#include<iostream>

using namespace std;

int APower(int N, int p)

{

if (p == 0)

return 1;

if (p == 1)

return N;

if (p % 2 == 0)

{

return APower(N, p / 2)\*APower(N, p / 2);

}

else {

return APower(N, (p - 1) / 2)\*APower(N, (p - 1) / 2)\*N;

}

}

class Colunm

{

int arr[100];

public : int INDEX;

char name;

public :Colunm(char c)

{

INDEX = -1;

name = c;

}

public :void Push(int n)

{

arr[++INDEX] = n;

}

public :int Pop()

{

return arr[INDEX--];

}

public :int Seek()

{

return arr[INDEX];

}

};

void DetermineMove(Colunm &A, Colunm &B)

{

if (B.INDEX == -1)

{

B.Push(A.Pop());

cout << A.name << "---->" << B.name << endl;

}

else if (A.INDEX == -1)

{

A.Push(B.Pop());

cout << B.name << "---->" << A.name << endl;

}

else if (A.Seek() >B.Seek())

{

A.Push(B.Pop());

cout << B.name << "---->" << A.name << endl;

}

else {

B.Push(A.Pop());

cout << A.name << "---->" << B.name << endl;

}

}

void Professorhanoi(int n)

{

int k = n;

Colunm A('A');

Colunm B('B');

Colunm C('C');

int count = 0;

for (int i = n; i > 0; i--)

{

A.Push(k--);

}

if (n % 2 == 1)

{

while (1)

{

DetermineMove(A, C);

count++;

if (count == (APower(2, n) - 1))

break;

DetermineMove(A, B);

count++;

if (count == (APower(2, n) - 1))

break;

DetermineMove(B, C);

count++;

if (count == (APower(2, n) - 1))

break;

}

}

else if(n%2==0)

{

while (1)

{

DetermineMove(A, B);

count++;

if (count == (APower(2, n) - 1))

break;

DetermineMove(A, C);

count++;

if (count == (APower(2, n) - 1))

break;

DetermineMove(B, C);

count++;

if (count == (APower(2, n) - 1))

break;

}

}

}

int main()

{

Professorhanoi(5);

system("pause");

}

Apower라는 함수는 수업시간에 구현한 지수승 계산 함수로 Disk의 개수가 n개일 때 총 실행횟수가 2^n-1인 것을 이용하기위해 사용하였다.

이 문제를 해결할 때 Colunm Class를 만들어 해결하고자 한다. Colunm 클래스로 구현하고자 한 것은 Stack이라는 자료구조 이다. 이 클래스 안에는 Pop, Push가 있으며 제일 최근에 들어간 원소를 확인 할 수 있는 Seek함수가 있다. 즉 Colunm Class의 인스턴스 A, B, C를 만들어 하노이 타워에 있는 출발지, 중간지점, 도착지점 기둥을 표현한다. DetermineMove 함수는 Direction만 바뀌며 수행이 끝날때 까지 반복되는 Case들에서 Direction을 결정해 주기위해 사용하는 함수인데 이 방향을 정할 때 Stack구조를 이용하여 Hanoi의 규칙과 맞게 방향을 정해준다. DetermineMove 함수를 활용하여 방향을 정하는 원리, 방법은 밑에서 설명하겠다.

하노이의 타워는 홀수, 짝수일때 다른 케이스가 반복된다.

For an even number of disks:  
make the legal move between pegs A and B (in either direction),  
make the legal move between pegs A and C (in either direction),  
make the legal move between pegs B and C (in either direction),  
repeat until complete.  
For an odd number of disks:  
make the legal move between pegs A and C (in either direction),  
make the legal move between pegs A and B (in either direction),  
make the legal move between pegs B and C (in either direction),  
repeat until complete.

과제에 제시된 순서로 각각의 Direction만 바뀌며 수행이 완료될 때 까지 반복되어 수행된다. 수행횟수는 2^n-1번이다. 그러므로 반복되는 경우에서 Direction만 정해주면 손쉽게 구할 수 있다. Direction을 구하는 원리를 알아보자. 예를들어 짝수의 경우에서 첫번째 순서인 A->B이냐 B->A이냐 를 결정할 때 둘중의 하나밖에 실행 할 수 없다. 왜냐하면 하노이의 규칙에 따라 큰Disk는 작은Disk위에 올라 갈 수 없고, A,B두 기둥중 한 기둥이 비었을때는 선택지가 둘중 하나밖에 없기 때문이다. 한번의 실행마다 두개의 방향을 결정하기 위해 자료구조중 Stack을 활용하였다. Stack은 First in Last Out 구조 이므로 하노이의 타워와 구조가 비슷해서 유용하게 활용할 수 있다. Disk의 개수가 n개일 때 가장 작은 Disk부터 각각 1부터 n까지 번호를 매긴다. 이를 제일 큰 Disk부터 제일 작은 Disk까지 즉 n번부터 1번까지 출발기둥을 뜻하는 Colunm A에 전부 Push한다. 이제부터 짝수, 홀수일때의 정해진 실행순서로 끝날때까지 반복 시행하되 각 실행에서 Direction을 결정할 때 이 Stack 구조를 이용한다. 두개의 기둥이 있을 때 한 기둥이 비어있으면( Instance의 스택이 비어있으면 ) 선택지는 하나밖에 없으므로 그 선택지를 택해 실행한다. 두개의 기둥이 비어있지 않은 경우 Colunm 클래스의 Seek()함수를 활용하면 Stack의 맨위에 있는 Disk번호(=크기)를 알 수 있고 이를 비교해서 각 기둥의 맨위의 Disk의 크기가 작은쪽에서 큰쪽으로 이동시켜주고, 작은쪽 기둥의 맨위 Disk를 Pop() 해서 큰쪽 기둥에 Push 해준다. Stack구조이므로 제일 작은 것이 맨위로 갈수밖에 없다. 이 의미는 기둥에서 Disk를 빼는 연산 즉, Pop()을 했을 때 그 기둥에서 제일 작은 Disk가 나올 수 밖에 없다는 것을 의미한다.즉 이렇게 매 연산마다 두개의 기둥을 의미하는 각각의 Colunm 클래스의 Instance를 이용해 각 기둥의 맨위에 있는 Disk의 크기 및 비어있는지 아닌지(Stack의 Index가 -1인지 아닌지) 를비교 할 수 있고 그에따라 Direction을 선택하여 2^n-1번 반복 연산을 진행해 주면 하노이의 타워를 해결할 수 있다. 홀수 ,짝수의 경우는 반복되는 Case만 다를 뿐, Direction을 정하는 원리는 같으므로 각각의 Case를 나눠서 실행해주면 문제될 것이 없다.