강의명: 자료구조

숙제 번호: 3

숙제 제목: Recursion(순환)

학생 이름: 김성현

학번: 201910783

1. Fibonacci(피보나치)

1.1

long fib(long n)

{

if (n == 0) return 0;

else if (n == 1) return 1;

else return (fib\_log(n-2) + fib\_log(n-1));

}

이렇게 작성한 이유:

fibonacci수열은 작년 C프로그래밍 시간에 처음 알았고‘반복’방법을 통해 구현해보았습니다. 하지만 Recursion(순환)을 통한 fibnoacci수열은 이번에 수업을 들으면서 처음 알게 되었습니다. 피보나치수열이란 것은 본론적으로 전항과 전전항을 더하는 것이기 때문에 이 전항과 전전항도 똑같이 다시 피보나치에 넣어주게 된다면 다시 각 항마다 전항과 전전항을 더하고, 이것이 계속 점층적으로 이어져 나가기 때문에 전항과 전전항을 피보나치함수에 다시 집어넣는 형식, 순환의 형식으로 함수를 만들었습니다. 그 전에 피보나치 수열의 시작은 0, 그리고 그 다음 두번째 항은 1이기 때문에 if문/elseif문을 사용하여 각각 0과 1을 반환하게 하여 구현하도록 하였습니다. 그 다음 항부터는 위의 설명한 대로 전항과 전전항들의 피보나치 순환을 위해 else라는 그 외 경우 모두를 포함시켜 순환토록 하였습니다. 여기서 왜 fib함수가 아닌 fib\_log함수를 썻냐고 물을 수 있는데 fib\_log함수는 fib함수를 다시 콜하면서 중간 과정들을 보기 위한 printf가 들어 있기에 fib\_log함수를 사용하였습니다.

1.2

s1910783@oak:hw03$ ./fib

n=5

-> fib(5)

|-> fib(3)

||-> fib(1)

||<- fib(1)=1

||-> fib(2)

|||-> fib(0)

|||<- fib(0)=0

|||-> fib(1)

|||<- fib(1)=1

||<- fib(2)=1

|<- fib(3)=2

|-> fib(4)

||-> fib(2)

|||-> fib(0)

|||<- fib(0)=0

|||-> fib(1)

|||<- fib(1)=1

||<- fib(2)=1

||-> fib(3)

|||-> fib(1)

|||<- fib(1)=1

|||-> fib(2)

||||-> fib(0)

||||<- fib(0)=0

||||-> fib(1)

||||<- fib(1)=1

|||<- fib(2)=1

||<- fib(3)=2

|<- fib(4)=3

<- fib(5)=5

함수 수행 횟수:

Fib(0): 3번

Fib(1): 5번

Fib(2): 3번

Fib(3): 2번

Fib(4): 1번

Fib(5): 1번

1.3

n=6일 때 fib(3)이 수행한 횟수: 3번

Recursive 함수fib의 성능 평가:

Recursive 함수fib의 성능은 한마디로 말하자면 ‘좋지 않은 상태’이다. 그 이유로는 당연히 같은 함수의 계속 반복적인 수행때문인데, 이는 순환으로 인한 ‘비효율성’에서 야기된 것이다. n=6인 경우 fib(6)->fib(5) / fib(4) 그리고 다시 fib(5)->fib(4) / fib(3) 또 fib(4)->fib(3) / fib(2)등 밑으로 가면 갈수록 수행해야할 것들이 계속해서 쌓여가는 형태이기 때문에 당장 fib(4)나 fib(3)같은 경우 한번 수행 시작한 함수의 종료도 그 밑의 추가적으로 call&return되는 함수들로 인해 상당한 시간이 걸릴 뿐더러, 이들 또한 다른 함수에서 불리우기에 다시 처음부터 계속해서 계산하는, 특히 fib(0), fib(1), fib(2)같은 뒷편에 호출되는 함수들은 계속 반복해서 5,6번씩 호출되는, 같은 함수가 반복적으로 불리우고 이를 계속 처음부터 다시 계산해야 한다는 문제점이 발생하고, 이는 n이 증가하면 증가할수록 심각해져 시간적인 낭비를 야기하여 결국에는 성능이란 것이 좋지 않다고 평가하게 되는 원인이 된다.

그렇다면 이 중복을 줄이는 방법에는 무엇이 있을까, 일단 당장 떠오르는 건 recursion(순환)이 아닌 ‘반복’을 통해 n에 대한 피보나치 수열을 알아내는 것이다. For문 반복을 통해 임시로 저장시킨 전항과 전전항에 해당하는 값들을 더하고, 새로운 값을 전항, 원래 전항이었던 것을 전전항으로 임시로 저장하여 그 다음 반복에서 다시 그 둘을 더하는 이런 행위를 반복한다면 적어도 순환에서의 중복에 의한 성능문제는 어느정도 해소할 수도 있다. 물론 n의 크기가 작으면 순환 쪽이 오히려 임시로 저장 공간을 쓰는 반복보다 더 효율적일 수도 있다. 이럴 경우에는 굳이 반복을 쓰지 않고 자주 반복해서 나오는 함수들, 예를 들어 fib(0), fib(1), fib(2)같은 낮은 위치의 함수값들을 미리 저장해놓아 호출될 때마다 다시 순환이 아닌 미리 저장된 값들을 쓰는 것도 성능면에서 나쁘지 않은 효과를 발휘할 것이라는 것이 내 개인적인 견해이다.

2. Hanoi(하노이)

2.1

void hanoi(int n, char from, char tmp, char to)

{

if (n==1) {

printf("원판 1을 %c에서 %c로 옮긴다. \n", from, to);

}

else {

hanoi(n - 1, from, to, tmp);

printf("원판 %d을(를) %c에서 %c로 옮긴다. \n", n, from, to);

hanoi(n - 1, tmp, from, to);

}

}

이렇게 작성한 이유:

하노이 탑(Hanoi tower) 문제는 이번에 교수님 수업인 순환에서 처음 배운 문제였습니다. 그렇지만 비슷한 문제를 고등학교, 그리고 대학 1학년 확률과 통계 때 목격한 적이 있기 때문에 이때 경험과 교수님 수업 때 배운 풀이를 적용하여 함수를 작성하였습니다. 제일 밑의 1개와 그 위로의 n-1개로 나누어 n-1개를 최종적으로 A->C로 옮기는, 구체적으로는 A->B로 임시적으로 옮기고, 다시 여기서 B->C로 옮기고 이 각각의 두 단계에서는 남는 C와 A를 임시로 쓰며 옮기는 그런 논리를 이 함수에 적용하였습니다. 먼저 마지막 1개를 옮기는 것을 하기 위해 n==1일 때 from(A)에서 to(C)로 옮기도록 하기 위해 if문을 작성하였습니다. 그 다음에는 n-1개를 다시 hanoi함수로 recursion(순환)시키면서 다시 n-1/1개씩 나누어 지게 하며 from(A)에서 tmp(B)로 임시거처 to(C)를 이용해 이동하도록 from, to, tmp순서로 넣었고, 똑같이 n-1개/1개씩 나누며 그 다음 tmp(B)에서 to(C)로 임시거처 from(A)를 이용해 이동하도록 tmp, from, to순서로 넣었습니다.

2.2

s1910783@oak:hw03$ ./hanoi

n=4

원판 1을 A에서 B로 옮긴다.

원판 2을(를) A에서 C로 옮긴다.

원판 1을 B에서 C로 옮긴다.

원판 3을(를) A에서 B로 옮긴다.

원판 1을 C에서 A로 옮긴다.

원판 2을(를) C에서 B로 옮긴다.

원판 1을 A에서 B로 옮긴다.

원판 4을(를) A에서 C로 옮긴다.

원판 1을 B에서 C로 옮긴다.

원판 2을(를) B에서 A로 옮긴다.

원판 1을 C에서 A로 옮긴다.

원판 3을(를) B에서 C로 옮긴다.

원판 1을 A에서 B로 옮긴다.

원판 2을(를) A에서 C로 옮긴다.

원판 1을 B에서 C로 옮긴다.

\*Disk 이동 횟수: 15회.

끝.