보고서

주제 : 하노이탑

과 목 명	:	고급프로그래밍
제출일자	:	2021년 4월 8일
학 과	:	소프트웨어학부
학 번	:	2020203002
이 름	:	박 상 천

KwangWoon University

목 차

Ⅰ. 코드의 흐름

```
int main()
{
    int a = 0;
    a = intro();
```

코드의 자세한 설명은 아래 3. 설계 이론에서 설명을 하고 이번 1. 코드의 흐름에서는 큰 관점에서 이 코드가 어떤 식으로 흘러가는지를 설명하겠다.

먼저 main 함수를 시작으로 설명을 하겠다. 처음 main 함수를 들어가면서 intro 함수의 반환값을 a에 초기화를 시켜 이것을 가지고 인터페이스 화면에서 사용자의 입력에 따라 다른 기능을 사용할 수 있도록 하였다.

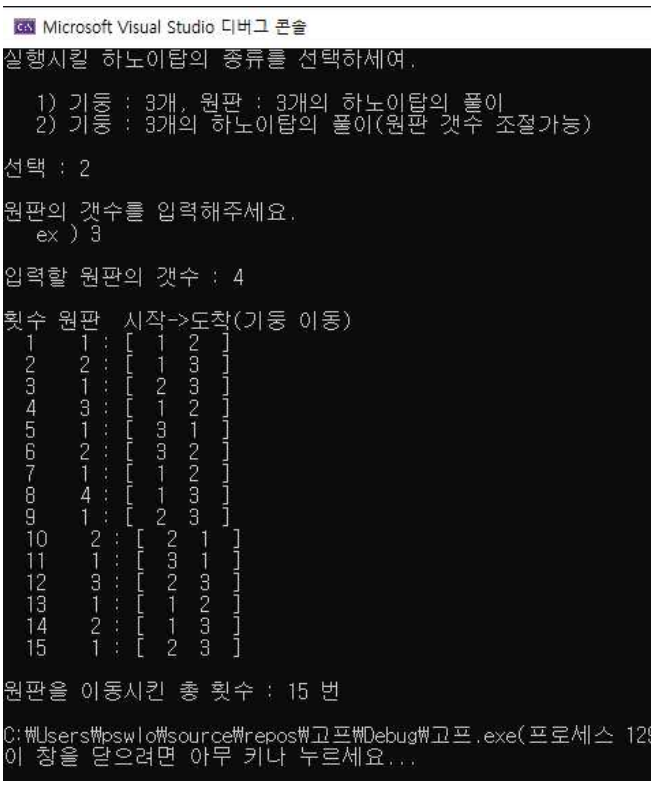
```
if (a == 1)  // intro가 1로 반환이 되었을 때
{
    int cnt = 0;
    int n = 3; // 최소로 옮기는 수는 2^n - 1
    cout << "횟수 원판  시작->도착(기둥 이동)" << endl;
    vector<vector<int>> re = sol(n);
    for (vector<int> v : re)
    {
        cnt++; // 실행횟수를 기록
    }
    cout << endl;
    cout << "원판을 이동시킨 총 횟수 : " << cnt << " 번";
    cout << endl;
    return 0;
}
```

그렇게 intro 함수에서 1이라는 반환값을 받으면 원래 설정이 되어있던 tower 3개가 있고, 한 tower에 3개의 disk이 있는 상태에서 시작하는 것을 보여준다.

```
else if (a == 2) // intro가 2로 반환이 되었을 때
{
    int cnt = 0;
    int n = 0; // 최소로 옮기는 수는 2^n - 1
    cout << "원판의 갯수를 입력해주세요." << endl;
    cout << "   ex ) 3" << endl << endl;
    cout << "입력할 원판의 갯수 : ";
    cin >> n;
    cout << endl;

    cout << "횟수 원판  시작->도착(기둥 이동)" << endl;
    vector<vector<int>> re = sol(n);
    for (vector<int> v : re)
    {
        cnt++; // 실행횟수를 기록
    }
    cout << endl;
    cout << "원판을 이동시킨 총 횟수 : " << cnt<<" 번";
    cout << endl;
    return 0;
}
```

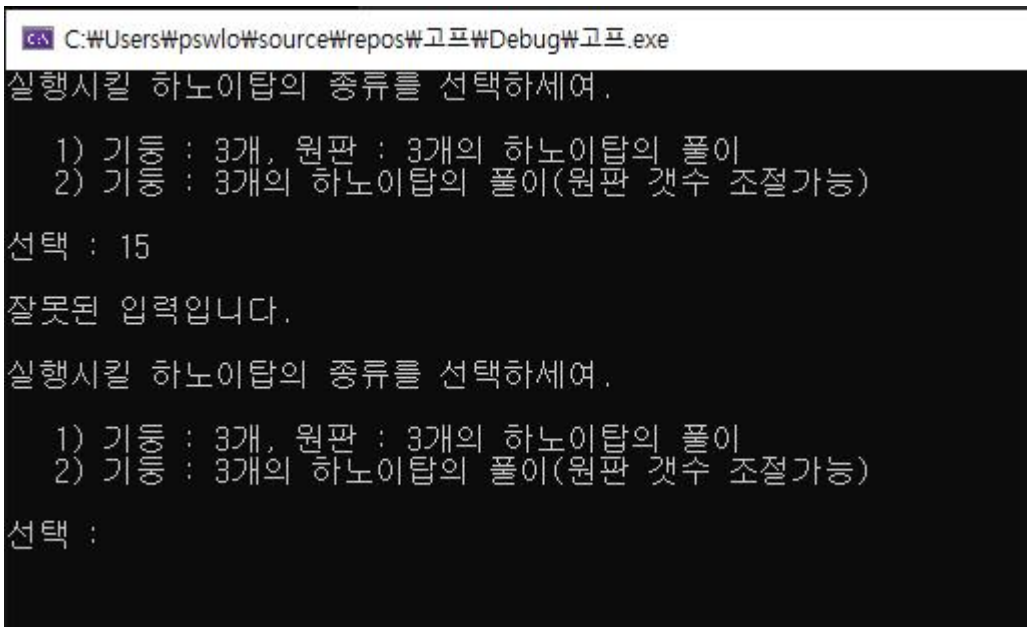
만약 intro 함수에서 1이 아닌 2로 값이 반환이 된다고 가정해보자. 그럼 tower은 3개로 설정이 되어있고, disk은 사용자가 입력한 값을 받아 그 값대로 disk의 개수가 정해진다. 그리고 정해진 disk은 한 tower에서 시작한다는 가정으로 실행이 되며, 그 과정과 횟수들이 출력

이 된다.



위의 캡처처럼 말이다.

만약 intro 함수에 1이나 2가 아닌 다른 값을 입력을 받았을 때(오류가 발생하였을 때)는 "잘못된 입력이다."라는 문구가 뜨면서 다시 사용자로부터 입력을 받을 수 있도록 설계하였다.



Ⅱ. 추가 기능

1. 하노이탑의 알고리즘을 적용시켜 자동화 시켰다.

- 사용자에게 값을 받아 그 값에 따라 disk을 이동시키게 된다면 적은 수의 disk을 옮길 때는 쉽게 할 수 있지만, 5,6,7~~ 그 값이 커진다면 직접 disk을 이동시키며 퍼즐을 푸는 것은 거의 불가능할 것이다. 그렇기 때문에 그런 단점을 보완하기 위해서 하노이탑의 규칙을 알아내 그것을 프로그램에 적용시켜서 자동화를 시켰다. 그 덕분에 우리는 많은 수의 disk을 하노이탑의 규칙에 맞춰서 쉽게 해결할 수 있게 되었다.

2. 인터페이스 화면을 구축하여 사용자의 선택지를 늘렸다.

-필자는 하노이탑이라고 하면 대부분 3개의 tower와 3개의 disk이 가장 먼저 떠올라 그것을 기반으로 코드를 설계하기 시작하였다. 하지만 코드를 짜면서 더 적거나, 더 많은 disk들을 사용하면 좋지 않을까라는 생각이 들었다. 그리고 그 내용을 실현시키기 위해서 인터페이스 화면을 구축하였다. 인터페이스에서는 1. tower 3개, disk 3개인 하노이탑, 2. tower 3개, 사용자 입력에 따른 disk의 개수 이렇게 2가지의 선택지를 만들어 사용자가 직접 선택할 수 있도록 설계하였다. 이렇게 함으로써 3개 disk뿐 아니라 다른 개수의 disk들은 어떤 방식으로 옮기는 것이 가장 좋은 것이지 알 수 있다.

Ⅲ. 설계 이론

```
void PrintTowers(vector<vector<int>>& v,int n, int from, int to)
{
    static int p=1; // static함수를 정의하여 실행횟수를 기록한다
    vector<int> tmp;
    tmp.push_back(from);
    tmp.push_back(to);
    v.push_back(tmp); // 시작봉과 도착봉에 대한 정보를 하나씩 담고, 모두를 v에 담는다
    cout << "   " << p++ << "     " << n << " :";
    cout << " [   " << from<<"   "<<to << "   ] " << endl;
}
```

먼저 PrintTowers 함수이다. PrintTowers 함수에서는 먼저 static 함수를 활용하여 이후에 여러 번 반복이 되어 Tower를 출력시키는데 그 과정에서 한번 초기화되고 그 뒤로부터는 초기화되지 않고, 증가할 수 있게 만듦으로써 Disk가 움직인 횟수를 기록하게 하였다. 그리고 이중 벡터 vector<vector<int>>를 활용하여 일반 벡터 vector<int>를 활용하여 from과 to를 각각 받고 그 값들을 모두 이중 벡터에 저장하게 하였습니다. 그리고 이중 벡터에 저장한 값들을 cout을 이용하여 출력시켰다.

```
void MoveDisk(vector<vector<int>>& v, int n, int from, int to, int ex)
{
    if (n == 1) // 원판이 1개일때 실행한다
    {
        PrintTowers(v,n, from, to);
    }
    else // 그외의 상황에서는 재귀함수를 이용한다
    {
        MoveDisk(v, n - 1, from, ex, to); // 위에서부터 n - 1개를 1번에서 2번으로 옮긴다
        PrintTowers(v,n, from, to); // 나머지 바닥에 있는 한 개를 1번에서 최종 3번으로 옮긴다
        MoveDisk(v, n - 1, ex, to, from); // 2번으로 옮긴 n - 1 개를 2번에서 최종 3번으로 옮긴다
    }
}
```

MoveDisk 함수는 하노이탑의 규칙을 적용시킨 함수이다.

1. 맨 아래 판을 제외한 나머지를 모두 ex로 옮긴다.
2. 맨 아래 판을 to로 옮긴다.
3. 남은 ex에 있던 판을 모두 to로 옮긴다.

이런 큰 틀에서 하노이탑이 실행이 되는데 그것을 코드로 옮겨놓은 것이 바로 MoveDIsk 함수이다.

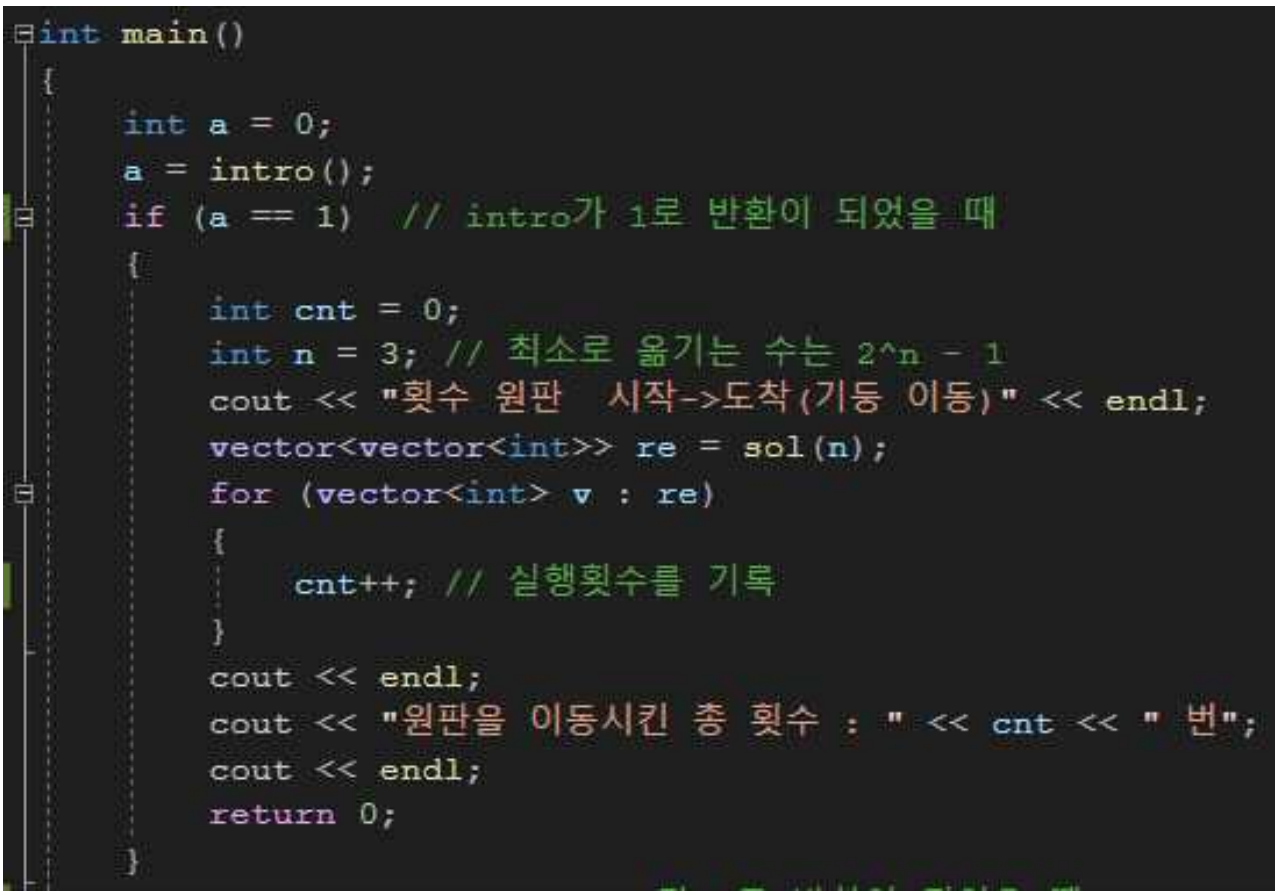
```
vector<vector<int>> sol(int n)
{
    vector<vector<int>> re;

    MoveDisk(re, n, 1, 3, 2);

    return re;
}
```

다음은 vector 함수이다. vector 함수에는 from tower와 to tower를 저장하기 위해서 이중 벡터를 선언하였다. 그리고 위에 설명한 MoveDIsk 함수에 선언한 변수 re와 입력받는 n 그리고 from에는 1, to에는 3, ex에는 2라는 숫자를 매겨줌으로써 disk들을 옮길 때 값으로써 1,2,3이 들어가며 코드가 작동을 한다.

```
int intro()  //하노이탑을 선택하는 화면 함수이다
{
    double n = 0;
    cout << "실행시킬 하노이탑의 종류를 선택하세여. " << endl << endl;
    cout << "   1) 기둥 : 3개, 원판 : 3개의 하노이탑의 풀이" << endl;
    cout << "   2) 기둥 : 3개의 하노이탑의 풀이(원판 갯수 조절가능)" << endl << endl;
    cout << "선택 : ";
    cin >> n;
    cout << endl;
    if(n==1) // 1 선택시
    {
        return 1;
    }
    else if (n == 2) // 2 선택시
    {
        return 2;
    }
    else // 나머지 오류에 대해서 처리
    {
        cout << "잘못된 입력입니다. " << endl << endl;
        intro();
    }
}
```

intro 함수이다. cout으로 사용자에게 어떤 선택지가 있는지를 출력하고, 사용자로부터 cin을 이용하여 선택지를 입력받는다. 만약 1을 입력받으면 반환 값으로 1을, 2를 입력받으면 반환

값으로 2를 반환하도록 하였다. 그리고 나머지 입력들에 대해서는 cout을 이용해 "잘못된 입력입니다"라는 문구를 출력하고 다시 선택지를 입력받을 수 있도록 intro 함수를 다시 실행하도록 만들었다. 사용자로부터 1또는 2를 입력받았을 때의 반환 값들은 main 함수에 그 반환값에 맞게 실행되도록 하였다.

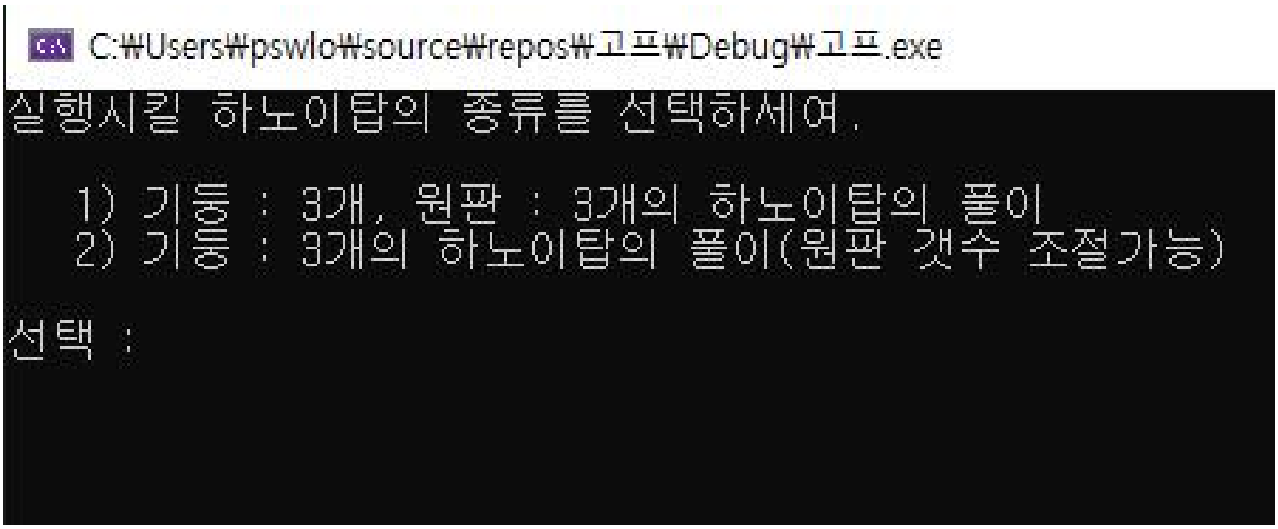


```
int main()
{
    int a = 0;
    a = intro();
    if (a == 1)  // intro가 1로 반환이 되었을 때
    {
        int cnt = 0;
        int n = 3; // 최소로 옮기는 수는 2^n - 1
        cout << "횟수 원판  시작->도착(기둥 이동)" << endl;
        vector<vector<int>> re = sol(n);
        for (vector<int> v : re)
        {
            cnt++; // 실행횟수를 기록
        }
        cout << endl;
        cout << "원판을 이동시킨 총 횟수 : " << cnt << " 번";
        cout << endl;
        return 0;
    }
```

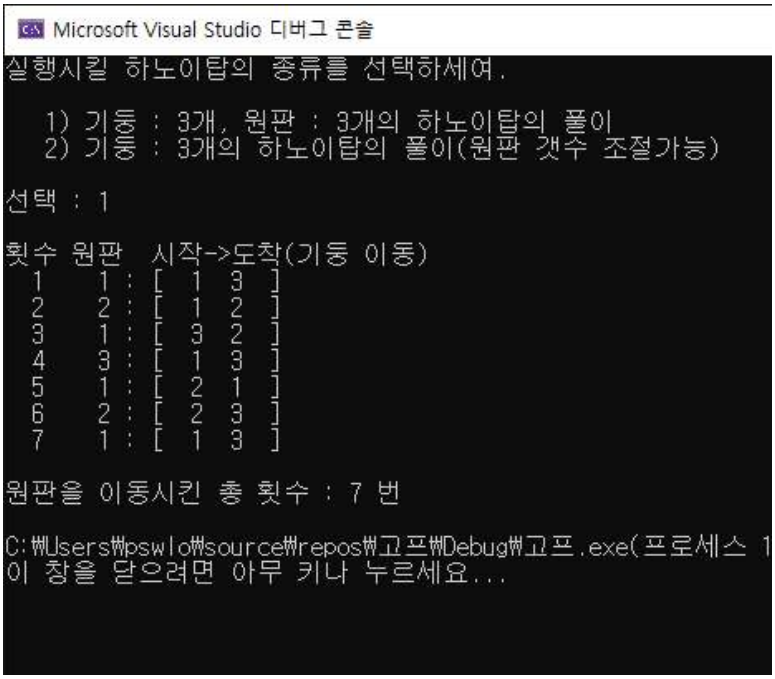
main 함수이다. 먼저 a를 선언한 후, 앞에서 설명한 intro 함수를 실행시켜 사용자가 입력하는 값에 따라 나온 반환값을 a에 초기화시키도록 하였다. 그리고 그 후에 a 값에 따라 3개의 disk으로 할지, 사용자에게 또 입력받은 disk의 개수로 진행할지 if 문을 이용하여 각각의 코드를 설계하였다. 위의 사진은 a에 1이 초기화되어있을 때이다. 그러면 disk의 개수 n=3으로 설정되어 위에서 설명한 vector 함수에 들어가고, 다시 vector 함수에서 MoveDisk 함수로 이동하여 전체적인 코드가 실행이 되는 것이다. 그럼 3개의 disk가 자동적으로 to tower에 이동이 된다. 그리고 disk가 움직인 총 횟수는 vector 함수가 돌 때마다 cnt를 1씩 증가시켜 기록하게 하였다.

```
else if (a == 2) // intro가 2로 반환이 되었을 때
{
    int cnt = 0;
    int n = 0; // 최소로 옮기는 수는 2^n - 1
    cout << "원판의 갯수를 입력해주세요." << endl;
    cout << "   ex ) 3" << endl << endl;
    cout << "입력할 원판의 갯수 : ";
    cin >> n;
    cout << endl;

    cout << "횟수 원판  시작->도착(기둥 이동)" << endl;
    vector<vector<int>> re = sol(n);
    for (vector<int> v : re)
    {
        cnt++; // 실행횟수를 기록
    }
    cout << endl;
    cout << "원판을 이동시킨 총 횟수 : " << cnt<<" 번";
    cout << endl;
    return 0;
}
```

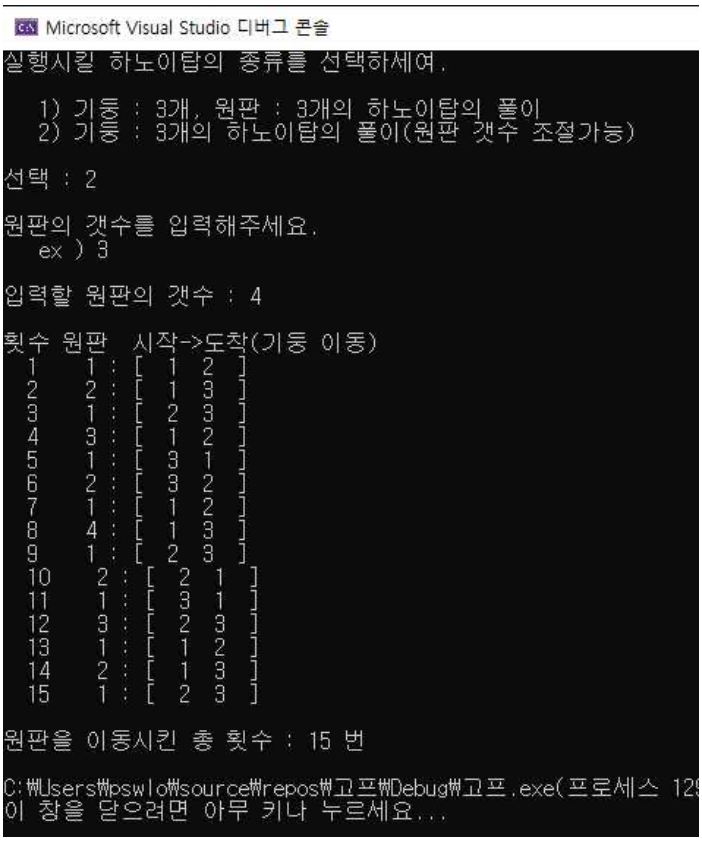
main 함수의 나머지 부분이다. 여기서는 a에 2라는 값이 들어갔을 때 실행이 되는 부분이다. 먼저 사용자에게 cin을 이용하여 disk의 개수를 입력받고 그 값을 n에 저장한다. 그 뒤로 실행하는 방법은 위 main 함수 윗부분에서 실행하는 부분과 동일하게 진행이 된다.



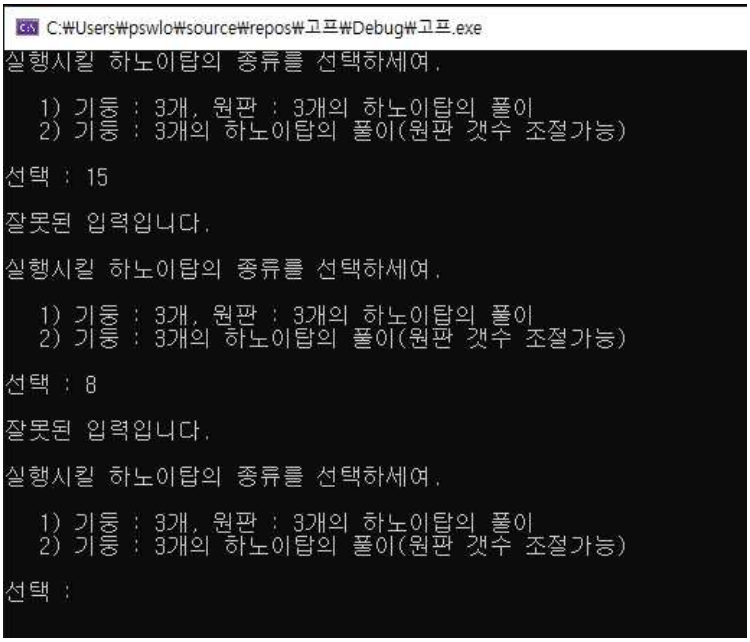
이 코드를 실행시키면 나오는 첫 번째 화면이다. 여기서 1을 입력하면



이런 실행 화면이 나오면서 disk가 3개, tower가 3개일 때 disk가 움직인 횟수, 각 횟수마다 움직인 disk의 번호, tower가 움직이는 과정을 할 수 있다.



이 실행 화면은 처음 화면에서 2를 입력하고 disk의 개수를 4로 입력하였을 때 나오는 실행 화면이다.



만약 위 실행 화면처럼 처음 실행 때 15, 8과 같이 설정이 되어 있지 않는 입력을 받으면 오류로 처리하여 사용자에게 다시 선택지를 입력받을 수 있도록 하였다.

Ⅳ. 느낀점

평소 코딩을 할 때는 원리에 대해서는 거의 생각을 하지 않고, 알고 있는 지식을 바로바로 적용하면서 코딩을 해왔다. 그러면 지금까지 해왔던 거의 모든 문제들이 풀렸기 때문이다. 하지만 이번 하노이탑 코딩은 그렇지 않았다. 이번 과제를 통해서 짧고, 간단하게만 생각해서는 그 원리를 이해할 수 없는 문제를 만나게 된다면 코딩이 막힌다는 것을 깨닫게 되었다. 그래서 이번 과제에서는 직접 각 disk의 개수마다 어떤 식으로 disk이 움직이는지 그려보고, 그것들에는 어떤 규칙이 있는지 깨달으려고 노력을 하였다. 그 과정에서 필자는 좀 더 프로그래머에 가까워진 거 같았다. 이런 깨달음을 얻은 반면 반대로 아쉬운 점들도 있었다.

먼저 사용자로부터 어디에 있는 disk을 어디로 옮기려는 것인지 등과 같이 좀 더 사용자가 개입이 많은 코드를 짜지 못한 것이 많이 아쉬웠다. 이점은 처음 코딩을 할 때는 거의 느끼지 못했지만, 하노이탑을 자동으로 해결해 주는 코드를 거의 다 짜면서 점차 느끼게 된 점이다. 처음에는 단지 자동으로 프로그램이 풀어주면 좋지 않아?라고 생각했지만, 정작 만들어보니 사용자와 동떨어진 느낌을 많이 받은 듯한 느낌이 들었기 때문이다. 다음에는 좀 더 사용자와 가까운 관계를 가지는 그런 코딩을 할 것이다.

다음은 오류에 대한 내용이다. 처음 시작 화면에서 2를 입력하여 disk의 개수를 받는 화면이 출력된다. 하지만 처음 시작 화면에서 숫자, 그러니까 정수형 문자가 입력이 된다면 오류가 없이 disk의 개수를 받는 화면이 출력이 된다. 문제는 정수형 문자 말고 다른 문자들이 들어왔을 때이다. 예를 들어 e, *와 같이 다른 형의 문자들이 들어오면 많은 횟수의 출력들이 다시 출력되면서 시스템이 다운이 된다. 필자의 예상으로는 입력받은 문자들의 형의 아스키코드 값만큼의 횟수가 출력이 되고, 시스템이 다운이 되는 듯싶었다. 이 문제를 고치기 위해서 intro 함수를 다시 봤지만, 도저히 무엇이 문제인지 몰라서 고치지 못했다. 그리고 또 비슷한 오류가 하나 더 있다. 처음 시작 화면에서 2를 입력받고, disk 개수를 입력받는 화면에서 전과 같이 정수형이 아닌 다른 문자형의 입력을 받을 때 발생한다. 그러면 실행 결과의 일부분만 출력이 되고, 시스템이 종료가 되는 오류가 있다.

이렇게 하노이탑을 설계하며 배운 점도 있었지만, 아쉬운 점이 더 많은 코딩이었던 거 같았다. 다음부터는 코딩 전에 좀 더 고민하고, 어떤 식으로 코딩을 하면 좋을지 확실히 하고 코딩을 하는 것이 좋겠다고 느끼는 계기가 되었던 것 같다.