Tower of Hanoi

2015202047 국제통상학부 이민호

Contents

1. Planning

해당 과제는 첫 번째 막대기의 모든 원판을 마지막 막대기로 모두 옮기는 놀이인 하노이 타워를 구현하는 것이다. 이 때, 한 번에 하나의 원판 만을 옮길 수 있으며, 중간의 원판은 옮길 수 없고, 크기가 작은 원판 위에 그보다 큰 원판을 얹을 수 없다. 이와 같은 세 가지 조건을 만족해야 하며, 처음부터 즐기고 싶다면 즐길 수 있어야 하는 것이 또 하나의 요점이다.

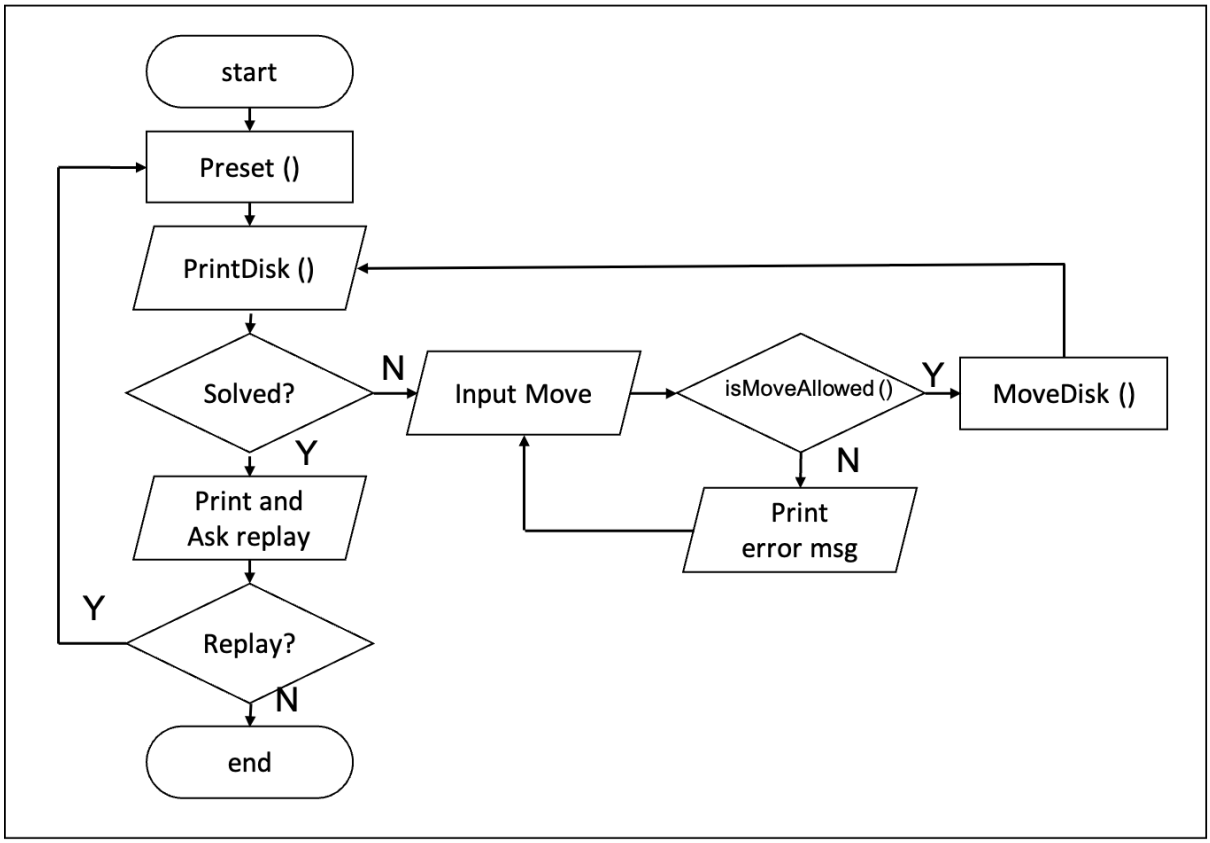

그림 1

하노이 타워 구현을 위하여 위 그림과 같이 Flow Chart를 구성하였다. 과제에서 제시한 세 함수 PrintDisk( ), isMoveAllowed( ), MoveDisk( ) 외, 디스크의 초기 설정과 관련된 void형 함수 Preset( ) 선언이 필요함을 알 수 있었다. PrintDisk( )는 디스크를 출력하는 함수이고, MoveDisk( )는 디스크를 내부적으로 위치를 변경하는 함수로써, 반환 값이 불필요하므로 void형으로 선언할 것이라 계획하였다. 또한, isMoveAllowed( )는 bool형으로 선언하여 사용자가 시도하려는 디스크의 움직임의 유효 여부를 true, false값으로 판별하는 것이 적합하다고 판단했다. 이외 기능은 별도 함수를 만들어도 범용성 확보에 별다른 도움이 되지 않을 것이라 생각하여 main( )에서 직접 구현을 하기로 계획하였다.

2. Function - Preset () Function – additional

```
void Preset(int size, vector<vector<int>>& row, vector<int>& tower)
{
    for (int i = 0; i < 3; i++)
    {
        row.push_back(tower);
    }//reset row

    for (int i = size; i > 0; i--)
    {
        row[0].push_back(i);
    }//set Disk

}
```

그림 2

기존 과제에서 요구한 함수 외에 추가한 함수이다. 디스크의 개수(size)와 main함수에서 선언된 vector<vector<int>>, vector<int> 변수를 주소 값으로 불러와 row 벡터를 구성하고, 첫 번째 기둥에 큰 디스크부터 차례대로(3,2,1) 넣음으로써, 함수 호출 시 간단하게 초기 설정이 가능하도록 구현하였다. 2차원 벡터를 초기화 하기 위해서 1차원 벡터의 변수로 초기화해야 하기 때문에 vector<vector<int>>, vector<int>형 인자를 동시에 받는다. 반복문과 push_back을 통해 각 타워를 구성하게 되고, 이 때 만들어진 2차원 벡터가 하노이 타워 작동의 기준이 된다.

2. Function - PrintTowers () Function - compulsory

과제에서 요구한 함수 중 첫 번째 함수이다. vector<vector<int> 형의 2차원 벡터의 변수를 주소값으로 받아, 모든 타워를 출력하도록 구현하였다. 각 타워 앞에는 i+1번째 기둥이라는 것을 명시하기 위해 [i+1] 형태의 텍스트를 추가하였고, 이에 뒤따라 각 타워에 저장된 디스크를 출력하도록 for문을 사용하였다. i와 j를 unsigned int로 선언한 이유는 row[i].size의 반환 값이 unsigned long인 탓에 경고 문구가 출력되었기 때문이다. 비주얼 스튜디오에서는 빌드가 가능하였으나, 과제를 수행하며 사용한 MacOS의 Xcode 컴파일러는 이를 허용하지 않아, int 형 변수에서 unsigned int형 변수로 수정하였다.

```
void PrintTowers(vector<vector<int>>& row)
{
    for (unsigned int i = 0; i < 3; i++)//rod == 3
    {
        cout << "[" << i + 1 << "]";
        for (unsigned int j = 0; j < row[i].size(); j++)
            cout << row[i][j] << " ";
        cout << "\n";
    }//Print tower
    cout << "\n";
}
```

그림 3

2. Function - isMoveAllowed () Function - compulsory

```
bool IsMoveAllowed(int fromWhere, int toWhere, int size, vector<vector<int>>& row)//
{

    if (fromWhere <= 0 || toWhere <= 0 || fromWhere==toWhere ||fromWhere>3 || toWhere>3)//input must be bigger
        return false;
    else if (row[fromWhere - 1].size() == 0)//disk must be moveable
        return false;
    else if (row[toWhere - 1].size() > 0)//when a disk is already set
    {
        if (row[toWhere - 1][row[toWhere - 1].size() - 1] < row[fromWhere - 1][row[fromWhere - 1].size() - 1])//
            return false;
    }

    return true;
}
```

그림 4

사용자가 디스크를 옮기려 할 때, 그 조건이 충족되는 지를 확인할 수 있도록 True, False값을 반환하는 bool형 함수를 선언하였다. 사용자의 입력의 조건은 다음과 같다.

1. 사용자가 움직이려는 타워의 번호가 0이하이거나, 3 초과이면 안 된다.
2. 사용자가 옮기려는 타워에 디스크가 있어야 한다.
3. 옮기려는 디스크가 이미 타워에 존재하는 디스크보다 크면 안 된다.

본래 하노이 탑의 조건에는 '한 번에 한 디스크만 옮길 수 있다'는 제약 조건이 있지만, 한 번에 하나만 옮길 수 있도록 입력을 받는다면 고려해야 할 조건이 아니라고 판단하여 별도로 조건을 만들지 않았다. 세 가지 조건에 따라 위 그림과 같이 조건문을 작성하였다. 첫 번째 조건문은 앞에서 ||연산자를 사용하여 조건 부합 여부를 판단할 수 있도록 하였으며, 이에 부합하지 않을 경우 false값을 return 한다. 두 번째 조건문은 디스크를 옮기려는 타워에 디스크가 존재하는지를 판단하는 조건문으로써, row[fromWhere-1].size를 통해 벡터의 크기를 확인하고, 디스크가 존재하지 않을 시, 즉 벡터의 크기가 0일 경우 false값을 반환한다. 마지막 조건문은 디스크가 옮겨질 타워에 이미 특정 디스크(row[toWhere-1][row[toWhere-1].size-1])가 있을 경우, 옮길 디스크(row[fromoWhere-1][row[fromWhere-1].size-1])의 크기를 비교한다. 만일, 옮길 타워에 이미 존재하는 디스크가 옮겨질 디스크보다 크기가 작다면 false를 반환하게 된다. 이로써 isMoveAllowed의 기능이 구현되었다.

2. Function - MoveDisk ()Function – compulsory

```
void MoveDisk(int fromWhere, int toWhere, vector<vector<int>> &row)
{
    int disk;
    disk = row[fromWhere-1][row[fromWhere-1].size() - 1];//save disk
    row[fromWhere-1].pop_back();
    row[toWhere-1].push_back(disk);
}
```

그림 5

MoveDisk( )는 옮기게 된 타워의 벡터 크기를 1 줄이고, 옮겨질 타워의 벡터 크기를 디스크의 크

기로 늘리는 함수이다. 사용자에게 입력 받을 fromWhere, toWhere를 함수 인자로 받고, 앞서 Preset( )에서 만들어진 벡터에 직접 접근하기 위해 주소값으로 인자를 받는다. push_back에 row[fromwhere-1][row[fromWhere.size( )-1]를 직접 입력해도 됐었지만, 지역변수 disk를 선언 및 저장함으로써 가독성을 높였다.

3. Main Function

```
vector<vector<int>> row = {};
vector<int> tower = {};
```

```
int size = 3;
int count = 1;
int fromWhere = 0;
int toWhere = 0;
char Playagain = 0;
char AdditionalFunction = 0;
```

그림 6

vector<vector<int>>형 2차원 벡터와 이를 초기화하기 위한 vector<int>형 변수를 각각 하나씩 선언하였다. 또한, 원 과제에서 요구한 디스크의 개수가 3개였으므로 3으로 초기화했다. 횟수를 세기 위하여 count, 사용자에게 이동할 위치를 받기 위한 fromWhere, toWhere 변수를 각각 int 형으로 선언하고, Y와N, 그리고 기타 글자를 입력 받기 위하여 char형 변수를 선언하였다

```
while (AdditionalFunction == 'n' || AdditionalFunction == 'N')
{
    count = 1;
    Preset(size, row, tower);
    PrintTowers(row);
```

그림 8

목차에서 볼 수 있다시피, 추가 기능을 구현하였기 때문에 어떠한 방식으로 하노이 탑을 실행할지에 대한 입력을 받았어야 했고, 이와 관련된 내용은 4.Additional Feature에서 서술할 예정이다. 과제에서 요구한, 추가 기능이 제외된 기능을 실행하기 위해 n또는 N을 입력 받으면, 해당 과제가 실행되는 while문을 실행하게 된다. 이와 동시에 사용자가 하노이 타워 재실행할 때, count를 초기화하기 위해 count를 1로 만들어주었다. Preset( )을 통해 [1]타워에 3,2,1순으로 디스크가 들어가도록 하고, 이렇게 저장된 벡터를 출력하기 위해 PrintTowers( ) 함수를 실행하면, 그림 9와 같은 결과물이 출력된다.

```
Do you want to play with additional function? (Y/N/A (A for auto)): n
[1]3 2 1
[2]
[3]


[1] From Which tower will you move a disk to which tower? (from=[1|2|3],
    to=[1|2|3]): 1 4
=>Move failed!

[1] From Which tower will you move a disk to which tower? (from=[1|2|3],
    to=[1|2|3]): 1 3
=>Move succeeded!
[1]3 2
[2]
[3]1


[2] From Which tower will you move a disk to which tower? (from=[1|2|3],
    to=[1|2|3]):
```

그림 9

타워가 출력된 후, 사용자에게 정상적인 입력이 이루어졌을 때에만 loop를 빠져나갈 수 있는 while문이 실행된다. 여기에서 앞서 선언한 IsMoveAllowed( )가 실행되고, 만일 false를 반환하면 위에서 보는 바와 같이 “=>move failed!” 텍스트가 출력되면서 다시 사용자에게 입력을 받는다. 정상적으로 이동할 수 있는 입력이 발생하면,

MoveDisk( )함수를 통해 사용자의 입력대로 디스크가 이동되고, PrintTowers( )함수를 통해 디스크가 옮겨진 모습으로 다시 출력되게 된다. 이와 같은 과정이 반복되다 마지막 타워의 디스크 개수가 3이 되면 해당 loop를 break하게 되며 다음 문으로 넘어간다. 코드의 내용은 그림 10과같다.

```
while (1)
{
    cout << "\n[" << count << "]" << " From Which tower will you move a disk to which tower? (from=[1|2|3], to=[1|2|3]): ";
    cin >> fromWhere >> toWhere;
    if (IsMoveAllowed(fromWhere, toWhere, size, row) == false)
    {
        cout << "=>Move failed! \n";
        continue;
    }
    MoveDisk(fromWhere, toWhere, row);

    cout << "=>Move succeeded! \n";
    PrintTowers(row);

    if (row[size - 1].size() == size)
        break;
    count++;
}
```

그림 10

```
    while (1)
    {
        cout << "Do you want to play again?(Y/N): ";
        cin >> Playagain;

        if (Playagain == 'y' || Playagain == 'Y')
            break;
        else if (Playagain == 'n' || Playagain == 'N')
            break;
        else
        {
            cout << "Incorrect input! Try again!\n";
            continue;
        }
    }

    if (Playagain == 'y' || Playagain == 'Y')
    {
        row.clear();
        cout << "\n";

        continue;
    }
    else if (Playagain == 'n' || Playagain == 'N')
        break;

}
```

그림 11

모든 과정이 수행된 후 while문을 빠져나오면, 다시 실행하는지 여부를 사용자에게 입력 받기 위해 또 다시 while문이 실행된다. 입력 받은 값은 Playagain에 저장되며, Y 또는 N 값을 받으면 해당 loop에서 빠져나온다. 그리고 이 입력값을 조건문으로 확인하는데, 만일 입력값이 Y라면 row 벡터를 .clear로 초기화하고, continue를 통해 위 문장 전체를 감싸고 있던 while문을 재실행하게 된다. 반면, N을 입력 받게 되면 break를 통해 loop를 빠져나오게 되고, 그대로 프로그램이 종료된다. 이로써 과제에서 요구한 모든 기능을 구현 완료하였으며, 추가 기능을 추가함으로써 더 완벽한 하노이 탑을 구현하고자 한다.

4. Additional Feature - Ask Before Play the Tower of Hanoi

```
while (1)
{
    cout << "Do you want to play with additional function? (Y/N/A (A for auto)): ";
    cin >> AdditionalFunction;

    if (AdditionalFunction == 'y' || AdditionalFunction == 'Y')
        break;
    else if (AdditionalFunction == 'n' || AdditionalFunction == 'N')
        break;
    else if (AdditionalFunction == 'a' || AdditionalFunction == 'A')
        break;
    else
    {
        cout << "Incorrect input! Try again!\n";
        continue;
    }
}/// ask additional function
```

그림 12

앞서 설명한 추가 기능을 판별하기 위해 만든 while문이다. 사용자에게 대소문자 상관 없이 Y, N, A 중 하나를 입력 받으면 해당 기능을 수행한다(Y는 추가 기능, N은 원 과제, A는 자동으로 하노이 타워를 완성하도록 한다).

```
while (1)
{
    cout << "How many disks do you want?:  ";
    cin >> size;

    if (size > 0)
        break;
    else
    {
        cout << "Incorrect input! Try again!\n";
        continue;
    }
}
```

그림 13

이와 비롯하여, 사용자가 디스크 개수를 0개보다 많도록 직접 설정할 수 있게 size를 입력 받는 while문을 추가하였다. 앞서 사용자에게 입력 받을 때와 마찬가지로 size의 입력이 정상적으로 이루어졌을 시에만 loop를 break한다. 이를 통해 사용자는 더 많은 경우의 수의 하노이 탑을 즐길 수 있게 되었다.

4. Additional Feature - additionalPreset ( ) Function

```
Do you want to play with additional function? (Y/N/A (A for auto)): y
How many disks do you want?:  4
#|
#||
#|||
#||||
#|||
#||
#|
```

그림 14

제시된 과제의 추가 기능 중 또 다른 하나는 숫자 대신 텍스트로 하노이 탑을 출력하는 것이다. 이 때, 디스크의 크기는 2n의 크기를 가지고 있으며, 각 타워의 바닥 면은 가장 큰 디스크와 같은 크기의 '#' 텍스트로 이루어져있다. 이를 구현하기 위해 새로운 함수 구현이 필요하였고, additionalPreset( )함수를 새로 선언하였다. 기존의 Preset 함수와 같이 디스크의 개수(size)를 int형 자료형으로 받고, 벡터에 직접 접근하기 위해 주소값을 받을 수 있도록 row와 tower를 인자를 받도록 하였다.

```
void additionalPreset(int size, vector<vector<int>>& row, vector<int>& tower)
{
    int k= 0;
    for (int i = 0; i < size*2*3; i++)
    {
        row.push_back(tower);
    }//reset row

    for (int i = size; i > 0; i--)
    {
        for(int j=0;j<(size-k)*2;j++)
        {
            row[j+k].push_back(i);
            //cout<<"preset success!"<<endl;
        }
        k++;
    }//set Disk

}
```

그림 15

```
Do you want to play with additional function?
How many disks do you want?:  2
#|
#||
#||
#|
```

그림16

우선, 가장 큰 디스크를 텍스트로 표현하면 그 크기는 사용자가 입력한 다스크의 개수(size)의 2배이고, 이와 같은 타워가 3개이므로 size*2*3개만큼 row를 tower로 push_back했다. 이후, 크기가 큰 디스크부터 차례로 디스크를 벡터에 입력하기 위해 또 다른 for문을 선언하였다. 이 때, 변수 k는 하나의 디스크가 벡터에 입력이 완료된 후 그 다음 디스크는 k보다 한 열 다음부터 push_back되도록 하며, k만큼 덜 push_back되도록 하는 역할을 한다. 예를 들어, 사용자가 원하는 디스크의 개수가 2개일 경우, '2'디스크가 row[0]부터 row[3]까지 push_back된 후, k는 1이 된다. 그 다음 '1'크

기의 디스크는 row[1]부터 size-k인 row[2]까지 push_back된다. 이렇게 좌상단의 이미지와 같이 디스크가 위치하게 된다.

4. Additional Feature - additionalPrintTowers ( ) Function

```
void additionalPrintTowers(vector<vector<int>>& row, int size)
{
    for (unsigned int i = 0; i < size*2*3; i++)//row.size()
    {
        cout << "#";
        for (unsigned int j = 0; j < row[i].size(); j++)
        {
            cout << "|";
        }
        cout << "\n";
        if((i+1)%(size*2)==0)
        {
            cout << "\n";
        }
    }//Print tower
    cout << "\n";

}
```

그림 17

앞서 additionalPreset( )에서 언급하였다시피, size*2*3의 열을 출력해야 한다. 타워의 바닥면을 "#"로 출력하고, 벡터가 유효한 만큼 "|"을 출력한다. 이와 같은 for문을 통해 row[i][j]에 저장된 모든 숫자가 "|" 텍스트로 출력이 이루어진다. 추가적으로, 타워를 구분하기 위해 size*2만큼 "#"이 출력된 후 줄 바꿈을 함으로써 세 타워의 밑 바닥을 구분하게 된다. 출력을 하게 되면 그림 14와 같이 출력된다.

4. Additional Feature - additionalIsMoveAllowed ( ) Function

```
bool additionalIsMoveAllowed(int fromWhere, int toWhere, int size, vector<vector<int>>& row)//
{
    if (fromWhere <= 0 || toWhere <= 0 || fromWhere==toWhere ||fromWhere>3 || toWhere>3)//input must be bigger than 0 and smaller than 4
        return false;
    else if (row[(fromWhere*size*2-size)].size() == 0)//disk must be moveable
        return false;
    else if (row[toWhere*size*2-size].size() > 0)//when the disk is already set
    {
        if (row[toWhere*size*2-size][row[toWhere*size*2-size - 1].size() - 1] < row[fromWhere*size*2-size][row[fromWhere*size*2-size - 1].size() - 1])
            smaller than the disk below
            return false;
    }
    return true;
}
```

그림 18

additionalIsMoveAllowed( ) 함수는 앞서 선언한 IsMoveAllowed( )와 같이 아래와 같은 조건을 충족해야한다. fromWhere과 toWhere은 사용자에게 입력를 받는다.

1. 사용자가 움직이려는 타워의 번호가 0이하이거나, 3 초과이면 안 된다.
2. 사용자가 옮기려는 타워에 디스크가 있어야 한다.
3. 옮기려는 디스크가 이미 타워에 존재하는 디스크보다 크면 안 된다.

그러나 IsMoveAllowed와 달리, 디스크에 할당되는 숫자만 다른 것이 아니라, 실제로 텍스트의 개수가 다르기 때문에 2번 조건과 3번 조건을 일부 수정이 필요했다. 각 타워의 중심이 되는 행의 벡터의 크기를 확인하면 되었기에 각 타워의 중심이 되는 fromWhere*size*2-size번째 row 벡터의 사이즈를 확인하기로 함으로써 타워의 디스크 유무를 확인하였다. 또한, 옮겨갈 타워에 디스크가 있을 경우(row[toWhere*size*2-size]>0), 디스크의 크기를 확인하기 위해 toWhere*size*2-size행에서 옮겨갈 toWhere*size*2-size-1번째 디스크와 fromWhere*size*2-size행으로 옮겨질 타워의 fromWhere*size*2-size-1번째 디스크의 대소를 비교하고, 만일 옮겨질 디스크의 크기가 옮길 타워의 디스크보다 크다면 false를 반환함으로써 세 번째 조건을 만족하게 된다.

4. Additional Feature - additionalMoveDisk ( ) Function

```
void additionalMoveDisk(int fromWhere, int toWhere, int size, vector<vector<int>> &row)
{
    int topdisk = 0 ;//size로 봐서 가장 긴거
    int disknum = 0;
    for (unsigned int i = (fromWhere-1)*size*2; i < fromWhere*size*2; i++)//row.size()
    {
        if(row[i].size()>topdisk)
        {
            topdisk=row[i].size();
        }

    }

    for (unsigned int i = (fromWhere-1)*size*2; i < fromWhere*size*2; i++)//row.size()
    {
        if(row[i].size()==topdisk)
        {
            disknum = row[i].back();
            row[i].pop_back();
        }
    }


    for (unsigned int i = size+size*2*(toWhere-1)-disknum; i < size+size*2*(toWhere-1)+disknum; i++)
    {
        row[i].push_back(disknum);
    }

    //cout<<"rowsize: "<<row.size()<<endl;

}
```

그림 19

옮겨갈 디스크를 파악하고, 디스크의 번호를 확인하기 위해 int형 변수 topdisk와 disknum을 선언하였다. 우선, 타워의 옮겨질 디스크 중 최상단의 위치한 디스크를 파악하기 위해 옮겨질 범위 내에서 for문을 선언하고, 이 중 가장 길이가 긴 벡터의 길이 크기를 topdisk에 저장한다. 가장 상단에 위치한 벡터의 길이는 topdisk와 크기가 동일하므로, row[i]의 사이즈가 topdisk와 같다면, 해당 벡터의 가장 뒤에 있는 원소를 disknum에 저장하고, pop_back으로 크기를 줄이는 for문을 선언한다. 이로써 타워에 끼워져 있던 디스크가 빠져나왔다.

옮겨갈 디스크는 디스크의 크기에 따라 push_back되는 시점이 다르다, 이를 식으로 표현하면 row[size+size*2(toWhere-1)-disknum]부터 row[size+size*2*(toWhere-1)+disknum-1]까지이다. 해당 범위만큼 for문을 작성하면 그림 19와 같이 된다.

4. Additional Feature - autoHanoi( ) & Path( ) Function

```
void Path(int size, int fromWhere, int toWhere, vector<vector<int>> &row)
{
    additionalMoveDisk(fromWhere, toWhere, size, row);
    additionalPrintTowers(row, size);
}
void autoHanoi(int disk, int size, int fromWhere, int toWhere, int temp, vector<vector<int>> &row, int &count)
{
    if (size == 1)
    {
        Path(disk, fromWhere, toWhere, row);
    }
    else
    {
        autoHanoi(disk,size-1, fromWhere, temp, toWhere, row, count);
        count++;
        Path(disk, fromWhere, toWhere, row);
        autoHanoi(disk, size-1, temp, toWhere, fromWhere,row, count);
        count++;

    }
}
```

그림 20

자동으로 하노이 탑을 완성시키는 함수이다. Path함수는 additionalMoveDisk와 additionalPrintTowers를 실행하는 함수이고, autoHanoi는 실질적으로 하노이 타워를 자동으로 만들어주는 함수이다. 이때, autoHanoi는 재귀함수로써, 디스크 크기를 받는 int형 disk, size변수와 어디에서 어디로 이동할 것인지를 결정하는 fromWhere, toWhere, 거쳐갈 타워인 temp를 인자로 받고, row와 count는 주소값으로 접근한다. 하노이 탑은 재귀적으로 과정 설명이 가능하다. 옮길 디스크가 1개이면, 바로 목표한 타워로 이동하면 끝이지만, 디스크가 size개라면 fromWhere기둥에 남아있는 디스크 중 size-1개를 temp로 옮긴다. 이후, fromWhere에 남아있던 디스크 하나를 toWhere로 옮기고 temp에 남아있는 디스크 중 size-1개의 원반을 다시 toWhere로 옮긴다. 이러한 재귀적 과정을 거쳐 하노이 탑 완성을 자동화시킬 수 있게 되었다. main( )에서의 구현은 아래 그림 21과 같다.

```
additionalPreset(size, row, tower);
additionalPrintTowers(row, size);


while (1)
{
    autoHanoi(size, size, 1, 3, 2, row, count);

    if (row[size*2*3 - size].size() == size)//clear
        break;
}
```

그림 21

마찬가지로 additionalPreset을 통해 초기 설정을 하였으며, 이를 additionalPrintTowers로 출력한다. autoHanoi가 작동한 후 마지막 타워의 중간 값이 size의 크기와 같아지면 break되며 결과를 출력하게 된다.

5. Conclusion

- Key Challenges

IsMoveAllowed( ) 함수의 조건 설정이 주 과제에서 가장 까다롭게 느껴졌다. 하노이 탑의 조건을 직관적으로 이해하는 것은 어렵지 않았으나, 이를 컴퓨터가 알아들을 수 있도록 조건을 설정하는 것이 쉽지는 않았다. 하지만, flow chart를 작성하고, 각 제약 조건을 연산자로 쉽게 작성할 수 있도록 변형하여 이해함으로써 쉽게 함수를 작성하는 것이 가능했다. 추가 기능으로 구현한 AdditionalIsMoveAllowed( )함수에서도 같은 원리를 사용함으로써 비교적 쉽게 함수를 수정해낼 수 있었다. AdditionalMoveDisk의 경우, 어떻게 가장 위에 있는 디스크를 찾고, 적절한 위치에 옮길 수 있을지가 고민이었다. 이 때, 작업 전 미리 직접 종이에 그린 그림과 유도한 수식을 보며 직관적으로 코드를 작성하였다.

- Limitations

본 코드는 try{ } catch{ }문을 학습하기 전에 작성된 코드로써, 기존에 알고 있던 C언어 지식을 활용하여 에러를 확인하였다. 코드 수정을 시도하였으나, while문을 남발한 탓에 코드를 쉽게 건드릴 수 없었다. 유지보수관리 측면에서 좋지 못한 코드를 작성하였다는 점이 아쉬움으로 남았으며, 이번 수업을 통해 C++언어에서만의 오류 확인 방법을 배운 만큼 앞으로 작성할 코드에선 각 언어의 장점을 살릴 수 있도록 노력해야겠다고 생각했다. 또한, 클래스 사용이 미숙하여 클래스를 선언하지 않았는데, 만일 클래스를 자유롭게 선언할 수 있었다면 더욱 짧은 코드로 과제를 완성시킬 수 있었을 것이다. 앞으로 수업에서 배우게 될 클래스에 대해 더욱 열심히 학습하여 객체지향 언어의 장점을 살릴 수 있도록 노력해야겠다는 생각을 갖게 되었다.

자동 하노이에서 Recursive를 통해 구현을 할 수 있을 것이라는 것을 직접 생각해내지 못하고 알고리즘 서적의 도움을 받았다는 것이 너무나 아쉬웠다. 하노이 탑의 디스크 이동 회수가 2^(n)-1 이라는 사실만 알고 있었지, 원리는 전혀 이해를 하지 못했다는 것은 본인의 학습 방법의 문제가 있음을 반증하는 것이었다. 많은 과제 중 하나라는 사실을 넘어, 전반적인 학습 방법에 대한 수정이 필요하다는 것을 배운 좋은 계기였다.

마지막으로, 하노이 탑을 세로로 출력하지 않고 가로로 출력하려고 시도하였으나, 고려해야 할 요소들이 너무 많고, 시간 제약으로 인해 구현을 하지 못할 것 같다는 결론을 내리고 구현을 포기하였다. 과제 내용을 너무 우습게 안 탓이 컸다고 생각한다. 항상 겸손한 자세로 수업 태도에 임해야겠다는 각오를 다지게 되었다.