C – 알고리즘 1

수학, 재귀 알고리즘

알고리즘 기초

- 알고리즘

  어떤 문제를 해결하기 위한 절차나 방법이다. 주어진 입력을 출력으로 만드는 과정을 구체적이고 명료하게 표현한 것이다.

분류	세부 내용
수학 알고리즘	덧셈 알고리즘, 세 수의 최대값, 최대공약수
재귀 알고리즘	팩토리얼, 피보나치 수열, 이진수 변환, 유클리드
정렬 알고리즘	순위 정하기, 버블 정렬, 선택 정렬, 삽입 정렬
탐색 알고리즘	순차 탐색, 이분 탐색, 문자열

수학 알고리즘

- 덧셈 알고리즘
  - 1부터 5까지의 합

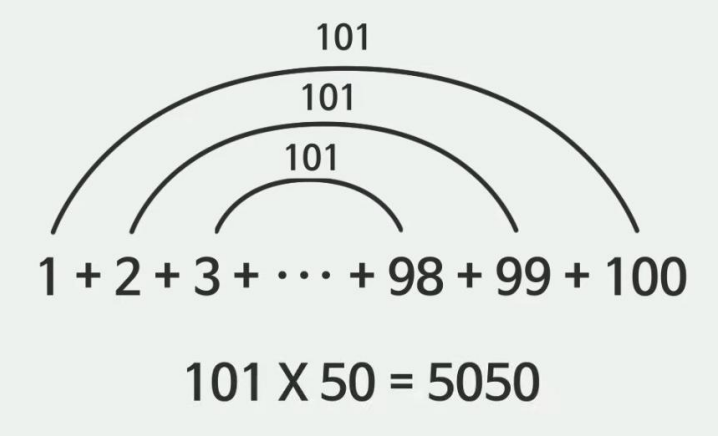
```c
//단순 합계
printf("%d\n", 1 + 2 + 3 + 4 + 5);

//for문 사용
int i, sum = 0;

for (i = 1; i <= 5; i++) {
    sum += i;
}
printf("%d\n", sum);
```

수학 알고리즘

- 1부터 n까지의 합 계산하기

1 + 2 + 3 + ··· + n ⟶ 방법 1

n x (n + 1) ÷ 2 ⟶ 방법 2



수학 알고리즘

- 1부터 n까지의 합 계산하기

```
int sumN(int n) {
    int i, sum = 0;

    for (i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}
```

```
int sumN2(int n) {
    int sum = 0;
    sum = (n * (n + 1)) / 2;

    return sum;
}
```

수학 알고리즘

- 1부터 n까지의 합 계산하기

```
//합계 1
int result1;
result1 = sumN(10);

printf("합계: %d\n", result1);

//합계 2
int result2;
result2 = sumN2(10);

printf("합계: %d\n", result2);
```

계산 복잡도 – 빅오(Big O)

- 계산 복잡도

  - **입력 크기와 계산 횟수**
    - 첫 번째 알고리즘 : 덧셈 n번
    - 두 번째 알고리즘 : 덧셈, 곱셈, 나눗셈(총 3번)

  - **대문자 O표기법(Big O) : 계산 복잡도 표현**
    - O(n) : 필요한 계산횟수가 입력 크기 n과 비례할 때
    - O(1) : 필요한 계산횟수가 입력 크기 n과 무관할 때
  - **판단**
    - 두번째 방법이 계산 속도가 더 빠름

알고리즘 기초

- 평균 구하기

```c
//평균 구하기
int a[] = { 70, 80, 65, 90 };
int sum = 0;
int i;

//단순 평균
printf("%d\n", (a[0] + a[1] + a[2] + a[3]) / 4);

//for문 사용
for (i = 0; i < 4; i++) {
    sum += a[i];
}
printf("평균은 %.1f입니다.\n", sum / 4.0);
```

막대 그래프 그리기

- 막대 그래프 그리기

```c
//int arr[] = { 3, 6, 4, 2 };
int arr[4];
int i, j;

//사용자 입력
for (i = 0; i < 4; i++) {
    printf("arr[%d] 입력: ", i);
    scanf("%d", &arr[i]);
}
printf("\n");

//막대 그래프 출력
for (i = 0; i < 4; i++) {
    printf("arr[%d]=%d|", i, arr[i]);
    for (j = 1; j <= arr[i]; j++) {
        printf("*");
    }
    printf("\n");
}
```

```
arr[0] 입력: 3
arr[1] 입력: 6
arr[2] 입력: 4
arr[3] 입력: 2

arr[0]=3|***
arr[1]=6|******
arr[2]=4|****
arr[3]=2|**
```

막대 그래프 그리기

- 막대 그래프 그리기

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>  //malloc(), free()
#include <stdbool.h> //true/false 사용
```

```c
//동적 메모리 할당
int* arr = NULL; //포인터 선언
int size;  //배열의 크기
int i, j;

printf("배열의 크기 입력: ");
scanf("%d", &size);

arr = (int*)malloc(sizeof(int) * size);
if (arr == NULL) {
    puts("메모리 할당에 실패했습니다.\n");
    return 1;
}
```

막대 그래프 그리기

- 막대 그래프 그리기

```
// 사용자 값 입력 및 유효성 검사
for (i = 0; i < size; i++) {
    while (true) {
        printf("arr[%d] 값 입력 (0 이상 정수): ", i);
        if (scanf("%d", &arr[i]) != 1 || arr[i] < 0) {
            printf("잘못된 입력입니다. 다시 입력하세요.\n");

            // 입력 버퍼 비우기
            while (getchar() != '\n');
        }
        else { //scanf("%d", &arr[i]) == 1
            break; //값 입력후 while문 빠져나옴
        }
    }
}
```

막대 그래프 그리기

- 막대 그래프 그리기

```
/*
    i=0, arr[0]=3, i++ , ***
    i=1, arr[1]=6, i++ , ******
    i=2, arr[1]=a, 잘못된 입력입니다.
    i=2, arr[2]=4, i++ , ****
    i=3, arr[3]=2, i++ , **
    i=4, 반복 종료
*/

//막대 그래프 출력
for (i = 0; i < size; i++) {
    printf("arr[%d]=%d|", i, arr[i]);
    for (j = 1; j <= arr[i]; j++) {
        printf("*");
    }
    printf("\n");
}
free(arr); //메모리 반납
```

```
배열의 크기 입력: 4
arr[0] 값 입력 (0 이상 정수): 3
arr[1] 값 입력 (0 이상 정수): 6
arr[2] 값 입력 (0 이상 정수): f
잘못된 입력입니다. 다시 입력하세요.
arr[2] 값 입력 (0 이상 정수): 4
arr[3] 값 입력 (0 이상 정수): 2

arr[0]=3|***
arr[1]=6|******
arr[2]=4|****
arr[3]=2|**
```

최대값 구하기

- 최대값 구하기

```c
// 두 수 중 큰 값 계산
int x = 10, y = 20;
int result;

result = (x > y) ? x : y;
printf("두 수중 큰 수: %d\n", result);

// 세 수중 큰 수
int a = 10, b = 20, c = 30;
int max;

max = a;  //a를 최대값 설정

if (b > max)
    max = b;
if (c > max)
    max = c;

printf("최대값은 %d입니다.\n", max);
```

최대값 구하기

- 최대값 구하기

```
int max3(int a, int b, int c) {
    int max = a;  //최대값 설정
    if (b > max) max = b;
    if (c > max) max = c;
    return max;
}

int main()
{
    printf("max3(%d, %d, %d) = %d\n", 3, 2, 1, max3(3, 2, 1));
    printf("max3(%d, %d, %d) = %d\n", 3, 1, 2, max3(3, 1, 2));
    printf("max3(%d, %d, %d) = %d\n", 2, 1, 3, max3(2, 1, 3));
    printf("max3(%d, %d, %d) = %d\n", 2, 3, 1, max3(2, 3, 1));

    return 0;
}
```

최대값 구하기 – 배열

- 최대값과 최대값 위치

```
int findMax(int a[], int len) { //최대값
    int i, maxVal;
    maxVal = a[0];
    for (i = 0; i < len; i++) {
        if (a[i] > maxVal)
            maxVal = a[i];
    }
    return maxVal;
}

int findMaxIdx(int a[], int len) { //최대값의 위치
    int i, maxIdx;
    maxIdx = 0;
    for (i = 0; i < len; i++) {
        if (a[i] > a[maxIdx])
            maxIdx = i;
    }
    return maxIdx;
}
```

최대값 구하기 – 배열

- 최대값과 최대값 위치

```
int arr[] = {53, 11, 65, 36, 29};
int max, maxIndex;

max = findMax(arr, 5);
maxIndex = findMaxIdx(arr, 5);

printf("최대값: %d, 최대값의 위치: %d\n", max, maxIndex);
```

```
max3(3, 2, 1) = 3
max3(3, 1, 2) = 3
max3(2, 1, 3) = 3
max3(2, 3, 1) = 3
```

최대값 구하기 – 동적 할당

- 사람의 키를 입력받아 최대값 구하기

```
int number;
int* height; //동적 할당할 포인터배열

printf("사람 수: ");
scanf("%d", &number);
height = (int*)malloc(sizeof(int) * number);

printf("%d명의 키를 입력하세요\n", number);
for (int i = 0; i < number; i++) {
    printf("height[%d]: ", i);
    scanf("%d", &height[i]);
}
printf("최대값은 %d입니다.\n", findMax(height, number));

free(height); //메모리 해제
```

```
사람 수: 3
3명의 키를 입력하세요
height[0]: 172
height[1]: 165
height[2]: 180
최대값은 180입니다.
```

최대값 구하기 – 동적 할당

- 사람의 키를 랜덤하게 생성하고 최대값 구하기

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int findMax(int a[], int len) { //최대값
    int i, maxVal;
    maxVal = a[0];
    for (i = 0; i < len; i++)
        if (a[i] > maxVal) maxVal = a[i];
    return maxVal;
}
```

```
사람 수: 9
height[0] = 113
height[1] = 153
height[2] = 153
height[3] = 152
height[4] = 160
height[5] = 120
height[6] = 174
height[7] = 170
height[8] = 147
최대값은 174입니다.
```

최대값 구하기 – 동적 할당

- 사람의 키를 랜덤하게 생성하고 최대값 구하기

```
int number;
int* height; //동적 할당할 포인터배열

printf("사람 수: ");
scanf("%d", &number);
height = (int*)malloc(sizeof(int) * number);
srand(time(NULL)); //시간으로 난수 seed 설정

for (int i = 0; i < number; i++) {
    height[i] = 100 + rand() % 91; //100 ~ 190의 난수 생성
    printf("height[%d] = %d\n", i, height[i]);
}
printf("최대값은 %d입니다.\n", findMax(height, number));

free(height); //메모리 해제
```

재귀 알고리즘

- 재귀 호출

  어떤 함수 안에서 자기 자신을 부르는 것을 말한다.

  재귀 호출은 무한 반복하므로 종료 조건이 필요하다.

```
func(입력값):
    if 입력값이 충분히 작으면:  //종료 조건
        return 결과값
    else
        func(더 작은 입력값)  //자기 자신 호출
```

재귀 알고리즘

- SOS 구현

```
void func(int n)
{
    //방법 2
    if (n <= 0) { //종료 조건
        return;
    }
    else {
        printf("Help Me!\n");
        func(n - 1);
    }

    //방법 1
    /*printf("Help Me!\n");
    n--;
    if (n <= 0)
        return; //종료 조건
    else
        func(n);
    */
```

재귀 알고리즘

- SOS 구현

```
int main()
{
    int count = 4;

    func(count);

    return 0;
}
```

재귀 알고리즘

- 팩토리얼 계산하기 – 일반적인 계산

```c
#include <stdio.h>
/*
    - 1부터 5까지 곱하기
      1x2x3x4x5 -> 5!
*/
int factorial(int n) {
    int i, facto = 1;
    for (i = 1; i <= n; i++) {
        facto *= i;
    }
    return facto;
}
```

재귀 알고리즘

- 팩토리얼 계산하기 – 일반적인 계산

```
int main()
{
    int a, b, c;

    a = factorial(1);  // 1 * factorial(0), 1 * 1 = 1
    b = factorial(2);  // 2 * factorial(1), 2 * 1 = 2
    c = factorial(3);  // 3 * factorial(2), 3 * 2 = 6

    printf("1!=%d, 2!=%d, 3!=%d\n", a, b, c);

    return 0;
}
```

재귀 알고리즘

- 팩토리얼 계산하기 – 재귀 알고리즘

```
/*
    4! = 4 x 3 x 2 x 1
    4! = 4 x 3!
    3! = 3 x 2 x 1
    3! = 3 x 2!

*/

int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

재귀 알고리즘

- 팩토리얼 계산하기

```
int main()
{
    int a, b, c;

    a = factorial(1);  // 1 * factorial(0), 1 * 1 = 1
    b = factorial(2);  // 2 * factorial(1), 2 * 1 = 2
    c = factorial(3);  // 3 * factorial(2), 3 * 2 = 6

    printf("1!=%d, 2!=%d, 3!=%d\n", a, b, c);

    return 0;
}
```

재귀 알고리즘

- **피보나치(Fibonacci) 수열**

  수학에서 피보나치 수는 첫째 및 둘째 항이 1이며, 그 뒤의 모든 항은 바로 앞 두 항의 합인 수열이다. 처음 여섯 항은 각각 1, 1, 2, 3, 5, 8이다.

  세째항 = 첫째항 + 둘째항

  2 = 1 + 1

  3 = 1 + 2

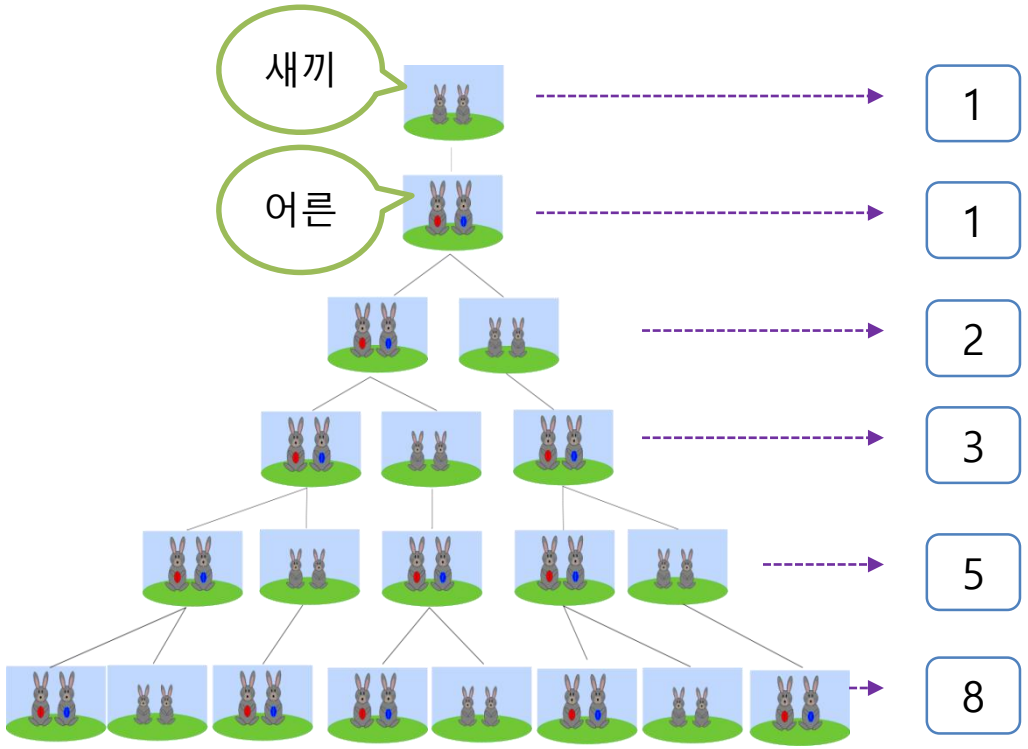
  5 = 2 + 3

  동작 특징

  - 중복 계산되므로 연산 속도가 매우 느림

재귀 알고리즘

- 토끼의 개체수



첫번째 달에 새로 태어난 토끼 한쌍이 있고, 둘째달에 토끼가 커서 그대로 어른토끼 한쌍, 세째달에는 새끼를 한쌍 낳아 어른, 새끼 두쌍, 네째달에는 어른이 새끼를 낳고, 새끼는 어른이 되어 총 세쌍, 이렇게 계속 새끼를 낳고, 죽지 않는다는 가정을 세우면 피보나치의 수가 된다.

재귀 알고리즘

- 피보나치(Fibonacci) 수열

```
int fibo(int n) {

    if (n <= 2)
        return 1;
    else
        return fibo(n - 2) + fibo(n - 1);
}

/*
     n=4, fibo(4), fibo(2) + fibo(3) = 1 + 2 = 3
     n=3, fibo(3), fibo(1) + fibo(2) = 1 + 1 = 2
     n=2, fibo(2), 1
     n=2, fibo(1), 1

*/
```

재귀 알고리즘

- 피보나치(Fibonacci) 수열

```c
int main()
{
    /*
    printf("%d\n", fibo(1)); //1
    printf("%d\n", fibo(2)); //1
    printf("%d\n", fibo(3)); //2
    printf("%d\n", fibo(4)); //3
    */

    //1년동안 토끼의 개체수 출력
    for (int i = 1; i <= 12; i++) {
        printf("%d ", fibo(i));
    }

    return 0;
}
```

```
1 1 2 3 5 8 13 21 34 55 89 144
```

재귀 알고리즘

- **메모이제이션(memorization)**

  메모화 또는 메모기법이라 불린다.

  어떤 문제에 대한 해답을 얻으면 그것을 메모해 둔다.

  - 피보나치 수열 재귀호출
    fibo(5) = fibo(3) + fibo(4)
    fibo(4) = fibo(2) + fibo(3)
    fibo(3) = fibo(1) + fibo(2)

    fibo(2)는 여러번 호출됨 - 중복 계산 발생
    시간복잡도: O(2^n)

재귀 알고리즘

- 메모이제이션(memorization)

```
int memo[50] = { 0 }; // 계산 결과 저장

int fibo(int n) {
    if (memo[n] != 0)
        return memo[n]; // 이미 계산했으면 바로 반환

    if (n <= 2)
        return memo[n] = 1;   // 저장 후 반환
    return memo[n] = fibo(n - 2) + fibo(n - 1);
}
```

재귀 알고리즘

- **비트로 표현할 수 있는 수의 범위**

비트수	표현할 수 있는 범위(십진수)	
1bit	0, 1(0~1)	2^1
2bit	00, 01, 10, 11(0~3)	2^2
3bit	000, 001, 010, 011, 100, 101, 110, 111(0~7)	2^3

16진수	2진수
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111
10	10000

자리 올림 발생

※ 10진수를 2진수로 바꾸기

가중치 방식

$10 = 1010_{(2)}$

8 4 2 1

1 0 1 0($1x2^3 + 1x\ 2^1 \rightarrow 8 + 2$)

재귀 알고리즘

- 십진수를 이진수로 변환하기

```
void printBin(int a){
    if (a == 0 || a == 1)
        printf("%d", a);
    else{
        printBin(a/2);
        printf("%d", a%2);
    }
}
```

재귀 알고리즘

- 십진수를 이진수로 변환하기

```
/*
        a 값                          몫  나머지
   printBin(11), printBin(11/2), 5   1
   printBin(5), printBin(5/2),   2   1
   printBin(2), printBin(2/2),   1   0
   printBin(1), printBin(1/2),   0   1
   //아래서 위로 기록 - 1011(뒤집어 출력)
   //가중치 방식
    11 -> 8 4 2 1
          1 0 1 1
*/
```

재귀 알고리즘

- 십진수를 이진수로 변환하기

```
int main()
{
    int x = 11;
    printBin(x);  //1011

    return 0;
}
```

비트 연산자

- **비트 논리연산자**

```
int num1 = 5;
int num2 = 10;
int result = num1 & num2;
```

```
  num1 : 0 0 0 0 0 1 0 1
& num2 : 0 0 0 0 1 0 1 0
  result : 0 0 0 0 0 0 0 0
```

- **비트 이동 연산자**

```
int num = 5;
num << 2;
```

```
num     : 0 0 0 0 0 1 0 1
num<<2 : 0 0 0 1 0 1 0 0
```

비트 연산자

- **비트 연산자**

```c
//비트 논리 연산
int num1 = 5;     //00000101
int num2 = 10;    //00001010
int result;

result = num1 & num2;
printf("result = %d\n", result);  //00000000

result = num1 | num2;
printf("result = %d\n", result);  //00001111
printf("======================\n");
```

비트 연산자

- **비트 연산자**

```c
//비트 이동 연산
int num3 = 2;   //00000010
int val1, val2, val3;

val1 = (num3 << 1);  //00000100
printf("result = %d\n", val1);

val2 = (num3 << 2);  //00001000
printf("result = %d\n", val2);

val3 = (num3 >> 1);  //00000001
printf("result = %d\n", val3);
```

```
result = 0
result = 15
======================
result = 4
result = 8
result = 1
```

실습 문제

❖ 실습 문제 – 코드 실행의 결과를 먼저 메모장에 적어 보세요.

```
int func(int n) {
    if (n == 1)
        return 1;
    else
        return n * func(n - 1);
}

int main(){
    int i;

    for (i = 5; i >= 0; i--) {
        if (i % 2 == 1)
            printf("func(%d) : %d\n", i, func(i));
    }
    return 0;
}
```

최대 공약수 계산 알고리즘

- 최대공약수 구하기

  최대 공약수(Greatest Common Divisor)는 두 개 이상의 공통 약수 중에서 가장 큰 값을 의미함

  풀이 과정

  1. 두 수중 더 작은 값을 i에 저장한다.
  2. I가 두 수의 공통된 약수인지 확인한다.
  3. 공통된 약수이면 이 값을 결과값으로 돌려주고 종료한다.
  4. 공통된 약수가 아니면 i를 1만큼 감소시키고 2번으로 돌아가 반복한다.

최대 공약수 계산 알고리즘

- 최대공약수 구하기

```
//두 수중 최소값 찾기 함수
int min(int x, int y) {
    return (x < y) ? x : y;
}

//최대 공약수 찾기 함수
int gcd(int a, int b) {
    int i;
    i = min(a, b);
    //printf("%d\n", i);

    while (true) {
        if (a % i == 0 && b % i == 0)
            return i;
        i--;
    }
}
```

최대 공약수 계산 알고리즘

- 최대공약수 구하기

```
/*
    a=4, b=6 인 경우
    i=4
    4%4==0 && 6%4==0, false
    i=3
    4%3==0 && 6%3==0, false
    i=2(최대 공약수)
    4%2==0 && 6%2==0, true
*/

int main()
{
    printf("%d\n", gcd(1, 5)); //1
    printf("%d\n", gcd(4, 6)); //2
    printf("%d\n", gcd(24, 60)); //12
    printf("%d\n", gcd(81, 27)); //27

    return 0;
}
```

최대 공약수 계산 알고리즘

- 유클리드 알고리즘(재귀 호출)

  수학자 유클리드가 발견한 최대공약수에 대한 성질

  특징

  - a와 b의 최대공약수는 'b'와 'a를 나눈 나머지'의 최대공약수와 같다.
  - 어떤 수와 0의 최대공약수는 자기 자신이다. 즉, gcd(n, 0) = n이다.

최대 공약수 계산 알고리즘

- 유클리드 알고리즘(재귀 호출)

```
//최대 공약수 찾기 함수
int gcd(int a, int b) {
    if (b == 0)
        return a;
    else
        return gcd(b, a % b);
}

/*
   - 유클리드 알고리즘
     a=60, b=24
     gcd(60, 24) = gcd(24, 60%24) = gcd(12, 24%12) = gcd(12, 0) = 12
     a=81, b=27
     gcd(81, 27) = gcd(27, 81%27) = gcd(27, 0) = 27

*/
```

최대 공약수 계산 알고리즘

- 유클리드 알고리즘(재귀 호출)

```
/*printf("%d\n", gcd(1, 5)); //1
printf("%d\n", gcd(4, 6)); //2
printf("%d\n", gcd(60, 24)); //12
printf("%d\n", gcd(81, 27)); //27*/

int x, y;
puts("두 수의 최대공약수를 구합니다.");
printf("정수를 입력하세요: ");
scanf("%d", &x);
printf("정수를 입력하세요: ");
scanf("%d", &y);
printf("최대공약수는 %d입니다.\n", gcd(x, y));
```

실습 문제 – 최소공배수 구하기

❖ 아래의 코드를 참고하여 최소공배수 계산 프로그램을 완성하세요

```
/*
    lcm(4, 6)
    i=6
    6%6==0 && 6%4==0, false
    7%6==0 && 7%4==0, false
    8%6==0 && 8%4==0, false
    9%6==0 && 9%4==0, false
    10%6==0 && 10%4==0, false
    11%6==0 && 11%4==0, false
    12%6==0 && 12%4==0, true

*/

int main()
{
    printf("LCM: %d\n", lcm(4, 6)); // 12
    printf("LCM: %d\n", lcm(18, 24)); // 72
    printf("LCM: %d\n", lcm(48, 180)); // 720

    return 0;
}
```