보고서 : 실습과제 1, 실습과제 2, 실습과제 3

2019112096 서준원

I. 실습과제 1

I-1) seoroso_find.m 코드 전문

```
%% 0. 서로소 찾기 스크립트
clear, clc; % 창 클리어, 메모리 초기화
%% 1. 최대값 입력, 인덱스 선언
N = input('서로소 탐색 최대값 N을 입력하시오 : '); % 탐색 범위 지정
index = 0; % 인덱스 초기화
%% 2. 알고리즘 실행
for i=2:N % 2부터 지정 범위까지 실행
    for j = i+1:N % 이중 for문, i의 다음 수부터 N까지
        if my_GCD3(i,j)==1 % i와 i 다음수의 최대공약수가 1이면
            index=index+1; % 인덱스 증가
            seoroso(index,1)=i; % 2차원 배열 사용, 1열은 서로소 쌍의 i
            seoroso(index,2)=j; % 2차원 배열 사용, 2열은 서로소 쌍의 j
        end
    end
end
%% 3. 결과값 표시
disp(N + "까지의 모든 서로소 쌍 : "); disp(seoroso); % 서로소 쌍 출력
disp(N + "까지의 모든 서로소 쌍의 개수 : " + index); % 서로소 쌍 갯수 출력
```

I-2) my_GCD3.m 코드 전문

```
%% 최대공약수 찾기 함수 my_GCD3
function gcd = my_GCD3(a,b)
if (b==0)
    gcd=a;
else
    gcd=my_GCD3(b, rem(a,b)); % 재귀호출
end
```

I-3) 실행 결과

스크립트 실행 시 입력받은 “최대 범위” 에 맞게 서로소 쌍을 구하여 출력하는 프로그램이 완성되었다.

```
서로소 탐색 최대값 N을 입력하시오 : 10
10까지의 모든 서로소 쌍 :
     2      3
     2      5
     2      7
     2      9
     3      4
     3      5
     3      7
     3      8
     3     10
     4      5
     4      7
     4      9
     5      6
     5      7
     5      8
     5      9
     6      7
     7      8
     7      9
     7     10
     8      9
     9     10

10까지의 모든 서로소 쌍의 개수 : 22
```

II. 실습과제 2

II-1) multiple_GCD.m 코드 전문

```
%% 0. 여러 수의 최대공약수 찾기 스크립트
clear, clc; % 메모리 초기화, 창 클리어
i=1;
%% 1. 대상 수 입력
while(1) % 조건값(0) 이 입력될 때 까지 반복
    N=input('최대공약수를 계산할 수들을 입력하시오(입력이 끝났으면 0 입력) : ');
    if N == 0 % 0이 입력되었다면
        i = i-1; % 맨 마지막에 더해진 1을 i에서 제하고
        break; % while루프 탈출
    end % 그렇지 않다면
    inputs(i)=N; % inputs 벡터의 i 번째에 N 대입
    i = i + 1; % i 1 증가
end
%% 2. 알고리즘 실행
if i == 1 % 입력된 수가 하나 밖에 없을 때
    disp("하나만 가지고 최대공약수를 어떻게 구하냐 닌"); % 에러 메시지 출력
elseif i == 2 % 입력된 수가 2개일 때
    j = 1;
    GCD_ans(1) = my_GCD3(inputs(1), inputs(2)); % 원래의 GCD함수 이용
else % 입력된 수가 3개 이상일 때
for j=1:i-2 % j는 j+2까지 참조하므로 i-2까지 루프해야 함
    if j == 1 % j가 1일 때, 즉 최초 실행일 때
      GCD_TEMP = my_GCD3(inputs(j), inputs(j+1)); % 임시 최대공약수 계산
      GCD_ans(j) = my_GCD3(inputs(j+2), GCD_TEMP); % 다음 수와 다시 계산
    else % 이미 한번 최초 실행을 거쳤을 경우
      GCD_TEMP = my_GCD3(GCD_ans(j-1), inputs(j+1)); % 다시 계산한 최대공약수를
      GCD_ans(j) = my_GCD3(inputs(j+2), GCD_TEMP); % 다음 수와 다시 계산
    end
end
end
%% 3. 결과값 표시
if i >= 2 % 에러 메시지를 출력할 상황이 아니라면
disp("입력된 수들의 최대공약수는 " + GCD_ans(j)); % 최대공약수 출력
end
```

II-2) 실행 결과

스크립트 실행 시 입력받은 수들의 최대공약수를 구하는 프로그램이 완성되었다.

```
최대공약수를 계산할 수들을 입력하시오(입력이 끝났으면 0 입력) : 48
최대공약수를 계산할 수들을 입력하시오(입력이 끝났으면 0 입력) : 12
최대공약수를 계산할 수들을 입력하시오(입력이 끝났으면 0 입력) : 96
최대공약수를 계산할 수들을 입력하시오(입력이 끝났으면 0 입력) : 128
최대공약수를 계산할 수들을 입력하시오(입력이 끝났으면 0 입력) : 0
입력된 수들의 최대공약수는 4
```

II-3) 시행착오

```
for j=1:i-2
  GCD_TEMP = my_GCD3(inputs(j), inputs(j+1));
  GCD_ans(j) = my_GCD3(inputs(j+2), GCD_TEMP);
end
```

위 코드는 ***II-1***의 ***%%2. 알고리즘 실행*** 원래 부분이다. 이 상태에서 코드를 실행했을 때, 대략 두 가지 오류를 찾을 수 있었다.

1) 입력받은 수가 1개이거나 2개일 때 오류가 발생함. (인덱스 초과)

입력받은 수가 1개라면 알고리즘 실행이 불가능하고, 2개라면 3개 이상을 가정하고 만든 알고리즘과 적합하지 않으므로, if~else 문을 통해 예외 처리를 하였다.

2) 4개가 넘는 수를 입력했을 때, 결과값이 이상하게 나오는 증상이 있음.

원래 알고리즘의 설계는, 두 수의 최대공약수를 구한 뒤, 그 최대공약수를 다시 그 다음수와 계산하는 설계였다. 하지만 위 코드는 “그 최대공약수” 가 아닌, 이전 수와 계산하도록 설계되어 있었다. GCD_ans 벡터 j-1번째 인덱스의 수를 계산 대상으로 다시 지정하여 오류를 해결하였다.

Ⅲ. 실습과제 3

Ⅲ-1) dec2bin_double.m 코드 전문

```
%% 0. 0보다 크고 1보다 작은 양의 실수를 2진수로 변환하는 스크립트
clear, clc; % 메모리 초기화, 창 클리어
%% 1. 변환 대상 입력
num = input('변환할 1보다 작은 양의 실수를 입력하시오 : '); % 변환 대상 입력
if (num<=0) || (num>=1) % 만약 0 이하거나 1 이상이면
    disp('1보다 작은 양의 실수를 가져오라고!'); % 에러 메시지 출력
    return; % 스크립트 종료
end
%% 2. 변수 선언
bin_index = 1; % 2진수 벡터 인덱스 선언
num_temp = num; % 계산에 사용될 num 변수 선언
MAX_ERR = 10^(-4); % 최대 오차 선언
ERR = 1; % 최초 오차 선언
%% 3. 알고리즘 실행
while (bin_index <= 15) && (ERR > MAX_ERR) % 최대 자리수 : 15, 오차 비교
    ERR = 1; % 오차 1로 초기화
    demical = 0; % 변환된 10진수 초기화
    if num_temp*2 == 1 % 만약 수에 2를 곱한게 1이라면
        bin(bin_index) = 1; % 이진수가 딱 떨어져 나옴.
        bin_index = bin_index + 1; % 배열 인덱스 1 추가 (아래 for문 고려)
        ERR = 0; % while문 탈출 플래그
    elseif num_temp*2 > 1 % 만약 수에 2를 곱한게 1보다 크다면
        bin(bin_index) = 1; % 2진수 벡터에 1 대입
        bin_index = bin_index + 1; % 2진수 벡터 인덱스 1 증가
        num_temp = num_temp*2 - 1; % 다음에 계산할 수는 1이 빠져야 함
    else % 만약 수에 2를 곱한게 1보다 작다면
        bin(bin_index) = 0; % 2진수 벡터에 0 대입
        bin_index = bin_index + 1; % 2진수 벡터 인덱스 1 증가
        num_temp = num_temp*2; % 다음에 계산할 수는 그대로
    end
    for i=1:bin_index-1 % 증가된 인덱스에서 1 빼야 함
       demical = demical + bin(i)*(1/2)^i; % 변환된 2진수를 다시 10진수로
    end
    ERR = num - demical; % 오차 계산
end
%% 4. 결과값 출력
bin % 2진수 벡터 출력
```

II-2) 실행 결과

스크립트 실행 시 입력받은 1보다 작은 양의 실수를 2진수로 변환시켜주는 스크립트가 완성되었다.

```
변환할 1보다 작은 양의 실수를 입력하시오 : 1/3

bin =

     0     1     0     1     0     1     0     1     0     1     0     1
```

```
변환할 1보다 작은 양의 실수를 입력하시오 : 0.6875

bin =

     1     0     1     1
```

```
변환할 1보다 작은 양의 실수를 입력하시오 : 0.5

bin =

     1
```

Ⅲ-3) 아름다운 답 표현 시도 과정

Ⅲ-3-1) dec2bin_double_try.m 코드 전문

```
%% 0. 0보다 크고 1보다 작은 양의 실수를 2진수로 변환하는 스크립트
clear, clc; % 메모리 초기화, 창 클리어
format long;
%% 1. 변환 대상 입력
num = input('변환할 1보다 작은 양의 실수를 입력하시오 : '); % 변환 대상 입력
if (num<=0) || (num>=1) % 만약 0 이하거나 1 이상이면
    disp('1보다 작은 양의 실수를 가져오라고!'); % 에러 메시지 출력
    return; % 스크립트 종료
end
%% 2. 변수 선언
bin_index = 1; % 2진수 벡터 인덱스 선언
num_temp = num; % 계산에 사용될 num 변수 선언
MAX_ERR = 10^(-4); % 최대 오차 선언
ERR = 1; % 최초 오차 선언
answer = 0; % 예쁜 답 변수 선언
%% 3. 알고리즘 실행
while (bin_index <= 15) && (ERR > MAX_ERR) % 최대 자리수 : 15, 오차 비교
    ERR = 1; % 오차 1로 초기화
    demical = 0; % 변환된 10진수 초기화
    if num_temp*2 == 1 % 만약 수에 2를 곱한게 1이라면
        bin(bin_index) = 1; % 이진수가 딱 떨어져 나옴.
        bin_index = bin_index + 1; % 배열 인덱스 1 추가 (아래 for문 고려)
        ERR = 0; % while문 탈출 플래그
    elseif num_temp*2 > 1 % 만약 수에 2를 곱한게 1보다 크다면
        bin(bin_index) = 1; % 2진수 벡터에 1 대입
        bin_index = bin_index + 1; % 2진수 벡터 인덱스 1 증가
        num_temp = num_temp*2 - 1; % 다음에 계산할 수는 1이 빠져야 함
    else % 만약 수에 2를 곱한게 1보다 작다면
        bin(bin_index) = 0; % 2진수 벡터에 0 대입
        bin_index = bin_index + 1; % 2진수 벡터 인덱스 1 증가
        num_temp = num_temp*2; % 다음에 계산할 수는 그대로
    end
    for i=1:bin_index-1 % 증가된 인덱스에서 1 빼야 함
       demical = demical + bin(i)*(1/2)^i; % 변환된 2진수를 다시 10진수로
    end
    ERR = num - demical; % 오차 계산
end
%% 4. 결과값 연산
for i=1:(bin_index-1) % 계산된 항 직전까지만 계산 실행
    answer = answer + bin(i)*((1/10)^i); % 연산
end
answer % long 형태로 답을 출력하면?
disp(num + "을 2진수로 변환하면 " + answer); % disp 함수를 이용하면?
```

III-3-2) dec2bin_double_try 실행 결과

```
변환할 1보다 작은 양의 실수를 입력하시오 : 0.6875

answer =

   0.101100000000000

0.6875을 2진수로 변환하면 0.1011
```

```
변환할 1보다 작은 양의 실수를 입력하시오 : 1/3

answer =

   0.010101010100000

0.33333을 2진수로 변환하면 0.010101
```

```
변환할 1보다 작은 양의 실수를 입력하시오 : 1/10

answer =

     1.100110011000000e-04

0.1을 2진수로 변환하면 0.00011001
```

III-3-3) 탐구

본인은 스크립트를 짜더라도 실사용 가능한 프로그램 형태로 만드는 것을 선호하기 때문에, 과제의 조건으로 걸려있지 않더라도 "직접 입력" 과 "예쁘게 출력"을 고집하는 편이다. 하지만, 이번 과제는 아무리 시도해 보아도 "예쁘게 출력"을 구현할 수 없었다. 문제는 바로 실수의 표현 방식에 있다고 판단된다.

MatLab 공식 사이트[1]에 표기된 숫자형 값의 표현 방식에 따르면, 기본적인 실수 표현 방식인 short형은 소수점 아래 5자리까지 표시하고, 그 이상은 부동 소수점 5자리 방식으로 표현한다고 한다. 그 이상으로 사용 가능한 고정 소수점은 15자리가 있는데, 그것이 long 방식이다.

그래서 위 ***dec2bin_double_try.m*** 에서는 format을 long으로 설정하였으나, 고정 소수점을 15자리까지 늘리니 이번엔 새로운 문제가 발생하였다. 필요 없는 자리까지 모두 0이 붙어 출력되는 것이었다. 심지어, disp() 함수를 이용하여 출력할 때는 자료형 변환이 작동하지 않았다.

또, 한 가지 더 이해가 가지 않는 부분이 있었다. 모든 disp()를 활용한 출력이 소수점 아래 5자리에서 끊긴다면 disp()는 short만 사용 가능하다고 이해할 수 있으나, 두 번째와 세 번째 출력을 보면, 소수점 아래 5자리 이상 출력되었다. 왜 disp() 함수에서 이러한 결과가 발생하였는지 탐구를 몇 번이고 시도해 보았으나, 아쉽게도 밝혀내지 못하였다.

---

1) https://kr.mathworks.com/help/matlab/matlab_prog/display-format-for-numeric-values.html?s_tid=mwa_osa_a