

컴퓨터 알고리즘과 실습 2주차

2016112158 김희수



<문제1>

1-1) 비순환 방식으로 회문 알고리즘 구현

(유사코드)

check\_palindrome(input):

    temp = input

    check = true

    for i to length(input)/2:

        if(temp[i] != arr[length(arr) - i - 1]):

            check = false

            break

    return check

회문의 특징은 문장을 반으로 잘랐을 때 선대칭을 이룬다는 점이다.

즉, 선대칭을 구현하려면 앞에서 i번째 원소와 뒤에서 i번째 원소가 같아야 한다.

그리고 판별의 횟수는 문장을 반으로 줄인 만큼 반복하게 된다.

1-2) 순환방식으로 회문 알고리즘 구현

(유사코드)

recursive\_check(input):

    temp = input

    if(length(input) <= 1):

        return true;

    start\_Substring = substring of input from index 0 to index 1

    end\_Substring = substring of input from length(input)-1 to length(input)

    if(start\_Substring != end\_Substring):

        return false;

    next\_input = substring of input from 1 to length(input)-2

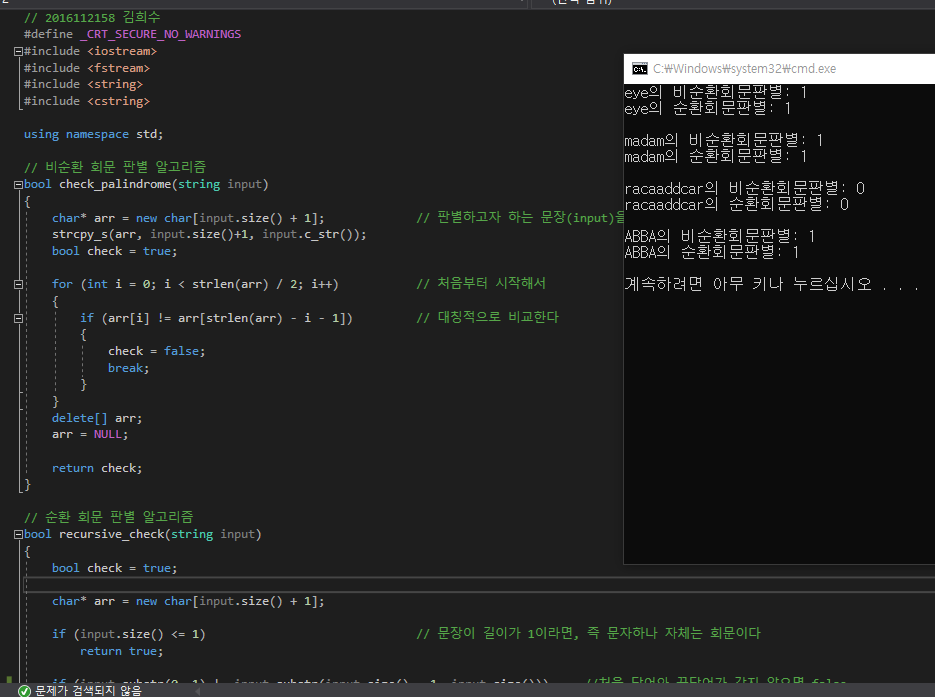
    return recursive\_check(next\_input)

재귀를 이용해 회문판별 할 때는 부분문자열을 이용한다.

역시 선대칭에 대해서 판단해야 하므로 앞의 부분문자열과 뒤의 부분문자열이 같아야 한다.

이를 재귀를 이용하면 결국 맨끝에서 가운데로 판별을 반복하게 된다.

1-1, 1-2의 프로그램 실행화면



<문제2>

2-1

(유사코드)

no\_recursive\_gcd(a, b):

    while(b):

        temp = a % b

        a = b

        b = temp

    return a

a를 b로 나눈 나머지가 0이 될떄까지 반복한다. 나머지가 0이 된다는 것은 나누어 떨어진다는 것이다. 나누어 떨어질떄까지 반복하고 반복이 끝낫을때의 값이 최대공약수이다.2-2

(유사코드)

recursive\_gcd(a, b):

    if b == 0:

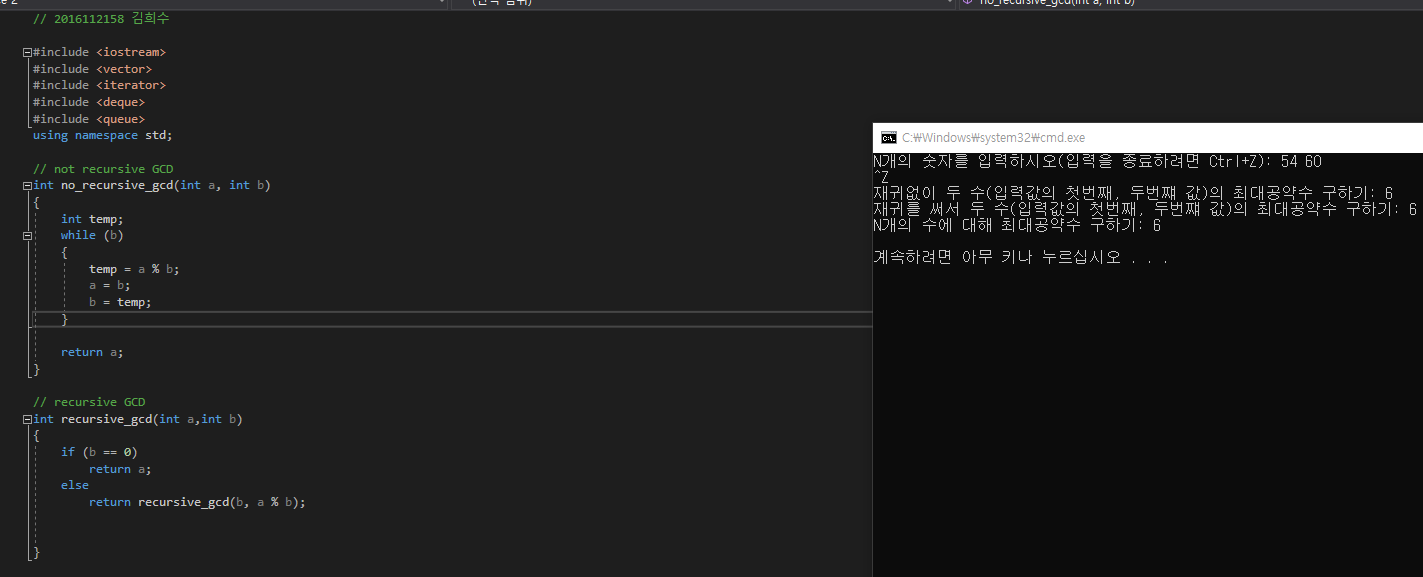
        return a

    else

        return recursive\_gcd(b, a % b)

재귀를 쓰지 않는 부분과 비슷하다. recursive\_gcd(b, a % b)의 내용의 앞에서 while문 안의 내용과 같은 내용이다. 재귀가 끝나는 부분 역시 while이 끝나는 부분과 같다.

2-1, 2-2의 프로그램 실행화면



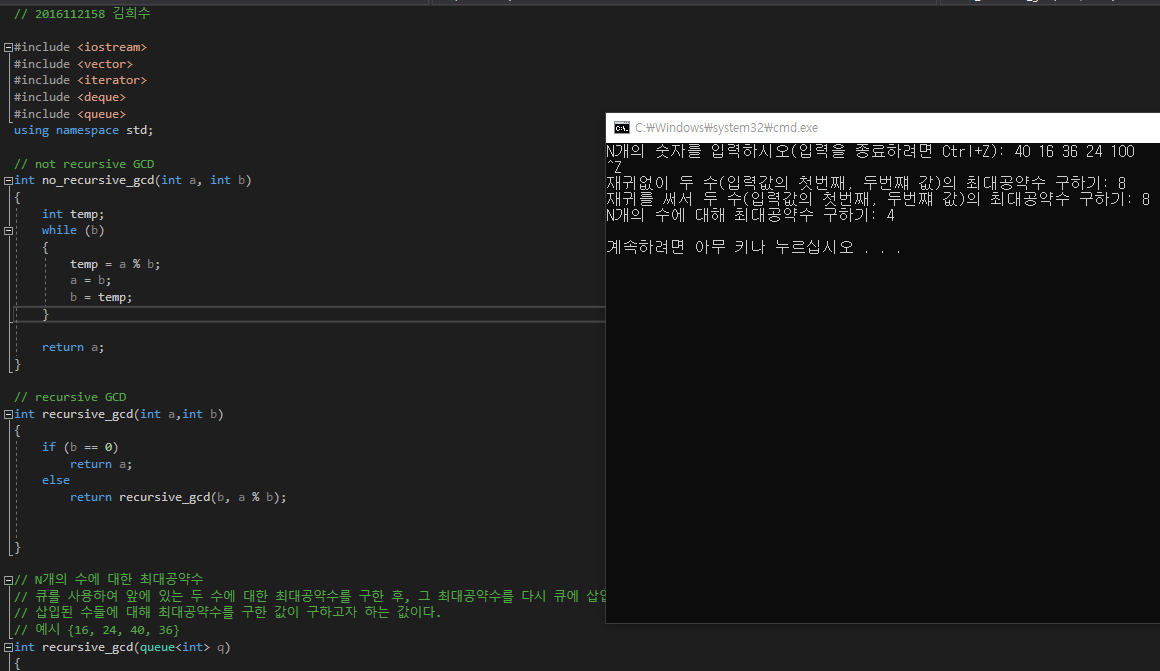
2-3 실행화면

큐를 이용해 입력된 숫자들의 최대공약수를 구하도록 하였다.

큐의 크기가 1일 때 재귀가 끝나는 부분이고, 큐 앞의 두개의 원소의 최대공약수를 큐에 푸시한다. 이를 반복하면 된다. 예를 들어

{16, 24, 40, 36}이 큐에 들어왔다면. 첫번쨰 연산에선 16과 24의 최대공약수 8이 큐에 저장된다.

이 때 큐는 {40, 36, 8}이 된다. 두번째 연산에서 40, 36의 최대공약수 4가 큐에 저장된다. 이 때의 큐는 {8, 4}이다. 이제 세번쨰 연산에서 8,4의 최대공약수 4를 큐에 저장하고 큐의 크기가 1이 되었으니 4를 반환한다.



<3번 문제>

- srand()를 이용했고 시간을 시드로 주어 실행할 때마다 다른 원소들이 배열에 저장되도록 하였다.

- 버블 정렬은 재귀적으로 프로그래밍을 하였는데, 예를 들어서 설명하겠다. arr[] = {1, 2, 3, 4, 5, 6} 이렇게 있다고하면 버블정렬의 첫번째 연산이 끝나면 {2,3,4,5,6,1}이 될 것이다. 그후 bubbleSort(arr,size-1)이 실행되면 맨 끝에 정렬이 된 1은 건들지 않고 {2,3,4,5,6}에 대해서만 버블정렬을 수행한다. 이를 정렬이 끝날떄까지 수행하게 되는 것이다. 부분 배열에 대해서 연산한다고 생각하면 편하겠다.

- 퀵 정렬에 대해서 먼저 파티션 함수는 주어진 배열의 가운데부터 파티션이 이루어지도록 프로그래밍하였다. 퀵정렬 자체에선 스택을 이용하였다. 파티션한 값 pivot에 대해서 pivot왼쪽의 원소에 대해선 원소의 인덱스먼저 푸시한다. pivot오른족의 원소에 대해선 피봇+1먼저 푸시한다

- 버블정렬과 퀵정렬의 소요시간 비교에 대해선 <chrono>헤더를 이용하였다. 퀵정렬이 버블정렬보다 압도적으로 빠름을 알 수 있었다.

