02 프로그래밍 프로젝트

01 다음과 같은 모양을 출력하는 함수를 작성하여 보자.

```
--------------------×--------------------
---------×--------------------×----------
----×---------×---------×---------×-----
-×----×----×----×----×----×----×-----×--
```

(1) 위와 같은 모양을 출력하는 순환 함수 draw_tree(int row, int left, int right)를 설계하여 보자. 먼저 함수의 매개변수는 row과 left, right가 된다. row은 ×를 그리는 행을 표시한다. 가장 위에 있는 행이 0이고 아래로 내려갈수록 숫자는 증가한다고 생각하자. left와 right는 각각 주어진 영역의 왼쪽 끝과 오른쪽 끝을 나타낸다. draw_tree 함수는 주어진 행에서 주어진 영역의 중간 위치를 계산하고 중간 위에 '×'를 출력한 다음에 주어진 영역을 2개로 나누어 각각의 영역에 대하여 각각 draw_tree 함수를 순환 호출하면 된다. 영역이 너무 작으면. 예를 들어 (right-left)<3이면 그냥 복귀하면 된다. 일단 문제를 쉽게 하기 위하여 2차원 문자열 배열에 그린 다음, 한꺼번에 화면에 출력하도록 하자.

```
#define MAX_LEVEL 50
#define MAX_WIDTH 40
char screen[MAX_LEVEL][MAX_WIDTH];

void draw_tree(int row, int left, int right)
{
    int mid;

    if( (right-left)<3 ){
        return;
    }
    mid = (right+left)/2;
    screen[row][mid] = 'X';
    draw_tree(_________________); // 왼쪽 영역
    draw_tree(_________________); // 오른쪽 영역
}
```

(2) main 함수와 screen 문자열을 초기화시키는 init 함수, screen 문자열을 화면에 출력하는 함수 display()를 작성하여 프로그램을 수행시켜본다.

(3) 각 함수가 호출될 때마다 함수의 이름과 매개 변수를 출력하는 문장을 삽입하여 순환 호출이 어떻게 일어나는지를 살핀다.

(4) 위의 순환 함수를 반복적인 기법을 사용하게끔 변환하여 본다.

02 입력되는 글자를 역순으로 출력하는 프로그램을 작성한다고 가정하자. 일반적인 방법으로 해결하려면 먼저 필수적으로 글자를

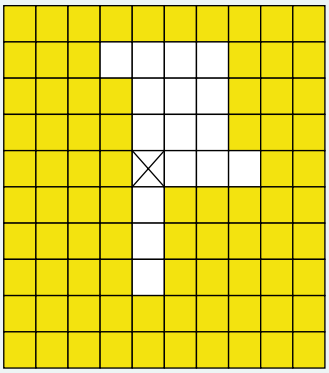
저장하는 배열이 있어야 한다. 즉 배열을 생성한 다음, 글자가 입력되면 먼저 배열에 저장하고 입력이 끝나면 배열에 있는 글자들을 거꾸로 읽어서 출력하여야 한다. 그러나 만약 순환 호출을 사용한다면 다음과 같이 배열없이도 역순으로 글자들을 출력할 수 있다.

```
void reverse()
{
    int ch;
    if( (ch = getchar() ) != '\n'){
        reverse();
        putchar(ch);
    }
}
```

받은 글자를 출력하기 전에 먼저 순환 호출을 하는 것에 유의하라. 따라서 순환 호출이 끝나야만 받은 글자들이 출력된다. 순환 호출이 끝나려면 사용자가 엔터키를 눌러야 한다. 엔터키가 눌려지면 더 이상 순환 호출을 하지 않고 복귀한다. 그 순간 순환 호출되었던 함수들이 차례대로 종료되면서 글자를 역순으로 출력한다. 즉 만약 "book"이라는 문자열을 입력하였다면 "koob"이 출력된다.
위의 프로그램에 함수가 호출될 때마다 함수의 이름과 사용자로부터 받은 문자를 출력하도록 하여 입력이 book이었을 경우에, 어떤 식으로 동작되는지 살펴보라.

03 컴퓨터 그래픽에서의 영역 채우기 알고리즘은 순환 기법을 사용한다. 영역 채우기란 다음과 같은 흰색 영역이 있을 때 이 영역을 특정한 색으로 채우는 것이다. 여기서는 이 영역 안쪽을 검정색으로 채운다고 가정해보라. 이런 경우에는 순환 호출을 어떻게 사용할 수 있을까? 2차원 배열이 다음과 같이 되어 있다고 가정하고 영역안의 한점의 좌표가 주어졌을 경우에 안쪽을 채우는 순환 호출 함수를 작성하여 보라. [그림 2-15]의 ×로 표시된 픽셀이 시작 픽셀이다.

[그림 2-15]
영역 채우기



(a) 영역 채우기 대상 도형

2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2
2	2	2	0	0	0	0	2	2	2
2	2	2	2	0	0	0	2	2	2
2	2	2	2	0	0	0	2	2	2
2	2	2	2		0	0	0	2	2
2	2	2	2	0	2	2	2	2	2
2	2	2	2	0	2	2	2	2	2
2	2	2	2	0	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2

(b) 2차원 배열을 이용하여 (a)를 표현

위의 문제를 해결하려면 먼저 위의 [그림 2-16] (a)를 2차원 배열을 이용하여 (b)와 같이 나타낸다. 노란색은 2로, 흰색은 0으로 영역을 표시한 다음, 0의 값을 갖는 픽셀을 전부 1로 바꾸면 되는 것이다. 다음 프로그램의 빈칸을 채운 다음, 수행시켜서 어떤 순서대로 채워지는 지를 살펴본다.

```
#define WHITE 0
#define BLACK 1
#define YELLOW 2
```

```
int screen[WIDTH][HEIGHT];
//
char read_pixel(int x, int y)
{
    return screen[x][y];
}
//
void write_pixel(int x, int y, int color)
{
    screen[x][y]=color;
}
// 영역 채우기 알고리즘
void flood_fill(int x, int y)
{
    if (read_pixel (x, y) == WHITE)
    {
        write_pixel (x, y, BLACK);
        _____ ; // 순환 호출
        _____ ; // 순환 호출
        _____ ; // 순환 호출
        _____ ; // 순환 호출
        }
}
```