
        sprintf(str2[i-6], "%x", i+1);
    }
    str2[19]="20";

    printf("{");
    for (int j=0; j<20; j++) {
        if(str2[j]==0){
            continue;
        }
        printf(" %s,",str2[j]);
    }
    printf("}");
    printf("\n");
}
```

-> 20까지 억지로 출력했습니다.. 뒤늦게 구현 안해도 된다는 얘기를 들었지만 굳이 없애지는 않았습니다..

출력 결과

```
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~

all
{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,}
```

<empty 함수>

```
void empty(int i, char **str2) {
    i=0;
    printf("{");
    printf(" }");
    printf("\n");
}
```

```
      if(strcmp(str1, "empty")==0){
          empty(i, str2);
//          free(str2);
//          char **str2 = (char **)malloc (sizeof(char*)*20);
//          for (int j=0;j<20;j++){
//              str2[j] = (char *)malloc (sizeof(char)*20);
//          }
          printf("연산을 선택하세요.\tadd : ~\tremove : ~\ttoggle : ~\tall : ~\tempty : ~\n\n");
          scanf("%s",str1);
      }
```

-> 위에가 empty고 아래가 메인에 있는 조건문입니다. 메모리 해제 후 다시 데이터 저장하는 것으로 구현해보려고 했는데 해제가 잘 안되는 오류가 계속 떠서 우선 돌아가겠끔 i값을 만져서 구현했습니다. 메모리 해제하는 방식으로 어떻게 구현해야 하는지 아직 방법을 찾지 못했습니다. 더 생각해보겠습니다.

출력 결과

```
{ 7, 9,}
연산을 선택하세요.   add : ~ remove : ~  toggle : ~  all : ~ empty : ~

empty
{ }
```

3.

<메인함수>

```c
int main(int argc, const char * argv[]) {
    int n1, n2;
    int *p1,*p2;

    printf("열의 수를 입력하세요:");
    scanf("%d",&n1);
    printf("행의 수를 입력하세요:");
    scanf("%d",&n2);

    p1 = &n1;
    p2 = &n2;
    //동적할당으로 배열 만들기
    int **arr = (int **)malloc (sizeof(int*)*n1);
    for (int i=0;i<n1;i++){
        arr[i] = (int *)malloc (sizeof(int)*n2);
    }
    //밑에서 조건 걸기 위해 모두 0으로 초기화
    for (int i=0; i<n1; i++) {
        for (int j=0; j<n2; j++) {
            arr[i][j]=0;
        }
    }
    arr_ij(p1, p2, arr);
    print(p1, p2, arr);
    return 0;
}
```

->변수 선언

n1: 배열의 열 수를 입력받아 저장하는 변수

n2: 배열의 행 수를 입력받아 저장하는 변수

p1, p2: 각각 입력받은 n1과 n2의 주소로 초기화했습니다..

arr: 2차원 배열.

코드 리뷰

우선 행의 수와 열의 수를 입력받게 했습니다. 그 후 call by reference방식을 사용하기 위해 p1과 p2에 각각 n1과 n2의 주소를 할당했습니다.

문제 조건에 따라 arr을 2차원 배열로 동적 할당했습니다. 먼저 n1개의 포인터를 저장할 공간을 할당하고, 각 포인터가 가리키는 곳에 n2개의 정수를 저장할 공간을 할당했습니다.

arr_ij 함수를 호출하고 print 함수 호출했습니다. 출력이 끝난 후 메모리 누수

를 방지하기 위해 free 함수를 사용하여 동적 할당된 메모리를 해제했습니다.

<arr_ij 함수>

```c
void arr_ij(int *n1, int *n2, int **tmp){
    int x = 0, y = 0, k = 1;
    while (k <= ((*n1)*(*n2))) {
        // 오른쪽
        while (((x < *n2) && (tmp[y][x] == 0))) {
            tmp[y][x] = k++;
            x++;
        }
        x--;
        y++;
        // 아래
        while (((y < *n1) && (tmp[y][x] == 0))) {
            tmp[y][x] = k++;
            y++;
        }
        y--;
        x--;
        // 왼쪽
        while (((x >= 0) && (tmp[y][x] == 0))) {
            tmp[y][x] = k++;
            x--;
        }
        x++;
        y--;
        // 위
        while (((y >= 0) && (tmp[y][x] == 0))) {
            tmp[y][x] = k++;
            y--;
        }
        y++;
        x++;
    }
}
```

->함수 arr_ij는 3개의 매개변수를 받게했습니다.
매개변수, 변수 선언
n1: 배열의 행 수를 저장하는 포인터 변수
n2: 배열의 열 수를 저장하는 포인터 변수
tmp: 2차원 배열을 가리키는 포인터의 포인터 (배열의 주소를 가리키는 이중 포인터)

x, y, k는 각각 배열의 인덱스와 값 할당에 사용되는 변수로 선언했습니다. x와 y는 초기값으로 각각 0으로 초기화하고, k는 나선형 배열에 할당될 값으로 초기값은 1로 초기화했습니다.

코드 리뷰
while (k <= ((*n1)*(*n2)))은 배열에 모든 값을 할당할 때까지 반복하도록 했

습니다. n1*n2번 하면 다 돌아가기 때문입니다.

배열을 나선형으로 채우기 위해 4개의 while 반복문을 사용했습니다. 각 반복문은 배열의 한 방향으로 값을 할당하고, 해당 방향으로 더 이상 값을 할당할 수 없을 때까지 반복하도록 했습니다. 조건을 걸때 그 길이만큼 갔는지와 다음에 0이 있는지 두가지를 체크해서 돌아가게 했습니다.

이렇게 하면 tmp 배열에 달팽이 모양으로 값이 할당되게 됩니다.
함수 arr_ij가 종료되면, 주소에 접근해서 call by reference 방식으로 매개변수를 받았기에 tmp 배열에는 달팽이 모양으로 1부터 n1 * n2까지의 숫자가 할당되어 있습니다.

<print 함수>

```c
//출력하는 함수
void print(int *row,int *col,int **pArr){
    for (int i = 0; i < *row; i++) {
        for (int j = 0; j < *col; j++) {
            printf(" %3d", pArr[i][j]);
        }
        printf("\n");
    }
}
```

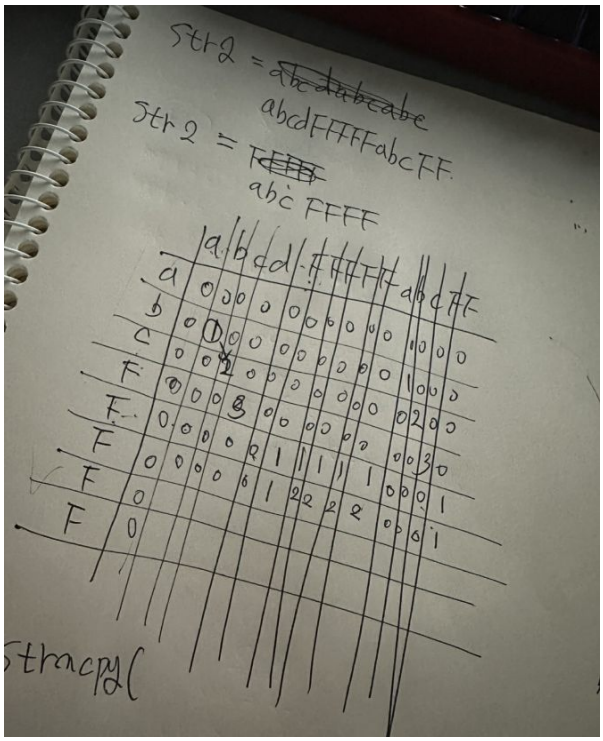
-> 2차원 배열 다 출력했습니다.

출력 결과

```
열의 수를 입력하세요:10
행의 수를 입력하세요:10
  1   2   3   4   5   6   7   8   9  10
 36  37  38  39  40  41  42  43  44  11
 35  64  65  66  67  68  69  70  45  12
 34  63  84  85  86  87  88  71  46  13
 33  62  83  96  97  98  89  72  47  14
 32  61  82  95 100  99  90  73  48  15
 31  60  81  94  93  92  91  74  49  16
 30  59  80  79  78  77  76  75  50  17
 29  58  57  56  55  54  53  52  51  18
 28  27  26  25  24  23  22  21  20  19
Program ended with exit code: 0
```

4.

〈알고리즘〉



(풀면서 그렸던 그림)

정수형 배열을 만들고 전꺼를 비교하면서 더해가는 알고리즘입니다. 반복문을 중첩으로 돌릴때 돌아가는 변수가 i,j라고 하면 둘중 하나라도 0이면 일단 0을 넣어두고 시작합니다. 배열[i-1]==배열[j-1]일때 dp_table이라는 정수형 이차원 배열에 전에 있던 숫자를 더해주는 값을 넣어줍니다. 이렇게 넣어주면 max값과 len값을 손쉽게 구해낼 수 있습니다.(설명이 좀 빈약한데 그림 봐주시길 바랍니다.)

〈메인 함수〉

```c
int main(int argc, const char * argv[]) {
    char input1[100];
    char input2[100];
    char* result = NULL;
    int length;

    while (1) {
        printf("입력1 : ");
        scanf("%99s", input1);

        if (strcmp(input1, "quit") == 0)
            break;

        printf("입력2 : ");
        scanf("%99s", input2);

        length = findLongestCommonSubstring(input1, input2, &result);
        printf("출력 : %s / %d\n", result, length);

        free(result);
        result = NULL;
    }
    return 0;
}
```

->main

변수 선언
 input1: 첫 번째 문자열을 저장하기 위한 문자열 배열 (크기 100)
 input2: 두 번째 문자열을 저장하기 위한 문자열 배열 (크기 100)
 result: 가장 긴 공통 부분 문자열을 저장하기 위한 포인터. f_Substring 함수에서 결과를 동적 할당하므로, 이후에 메모리 해제를 위해 사용했습니다.
 length: f_Substring 함수의 반환값을 저장할 변수로, 가장 긴 공통 부분 문자열의 길이를 나타냅니다.

코드 리뷰
 무한 루프로 사용자로부터 입력을 반복적으로 받게했습니다.

 사용자로부터 첫 번째 문자열을 입력받을때 "입력1 : " 메시지를 출력하고, scanf를 사용하여 사용자가 입력한 문자열을 input1에 저장하게 했습니다. %99s 서식 지정자를 사용하여 최대 99개의 문자를 입력받게했습니다. 조건을 걸어서 입력된 문자열이 "quit"이라면, 무한 루프를 종료하고 프로그램이 종료되도록 했습니다.

 사용자로부터 두 번째 문자열을 입력받을때 역시 printf를 사용하여 "입력2 : " 메시지를 출력하고, scanf를 사용하여 사용자가 입력한 문자열을 input2에 저장하게 했습니다. %99s 서식 지정자를 사용하여 최대 99개의 문자를 입력받았습니다.

 f_Substring 함수를 호출하여 두 문자열에서 가장 긴 공통 부분 문자열과 그 길이를 계산하게 했습니다. input1, input2, 그리고 result의 주소를 f_Substring 함수매개변수에 보냈습니다.

 f_Substring 함수가 실행 후 result는 가장 긴 공통 부분 문자열의 포인터 변수가 저장되게 했습니다. 또한, length 변수에는 가장 긴 공통 부분 문자열의 길이가 저장되게 했씁니다.

result에 동적으로 할당된 메모리를 해제 해줬습니다. 이후에 result를 NULL로 설정하여 잘못된 참조를 방지했습니다.

다시 돌아가서 새로운 입력을 받거나, "quit"을 입력하여 무한 루프를 종료하고 프로그램이 종료되게 했습니다.

<f_Substring 함수>

```
int f_Substring(const char* str1, const char* str2, char** result) {
    int len1 = strlen(str1);
    int len2 = strlen(str2);

    int** dp_table = (int**)malloc((len1 + 1) * sizeof(int*));
    for (int i = 0; i <= len1; i++) {
        dp_table[i] = (int*)malloc((len2 + 1) * sizeof(int));
    }

    *result = (char*)malloc((len1 + 1) * sizeof(char));

    int max_len = 0;
    int max_end = 0;

    for (int i = 0; i <= len1; i++) {
        for (int j = 0; j <= len2; j++) {
            if (i == 0 || j == 0) {
                dp_table[i][j] = 0;
            }
            else if (str1[i - 1] == str2[j - 1]) {
                dp_table[i][j] = dp_table[i - 1][j - 1] + 1;
                if (dp_table[i][j] > max_len) {
                    max_len = dp_table[i][j];
                    max_end = i;
                }
            }
            else {
                dp_table[i][j] = 0;
            }
        }
    }

    for (int i = 0; i <= len1; i++) {
        free(dp_table[i]);
    }
    free(dp_table);

    strncpy(*result, &str1[max_end - max_len], max_len);
    (*result)[max_len] = '\0';
    return max_len;
}
```

-> 우선 비교를 위해 두 입력 문자열 str1과 str2의 길이를 구했습니다. 두 문자열의 길이 중 짧은 길이를 shortest_len에 저장했습니다.

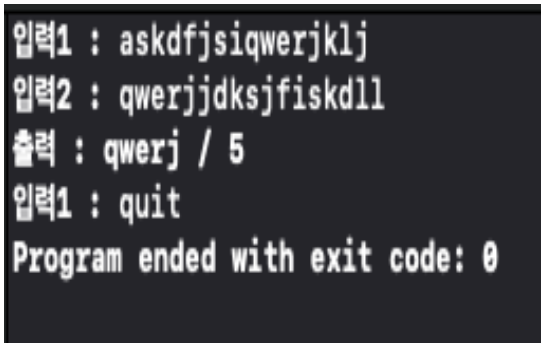
dp_table(다이내믹 프로그래밍 테이블)이라는 2차원 배열을 동적 할당하여 사용했습니다. result 배열을 가장 긴 공통 부분 문자열을 저장하는데 사용하기 위해 동적 할당하여 사용했습니다.

max_len 변수에 가장 긴 공통 부분 문자열의 길이를 저장했습니다. 초기값은 0으로 설정했습니다.

이제 dp_table을 채워 나갑니다. 이중 반복문을 사용하여 str1과 str2의 각 문자를 비교하면서 가장 긴 공통 부분 문자열의 길이를 계산합니다.
만약 str1[i - 1]과 str2[j - 1]가 서로 같으면, dp_table[i][j]의 값은 dp_table[i - 1][j - 1] + 1이 됩니다. 이렇게 하면 공통 부분 문자열의 길이가 1 증가하게 됩니다.

만약 dp_table[i][j]가 max_len보다 크면, max_len을 업데이트하고, result에 가장 긴 공통 부분 문자열을 복사합니다.
그렇지 않으면, dp_table[i][j]는 0이 됩니다. 즉, 현재 위치에서는 공통 부분 문자열이 끝났음을 의미합니다.
계산이 완료되면 dp_table의 메모리를 해제합니다.
(*result)[max_len] = '₩0'을 통해 result에 문자열의 끝을 표시합니다.
가장 긴 공통 부분 문자열의 길이인 max_len을 반환합니다.

출력 결과



5.
<메인 함수>

```c
int main(int argc, const char* argv[]) {
    char input1[100];
    char input2[100];
    char input3[100];
    char* result = NULL;
    int length;

    while (1) {
        printf("입력1 : ");
        scanf("%99s", input1);

        if (strcmp(input1, "quit") == 0)
            break;

        printf("입력2 : ");
        scanf("%99s", input2);

        printf("입력3 : ");
        scanf("%99s", input3);

        length = f_CommonSubstring(input1, input2, input3, &result);
        printf("출력 : %s / %d\n", result, length);

        free(result);
        result = NULL;
    }
    return 0;
}
```

-> 4번과 똑같은데 배열을 하나 더 늘렸습니다.

<f_CommonSubstring 함수>

```c
int f_CommonSubstring(const char* str1, const char* str2, const char* str3, char** result) {
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int len3 = strlen(str3);

    int*** dp_table = (int***)malloc((len1 + 1) * sizeof(int**));
    for (int i = 0; i <= len1; i++) {
        dp_table[i] = (int**)malloc((len2 + 1) * sizeof(int*));
        for (int j = 0; j <= len2; j++) {
            dp_table[i][j] = (int*)malloc((len3 + 1) * sizeof(int));
        }
    }

    *result = (char*)malloc((len1 + 1) * sizeof(char));

    int max_len = 0;
    int max_end = 0;

    for (int i = 0; i <= len1; i++) {
        for (int j = 0; j <= len2; j++) {
            for (int k = 0; k <= len3; k++) {
                if (i == 0 || j == 0 || k == 0) {
                    dp_table[i][j][k] = 0;
                }
                else if (str1[i - 1] == str2[j - 1] && str2[j - 1] == str3[k - 1]) {
                    dp_table[i][j][k] = dp_table[i - 1][j - 1][k - 1] + 1;
                    if (dp_table[i][j][k] > max_len) {
                        max_len = dp_table[i][j][k];
                        max_end = i;
                    }
                }
                else {
                    dp_table[i][j][k] = 0;
                }
            }
        }
    }

    for (int i = 0; i <= len1; i++) {
        for (int j = 0; j <= len2; j++) {
            free(dp_table[i][j]);
        }
        free(dp_table[i]);
    }
    free(dp_table);

    strncpy(*result, &str1[max_end - max_len], max_len);
    (*result)[max_len] = '\0';
    return max_len;
}
```

-> 앞에와 같이 테이블을 만들어서 구현했습니다. 3개를 비교하기에 3차원 배열로 해서 똑같이 구현했습니다.

출력 결과

```
입력1 : ajsdfkqWEefasdfegweoijo
입력2 : fkqWEefaEFAEFBVEWQGF
입력3 : fkqWEefa
출력 : fkqWEefa / 8
입력1 : quit
Program ended with exit code: 0
```