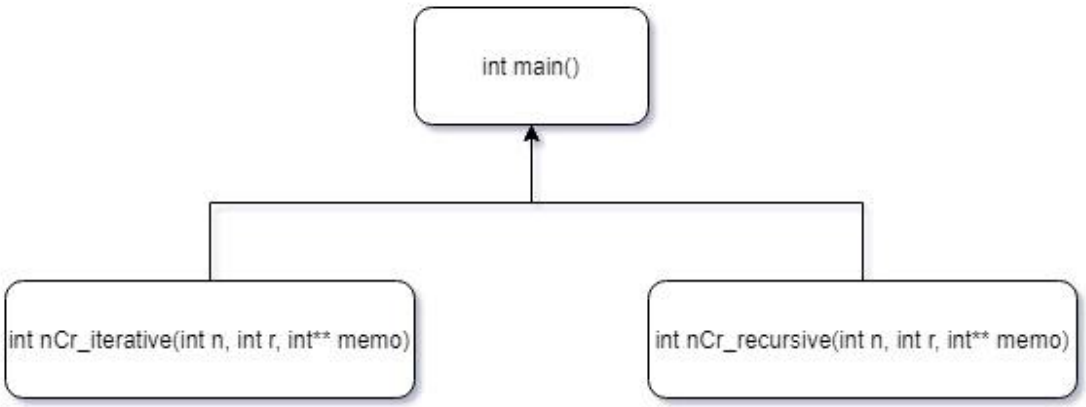
# 20121277 김주호 HW1 보고서

## 1번.

- 함수 구성 다이어그램




- 함수 설명

① int nCr_iterative(int n, int r, int** memo)

nCr의 값을 구하는 함수를 작성하는 데에 다음의 방법을 고려하였다가 폐기하였다.

<방법 : n!을 구현하는 함수를 작성하여 nCr_interative 함수를 단순한 수식 계산으로 n! / (n-r)! r!로 작성한다.>

해당 방법은 10C3을 구하는 데에만 해도, 10!을 계산하게 되어 비효율이 굉장히 심해진다. 왜냐하면, 함수의 큰 방향성이

Step1. 큰 계산을 분자에서 먼저 하고

Step2. 그 큰 수를 분모에서 어느 정도 제거

하는 방향이기 때문이다. 그리 큰 숫자가 아님에도 불구하고 Step1에서부터 overflow가 발생하여 폐기하였다. 따라서, 파스칼의 삼각형의 맨 윗단인 1C0과 1C1을 memo 2차원 배열에 기록한 뒤, 이를 통해 2C0, 2C1, 2C2를 구하고, (물론 모두 구하는 것이 아니고 요구하는 nCr 값에 따라 필요한 항들만 구하게 된다.) 그 아랫 줄을 구하여 memo에 기록하는 방식으로 함수를 구현하였다.

② int nCr_recursive(int n, int r, int** memo)

nCr의 값을 구하는 함수를 작성하는 데에 다음의 방법을 고려하였다가 폐기하였다.

<방법 : f(n,r) = f(n-1, r-1) + f(n-1, r)을 단순히 재귀적으로 구현한다.>

해당 방법은 너무 많은 중복된 함수 call이 있어서, input의 숫자가 증가하면서 지수의 속도로 연산의 부담이 커졌다. 따라서, memo 배열에 구한 값을 기록하고, 이미 기록되어있는 값이라면, 함수를 call하는 대신 기록된 값을 가져오는 방향으로 함수를 작성하였다.

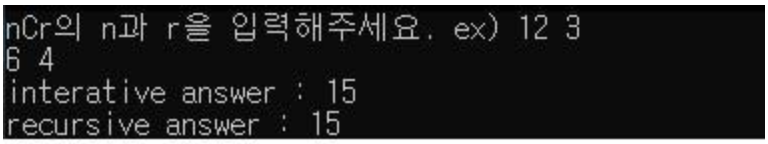
- 코드

```
#include <stdio.h>
#include <stdlib.h>

int nCr_iterative(int n, int r, int** memo) {
	// 이차원 배열에 파스칼의 삼각형 맨 위의 초깃값 1C0, 1C1을 memo[1][0] memo[1][1]에 저장하고,
	// 이를 활용해서 원하는 combination값 까지 추적하는 함수이다.
	int i, j;
	memo[1][0] = 1;
	memo[1][1] = 1;

	for (i = 2; i <= n; i++) {
		for (j = 0; j <= r; j++) {
			if (i == j || j == 0) {
				memo[i][j] = 1;
			}
			else {
				memo[i][j] = memo[i - 1][j - 1] + memo[i - 1][j];
			}
		}
	}
	return memo[n][r];
}

int nCr_recursive(int n, int r, int** memo) {
	// 함수 call의 중복을 막기 위해 memo[n][r]에 nCr 함숫값을 기록
	memo[1][0] = 1;
	memo[1][1] = 1;

	if (r == 0 || n == r) {
		memo[n][r] = 1;
		return memo[n][r];
	}
```

```
        else {
                // memo에 기록된 바 있으면, 그 값을 가져온다.
                if (memo[n][r] != -1) {
                        return memo[n][r];
                }
                // memo에 함숫값 기록이 없다면, 재귀를 통해 함숫값을 구한다.
                else {
                        return nCr_recursive(n - 1, r - 1, memo) + nCr_recursive(n -
1, r, memo);
                }
        }
        return memo[n][r];
}


int main() {
        int** memo;
        int n, r;
        int i,j;
        int interative_ans, recursive_ans;

        printf("nCr의 n과 r을 입력해주세요. ex) 12 3₩n");
        scanf("%d", &n);
        scanf("%d", &r);

        // (n+1) by (r+1) 행렬을 생성
        memo = (int**)malloc((n+1) * sizeof(int*));
        for (i = 0; i <= n; i++) {
                memo[i] = (int*)malloc((r + 1) * sizeof(int));
        }

        // (n+1) by (r+1) 행렬을 초기화
        for (i = 0; i <= n; i++) {
                for (j = 0; j <= r; j++) {
                        memo[i][j] = -1;
                }
        }
        interative_ans = nCr_iterative(n, r, memo);

        // (n+1) by (r+1) 행렬을 초기화
        for (i = 0; i <= n; i++) {
                for (j = 0; j <= r; j++) {
                        memo[i][j] = -1;
```

```
                }
        }
        recursive_ans = nCr_recursive(n, r, memo);

        printf("interative answer : %d₩n", interative_ans);
        printf("recursive answer : %d", recursive_ans);

        return 0;
}
```

- 실행 결과

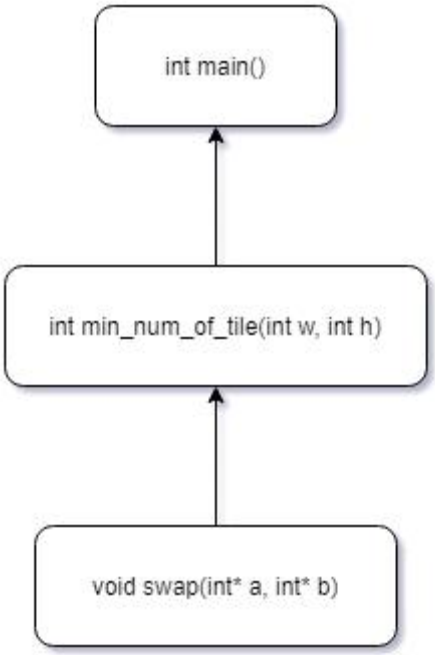







## 2번.

- 함수 구성 다이어그램




- 함수 설명

① void swap(int* a, int* b)

int a와 int b를 가리키는 포인터를 입력으로 받아 a와 b의 값을 바꾸는 함수이다. int min_num_of_tile(int w, int h) 함수에서 input으로 들어오는 가로의 길이와 세로의 길이가 뒤바뀌어도 return하는 값이 같기 때문에, logic을 짜는 편의성을 위해 늘 가로의 길이가 긴 상황으로 설정하기 위한 swap 함수이다.

② int min_num_of_tiles(int w, int h)

주어진 욕실에 들어가는 최소 타일의 개수를 구하는 함수이다. 정사각형 타일의 한 변의 길이가 항상 2의 거듭제곱으로 주어지기 때문에, greedy하게 추적하여도 최선의 결과가 유도된다.

높이 h의 길이보다 작게 타일의 최대길이를 설정하고, 이 타일을 가로 방향으로 최대한으로 채운다. 그렇게 되면, 영역 1과 영역 2가 남게 되는데, 이 부분을 재귀함수로 구현하여 같은 logic으로 최소 타일의 개수를 구하면 된다.

- 코드

```
#include <stdio.h>
#include <math.h>

void swap(int* a, int* b) {
	int tmp;
	tmp = *a;
	*a = *b;
	*b = tmp;
	return;
}

int min_num_of_tiles(int w, int h) {
	int i;
	int len_of_square;
	int q, r;
	int w1, h1, w2, h2;
	int count;

	// 종료조건
	if (w == 0 || h == 0) {
		return 0;
	}
	else {
		// width를 항상 height보다 큰 값으로 만들기 위한 과정
		if (w < h) {
			swap(&w, &h);
		}

		// width에 정사각형 tile이 몇 개 들어가는지 세는 함수
		i = 0;
		while (pow(2, i) <= h) {
			i += 1;
		}
		i -= 1;

		// q : width에 들어갈 수 있는 타일의 개수
		// r : 타일을 최대로 집어넣고 남은 width의 길이
```

```
                len_of_square = (int)pow(2, i);
                q = w / len_of_square;
                r = w % len_of_square;

                w1 = w;
                h1 = h - len_of_square;
                w2 = r;
                h2 = len_of_square;
                count = q + min_num_of_tiles(w1, h1) + min_num_of_tiles(w2, h2);

                return count;
        }
}

int main() {
        int ans;
        int w, h;

        printf("욕실의 가로 세로 크기를 입력하세요. : ");
        scanf("%d", &w);
        scanf("%d", &h);

        ans = min_num_of_tiles(w, h);

        printf("들어갈 수 있는 최소 타일의 개수 : ");
        printf("%d", ans);

        return 0;
}
```

- 실행 결과