


MY
COMPUTER

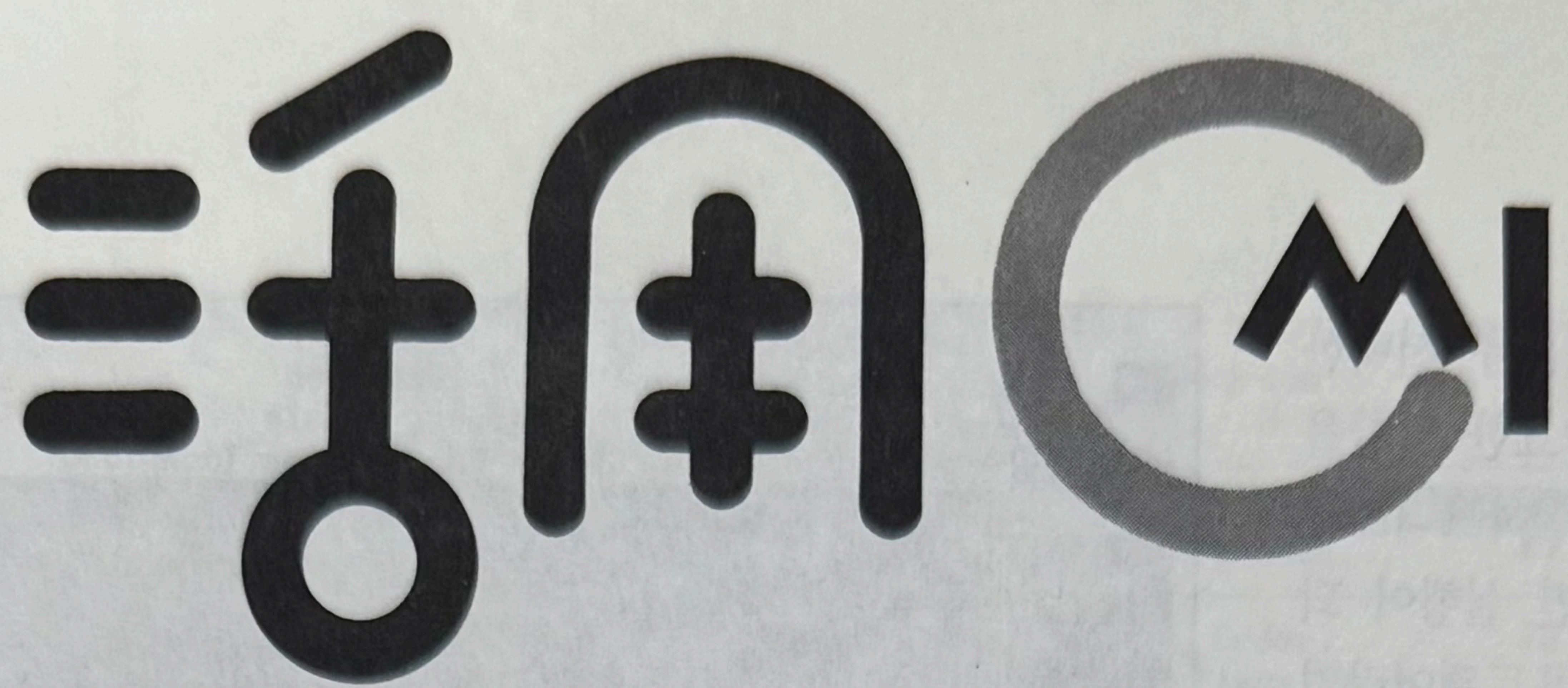
월간 마이컴 통권108호 1992. 9. 25일 발행 1985년 12월 5일 제 3 종우편물(나) 급인가 철도승인141호

특집 / 윈도우 길들이기 기획 / 프린터, 이렇게 구입한다

두대의 컴퓨터에서 자료 전달
화면을 잡는 Snap-Pro 2.0
애니메이션 제작툴 '액션'
하드디스크를 정리한다
펜 컴퓨터란 무엇인가
C언어로 짠 미로탈출 프로그램
우리별 1호의 지상 명령국

별책부록 (게임컴) / 수퍼대전략 / 격투풍운
/ 이스하의 전설 / 에터남 / 울티마 VII 2부





재귀호출을 이용한 미로탈출 프로그램

독자 여러분은 어렸을 때, 미로에 들어가는 꿈을 꾼적이 있을 것이다. 해가 진 후 길을 잊고 나가는 길을 찾는 것은 재미 있지만 한편으로는 무섭기도 했다. 만약 미로에 빠져 있다면, 여러분은 지나가는 길을 표시하면서 가는 방법을 생각할 것이다. 그러다가 막다른 길에 다다르면 다시 돌아가서 마지막으로 지나친 교차로 까지 가서 다른 방향으로 갈 것이다. 출구를 찾을 때까지 이런 방법을 반복할 것이다. 이렇게 길을 되돌아서 가는 것은 미로를 찾는 방법 중 아주 중요한 원리이다.

이러한 미로를 스스로 빠져나갈 수 있는 프로그램을 생각해 본다. 여러 가지 알고리즘이 쓰일 수 있겠지만, 그 중 대표적인 두 가지는 스택(STACK)을 이용하는 방법과 재귀호출(RECURSION)을 이용한 방법이 있다. 스택을 이용하는 방법은 쉽고 또 많이 소개되었으니까 여기서는 재귀호출을 이용해서 프로그램을 작성하는 법을 알아 본다.

재귀호출이란?

우선 재귀호출에 대해서 알아 보도록 한다. 재귀호출이란 한 함수 내에서 그 자신의 함수를 다시 호출(CALL)하는 방식으로 반복되는 루틴에 쓰이는 아주 강력한 프로그래밍 테크닉을 말한다. 간단한 예제를 통해서 좀더 자세히 알아 보도록 한다.

수학에서는 계승(Factorial)이라는 것이 있는데, 어떤 수의 계승은 그 수부터 시작해서 1씩 작은 수를 1일 때까지 곱해나가서 얻은 값을 말한다. 예를 들면, 5의 계승은 $5 * 4 * 3 * 2 * 1 = 120$ 이 되는 것이다. 계승은 숫자 뒤에 ‘!’를 붙여서 표시한다. 계승을 좀더 수학적으로 표시하면 아래와 같다.

$$n! = \begin{cases} 1, & (n=0 \text{ 일 때}) \\ n * (n-1) * (n-2) * \dots * 1, & (n > 0 \text{ 일 때}) \end{cases}$$

이를 다시 생각하면 아래와 같이 표시할 수 있다.

$$n! = \begin{cases} 1, & (n=0 \text{ 일 때}) \\ n * (n-1)!, & (n > 0 \text{ 일 때}) \end{cases}$$

그러면 두 번째 방법으로 $4!$ 을 계산해 보자. 두 번째 방법에서 우리가 알 수 있는 값은 n 이 0일 때 1이라는 것 뿐이다.

먼저, $4! = 4 * 3! \dots$ (1) 이다. 그리고 $3! = 3 * 2! \dots$ (2), 계속해서 $2! = 2 * 1! \dots$ (3), $1! = 1 * 0! \dots$ (4)라고 하자. 이 때 ‘ $0!$ ’이 ‘1’이라는 것을 알고 있으므로 이 값을 (4)에 대입하면 ‘ $1!$ ’은 ‘1’이라는 값을 얻게 되고, 이런 식으로 (3), (2), (1)에 대입하면서 올라가면 ‘ $4! = 24$ ’라는 값을 최종적으로 얻을 수 있을 것이다. 좀 힘든 계산법이지만, 이런 식으로 기초를 알고 있으면 어려운 문제도 금방 생각할 수 있다. 그러면 직접 프로그램으로 만들어 본다.

```
1 N_Factorial(N)
2 int N;
3 {
4 if (N <= 0)
5 return( 1 );
6 else
```

```
7 return( N * N_Factorial(N-1) );  
8 }
```

위 프로그램의 7번째 줄에서 N_Factorial 함수를 다시 부르기 때문
에 재귀호출이라고 하듯 것이다.

재귀호출 합수의 조건

재귀호출 함수에는 중요한 세가지 조건이 필요하다. 이는 재귀호출 함수가 제대로 작동하는지를 확인하는 역할을 한다.

첫번째는 ‘재귀호출을 하지 않고 바로 그 함수에서 나올 수 있는 길이 있으며, 바르게 쓰여지는가?’이다. 즉, 위 프로그램에서 4, 5번 줄과 같이 N_Factorial 함수를 부르지 않고 끝내는 경우($N \leq 0$)가 있는지를 말 한다. 왜냐면 이런 경우(베이스 케이스라고 한다.)가 없다면 그 함수는 무한히 계속 자신을 불러대기 때문이다. 두번째는 ‘처음에 부른 함수가 다시 자신을 부를 때, 처음보다 적은 값으로 부르는가?’이다. 즉, 위에서 변수 N을 갖는 함수 N_Factorial 이 N_Factorial($N-1$)을 부르는 것처럼 파라미터(Parameter)의 값이 점점 작아지는가를 말한다. 세번째는 ‘재귀호출이 될 때 그 함수가 일반적인 경우 올바른 값을 갖는가?’이다. 즉, N_Factorial(N)이 바로 $N!$ 의 값을 갖게 되는가를 말한다. 위의 프로그램에서는 $N! = N * (N-1)!$ 갖게 되므로 맞다.

그러면 재귀호출 함수를 만드는 경우,
그 수서를 알아 본다.

1. 해결해야 할 문제의 정확한 정의를 내린다. 이것은 다른 어떤 프로그래밍에도 해당되는 것이다.

2. 문제의 전체 크기를 정한다. 이
크기는 처음 함수를 부를 때의 파라
미터 값으로 정한다. 즉, 위 프로그
램에서는 N을 말한다.

3. 베이스 케이스를 정한다.

4. 일반적인 경우, 올바른 답이나
올 수 있도록 수식을 만든다. 이 때
중요한 것은 위의 두번째 조건에서
말한것처럼 파라미터가 점점 적어지
도록 만들어야 한다.

미로 탈출의 음악

이제 재귀호출을 이용한 미로탈출 알고리즘을 생각해 보자. 우선 전체적인 프로그램의 진행 순서는 다음과 같다.

1. 입력 : 입력은 $10 * 10$ 의 이차원 행렬을 통하여 받아들이는 것으로 한다. 왜냐면 생각하기가 간단하고 원하면 나중에 확장할 수 있기 때문이다. $10 * 10$ 의 바둑판을 생각하면 된다. 내용은 .(길), #(벽), E(출구) 등으로 되어 있다.

2. 출력 : 시작되는 곳에 ‘*’ 표시를 하고 출구를 찾았을 때 ‘만세! 출구를 찾았습니다!’라는 말을 출력한다. 만약, 출구를 찾지 못하였을 때는 ‘도와주셔요, 길을 찾을 수가 없어요!’라고 출력하고, 그리고 출구를 찾기까지 지나간 거리를 측정해 출력하는 것도 잊어서는 않된다.

3. 수행 알고리즘 : 처음 임의의 시
작위치를 잡고 옮기기 시작한다. 그
리고 출구를 찾았거나 막다가 곳에
이를 때까지 이동한다. 이동은 길(0
표시)을 통해서만 할 수 있다는 것을
잊어서는 않된다. 이를 간단하게 설
명하면 다음과 같다.

```
GetMaze() /* 미로파일을 읽어들인다. */  
/* 시작위치를 잡고 '*' 표시한
```

다. */

```
PrintMaze() /* 현재의 미로를 그린다. */
```

TrytoEscape() /* 시작위치에서 출구를 찾

는다. * /

PrintResult() /* 탈출을 시도한 후의 미로를
그린다. */

}

()'라는 함수에서 쓰인다. 즉, 상하좌우 중 막히지 않은 곳을 계속 찾아가면서 출구를 찾을 때까지 재귀 호출을 하게 되는 함수이다. 뒤에 있는 <프로그램 리스트>를 보면 이해가 될 것이다. <프로그램 리스트>에 있는 프로그램의 결과는 다음과 같다.

출력 결과:

..#E#.##..## #
.##.##.##...
.....##.##.
##.## #..
...##...##.##
.##.##.## ##.##
.##.##...##..
##.## ## ##.## #.
.## *....##.
##.## ##.##...

..#E## + + ## ##
.## ## + ## + ##...
... + + + ##.##.
+ ## ..
+ + + ## + + .##.##
+ ## + ## + ## ##.##
+ ## + ## + + + ##..
+ ## ## ## + ## ..
.## + + + + + ## ..
.## + ## ## + ## ...

만세! 출구를 찾았습니다!

미로 데이터(temp.dat):

..#E#..##
.##.#.##...
.....#.##.
####.#.##..
...#...#.##
.#.#.##.##.
.#.##...##..
##.##.##.##.
.##.....##.
.##.##.##...

이상으로 재귀호출을 이용한 미로 탈출 프로그램을 간단히 만들어 보았다. 의문나는 점이 있으면 필자의 Hitel ID인 bitdoll로 문의 바라며, 프로그램을 잘 이용한다면 멋진 게임도 제작할 수 있을 것이다.

김광·한양대 전산과 3학년

여기서 재구호출은 ‘TrytoEscape

【리스트】

```
#include <stdio.h>

#define START 'S'           /* 시작 위치 */
#define EXIT 'E'            /* 출구 */
#define TRAP 'T'             /* 벽 */
#define OPEN ' '             /* 길 */
#define TRIED '+'            /* 지나간 길 */

#define MINMAZE 1             /* 미로의 갯수 */
#define MAXMAZE 10            /* */
#define TRUE 1                /* Boolean 값을 */
#define FALSE 0               /* 표시하기 위함 */

unsigned char maze[MAXMAZE+2][MAXMAZE+2]; /* 미로의 값 */
int row, col;                                /* 행, 열 */
#define Freed:                /* 출구를 찾았는가? */
FILE *mazefile;                             /* 입력파일 */

/* 쓰이는 함수를 미리 정의한다 */

void Getmaze();
void Printmaze();
void TrytoEscape( int row,int col );
void PrintResult();

/* 간단한 테스트 프로그램입니다 */

void main()
{
    Getmaze();                                /* 처음 시작위치를 임의로 써준다
    row= 9; col= 3;
    maze[row][col] = START;                   /* 'S' 표시를 한다 */
    Freed = FALSE;                           /* 초기치 설정 */
    TrytoEscape(row,col);                   /* 출구를 찾는다 */
    PrintResult();                          /* 결과를 출력 */
    Printmaze();
}

void Getmaze()
{
    int row,col,i;
    unsigned char dummy;

    for (i=MINMAZE-1;i<=MAXMAZE+1;i++)      /* 둘레에 벽을 만들어 놓는다 */
    {
        maze[1][MINMAZE-1] = TRAP;
        maze[1][MAXMAZE+1] = TRAP;
        maze[MINMAZE-1][i] = TRAP;
        maze[MAXMAZE+1][i] = TRAP;
    }
    mazefile = fopen("temp.dat","rt");          /* 파일로 읽어들이고 */
    for (row = MINMAZE;row <= MAXMAZE;row++){
        for (col = MINMAZE;col <= MAXMAZE;col++)
            fscanf(mazefile,"%c",&maze[row][col]); /* 미로를 읽는다 */
            fscanf(mazefile,"%c",&dummy);           /* \n 을 읽기 위함 */
    }
    fclose(mazefile);
}

void Printmaze()
{
    int row,col;
    printf("\n");
    for(row=MINMAZE;row<=MAXMAZE;row++){
        for(col=MINMAZE;col<=MAXMAZE;col++)
            printf("%c",maze[row][col]);
        printf("\n");
    }
}

void TrytoEscape( int row,int col )
{
    if ((maze[row][col] != TRIED) && (maze[row][col] != TRAP) && (!Freed))
        if (maze[row][col] == EXIT)           /* 지나가지 않았고 벽이 아니면 */
            Freed = TRUE;                  /* 출구이면 끝내고 */
        else
        {
            maze[row][col] = TRIED;        /* 출구가 아니면 */
            TrytoEscape( row + 1, col);   /* 지나간 길을 표시하고 */
            TrytoEscape( row - 1, col);   /* 상하좌우에 시도해본다 */
            TrytoEscape( row, col + 1);
            TrytoEscape( row, col - 1);
        }
}

void PrintResult()
{
    if (Freed)
        printf("\n만세! 출구를 찾았습니다!\n");
    else
        printf("\n도와주세요. 길을 찾을 수가 없어요!\n");
}
```

원래 지도 (temp.txt)



출발점 선택



최종 방문결과



□ 통로

■ 벽

★ 출발점

○ 방문

☆ 도착점