고급프로그래밍
- Tower of Hanoi

제출일: 2021.04.10.

소프트웨어학부 2020203081
심유미

목차

I. 서론

1. 하노이 탑이란

하노이 탑은 막대와 원판을 가지고 하는 일종의 퍼즐이다. 원판은 크기 순서대로 끝에 있는 한 막대에 나란히 꽂혀 있고, 한 번에 하나의 원판만 움직여 반대편에 있는 막대에 동일한 형태로 원판들을 옮기면 된다. 이때 지켜야 할 규칙은 크기가 작은 원판 위에 크기가 큰 원판이 올 수 없다는 것이다. 무조건 크기가 작은 원판이 위로 올라가야한다. 최대한 적은 횟수로 원판을 옮기는 방법을 찾아내는 것이 하노이 탑의 목표이다.

2. 프로그램의 구조

게임이 시작되면 우선 하노이 탑의 현재 상황을 보여준다. 하노이 탑이 어떤 상태인지 확인한 플레이어는 주어지는 질문에 대해, 어느 자리의 원판을 어느 자리로 옮길지 답변을 한다. 답변을 받은 프로그램은 플레이어가 원하는 대로 원판을 옮길 수 있는지 판단을 하고, 옮길 수 없다면 답변을 재입력하도록 요구한다. 원판을 옮길 수 있다면 프로그램은 답변에 따라 원판을 옮겨주고, 현재의 하노이 탑이 최종 목표와 동일한 형태인지를 판단한다. 동일한 형태가 아니라면 바뀐 하노이 탑을 출력하여 다시금 원판을 어떻게 옮길 것인지에 대한 답변을 받도록 한다. 최종 목표와 동일하다면 해당 게임은 마무리됨과 동시에 게임을 다시 시작할 것인지에 대해 질문을 한다.

이 과정에 따라 프로그램을 6개의 함수로 구성했다. 하나의 함수는 프로그램의 전체적인 흐름을 잡아주고, 다른 5개의 함수로 필요한 연산 등을 진행한다. 이중 벡터를 이용해 막대 역할을 하는 3개의 자리를 마련해두고, 원판 역할을 하는 3개의 숫자를 넣도록 했다. 이때 숫자의 값이 원판의 크기를 나타낸다. 기본 환경이 마련되면 함수를 이용해 하노이 탑을 출력하고, 플레이어로부터 답을 받고, 원판을 이동할 수 있는지 판단하고, 원판을 이동시키고, 최종 목표와 동일하게 됐는지를 판단한다.

II. 본론

1. 주요 함수

가. StartGame 함수

함수를 이동하면서 각종 값에 접근하기 좋아야 하기에 구조체와 구조체 변수를 만들어 변수의 주소 값을 함수의 인자로 전달하는 방식으로 프로그램을 작성했다. 구조체는 이중 벡터 1개, int형 변수 4개로 구성돼있다. 이중 벡터는 하노이 탑을 나타내고, int형 변수는 이동횟수, 하노이 탑의 크기(원판의 개수), 플레이어가 입력한 답변 2개를 나타낸다. 하노이 탑의 크기를 제외한 나머지 변수와 이중 벡터는 함수 간 이동을 통해 값이 지속적으로 변한다.

main 함수가 시작되면 바로 StartGame 함수로 넘어간다. StartGame 함수의 시작에서는 구조체 변수와 bool 변수를 생성하고 초기화한다. 반복문을 이용해 이중 벡터에 값을 채워 넣는데, 첫 번째 자리에 값을 크기 순서대로 집어넣고 나머지 빈자리는 오류가 발생하지 않도록 숫자 0으로 채운다.

```
while (B != true) {
```

```
        PrintTowers(&H);
        GetAnswer(&H);
        IsMoveAllowed(&H);
        B = CheckTowers(&H);
 } //H는 구조체 변수 이름
```

위의 while문은 StartGame을 이루고 있는 것이다. StartGame 함수에는 bool타입 변수 B가 존재하고, 이 변수는 false 값으로 초기화돼있다. while문을 구성하는 모든 함수는 인자로 구조체의 주소를 받는다.

PrintTowers 함수는 현재 하노이 탑의 상태를 출력한다. GetAnswer 함수에서는 플레이어에게서 원판을 어떻게 이동시킬지에 대한 입력 값을 받는다. IsMoveAllowed 함수는 플레이어에게서 받은 입력 값에 따라 원판 이동 가능한지를 판단한다. 이동이 가능하면 MoveDisk 함수를 불러 원판을 이동시키고, 그렇지 않으면 아무런 변화 없이 함수에서 빠져나온다. 이후 CheckTowers 함수를 통해 현재 하노이 탑이 최종 목표에 도달했는지 확인한다. 하노이 탑 이동을 전부 끝내면 게임을 다시 시작할 것인지에 대해 질문을 하고, 입력 값에 따라 StartGame 함수로 돌아가거나 프로그램을 종료한다.

나. PrintTowers 함수

```
 for (int x : P->V_towers[i])
        if (x != 0)
                cout << " " << x;
```

PrintTowers 함수에 있는 이중 for문의 일부이다. 명령어를 간결하게 적기 위해 range-for문을 사용했다. 값이 저장돼있는 이중 벡터를 자리 별로 출력을 하는데, if문을 활용하여 지정된 벡터의 값이 0이 아닌 경우에만 출력하여 실질적인 값만 나타내도록 했다.

다. GetAnswer 함수와 IsMoveAllowed 함수

앞서 GetAnswer 함수에서 플레이어에게 질문을 보여준다. 어느 자리에서 어느 자리로 원판을 옮길지 물어보는 것이다. 플레이어의 입력 값은 구조체에 들어있는 변수에 저장되어 다음 함수인 IsMoveAllowed에서 사용된다. 플레이어의 입력 값은 IsMoveAllowed 함수에 정의돼있는 max 변수와 함께 벡터의 값에 접근하기 위한 인덱스로 사용되므로 때에 따라 1을 더하고 빼는 등 값 조정이 되기도 한다.

이 함수에서는 원판이 이동 불가한 경우 4가지를 정의하고 있다. if문을 이용하여 이동이 가능한 경우에만 다음 함수로 이동하게 구현을 했고, 명령어를 되도록 짧게 만들기 위해 하나의 if문에 여러 조건을 다 같이 적어두었다.

```
 if (0 > P->sel1 || P->sel1 > 2 || 0 > P->sel2 || P->sel2 > 2)
 else if (P->sel1 == P->sel2 || P->V_towers[P->sel1][0] == 0)
 if (max2 != -1 && P->V_towers[P->sel1][max1] > P->V_towers[P->sel2][max2])
```

첫 번째 if문은 플레이어가 선택한 자리가 존재하지 않는 자리일 경우이다. 자리는 1~3번까지만 있기에 그 이외의 숫자를 입력한 경우 오류 문구를 출력하도록 했다. 두 번째 if문은 플레이어가 2개의 입력 값을 동일하게 작성했거나, 혹은 원판을 빼와야 할 자리에 원판이 존재하지 않을 경우를 나타낸다. 두 개의 조건이 동시에 만족되는 경우가 있기에 하나의 if문에 같이 적어 넣었다.

첫 번째와 두 번째 if문을 통과했다는 것은 일단 원판을 이동시킬 수는 있는 상태임을 의미한다. 여기서 끝나는 게 아니라, 하노이 탑 게임 규칙에 따라 작은 크기의 원판이 큰 크기의 원판 위로 올라가고 있는지를 확인해야 한다. for문을 이용해 플레이어가 선택한 각 자리에서 가장 위에 자리한 원판, 즉 해당 자리에서 크기가 가장 작은 원판이 벡터의 몇 번째에 존재하는지 확인한다. 여기서 쓰이는 변수 max1과 max2는 원판이 존재하는 벡터의 인덱스 값을 나타내는 데에 쓰인다. 변수를 이용해 벡터에 존재하는 값을 비교했을 때 큰 원판이 작은 원판 위로 올라올 경우 이동이 불가한 것으로 처리한다.

원판 이동이 불가하여 오류 문구를 출력된 경우, 이동횟수와 이중 벡터의 값은 그대로이다. 모든 if문을 통과하여 원판의 이동이 가능하다 판단되면 원판을 이동하기 위해 인덱스 값과 구조체 주소를 다음 함수인 MoveDisk의 인자로 넘긴다.

라. MoveDisk 함수와 CheckTowers 함수

```
P->V_towers[P->sel2][max2] = P->V_towers[P->sel1][max1];
P->V_towers[P->sel1][max1] = 0;
P->moving++;
```

MoveDisk 함수는 위와 같이 구현돼있다. 첫 번째 문장은 플레이어가 두 번째로 선택한 자리에 원판을 옮기는 것을 뜻한다. 이전 IsMoveAllowed 함수에서 인자를 넘길 때 max2의 값은 1을 더 증가시켜 기존 값들이 존재하는 칸 바로 다음 칸에 값이 저장될 수 있도록 했다. 한편 플레이어가 첫 번째로 선택한 자리는 원판이 빠져 나갔으므로 기존에 저장돼있던 값은 0으로 바꿔 저장한다. 이동을 마무리 하면 이동횟수를 1 증가시키고, 이동이 완료됐음을 알리는 문구를 출력한다.

MoveDisk 함수에서 빠져나오면 StartGame 함수로 돌아가게 되고, StartGame 함수의 구성에 따라 CheckTowers 함수로 넘어간다. CheckTowers 함수는 하노이 탑의 최종 목표인 가장 마지막 세 번째 자리에 모든 원판이 크기순대로 쌓여져 있는 지를 판단한다.

```
for (int i = 0;i < P->size;i++)
      if (P->V_towers[2][i] != P->size - i)
            return false;
```

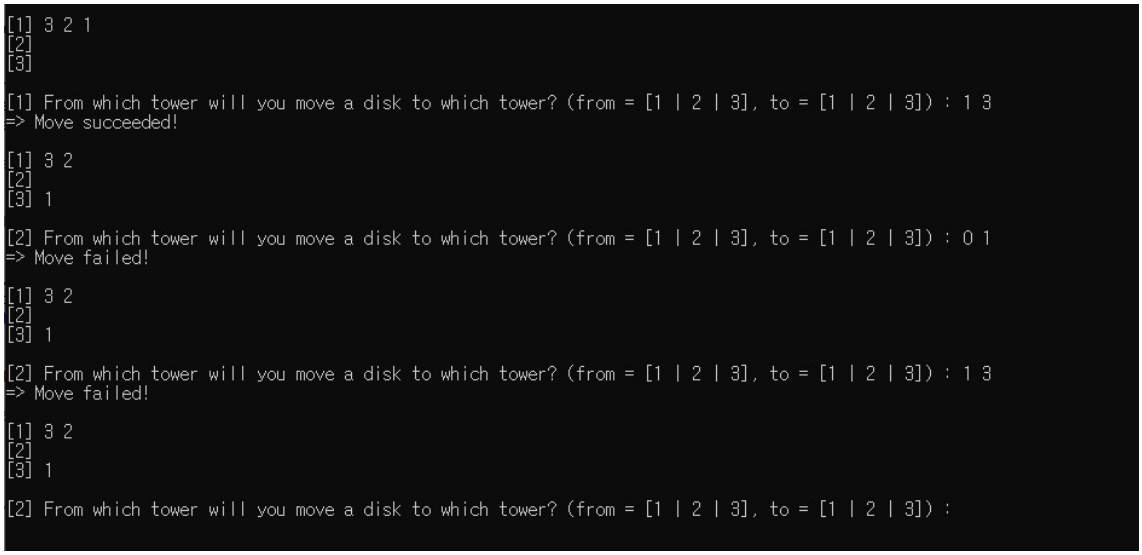
위와 같이 for문을 이용해 판단하는데, 현재의 하노이 탑이 최종적인 형태와 모양이 다를 경우 false 값을 반환한다. 잠시 StartGame 함수를 확인하자면 해당 반환 값은 bool 변수인 B에 저장이 된다.

B = CheckTowers(&H); //StartGame 함수

변수 B가 false 값을 가지고 있는 이상 StartGame 함수의 while문은 계속해서 반복된다. 즉 최종 목표를 달성하지 못하면 플레이어는 하노이 탑의 상태를 확인하고, 원판을 이동하는 작업을 계속해야 하는 것이다.

최종 목표에 도달하여 for문을 통과하면 하노이 탑의 모든 원판을 이동시켰다는 축하 문구를 출력하면서 게임을 깨는 데까지 누적된 이동횟수를 함께 안내한다. 이어서 게임을 다시 한 번 진행할 것인지에 대한 질문을 출력하는데, 플레이어가 'Y'나 'N'이 아닌 다른 값을 입력하면 프로그램은 올바른 답으로 재입력하길 요구한다. 플레이어가 'Y'를 입력하면 StartGame 함수의 가장 처음으로 돌아가 모든 설정을 초기화 한 채 게임을 재진행하고, 'N'을 입력하면 true(=변수 B) 값을 반환하면서 프로그램을 종료한다.

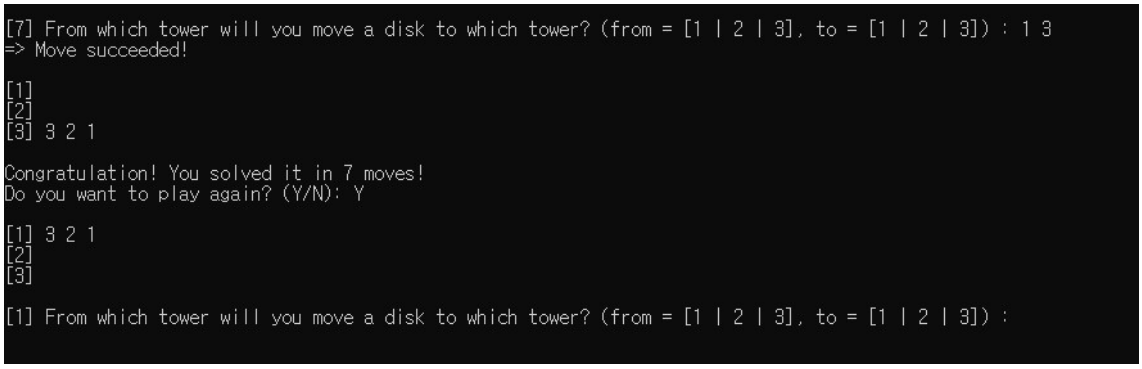
2. 실행 예시 (하노이 탑 기본형)



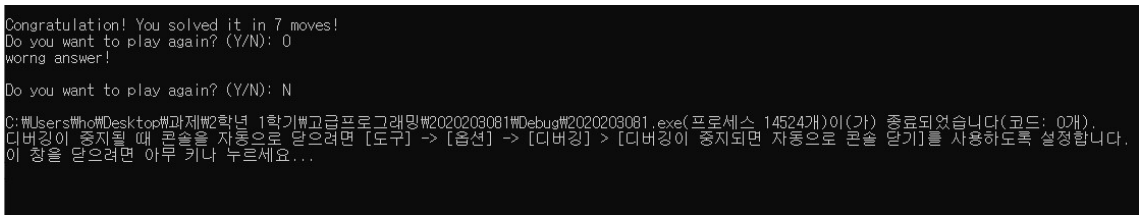
첫 번째 입력에서 1번 자리의 원판을 3번 자리로 이동 한 것에 대해, 어긋난 규칙이 없으므로 값이 올바르게 이동한 것을 볼 수 있다.

두 번째 입력에서 0번 자리의 원판을 1번 자리로 이동 한 것에 대해, 주어진 자리에 원판이 존재하지 않으므로 에러 문구를 보이고 값을 재입력하도록 하는 모습을 볼 수 있다.

세 번째 입력에서 1번 자리의 원판을 3번 자리로 이동 한 것에 대해, 하노이 탑의 규칙 상 크기가 큰 원판(2)이 작은 원판(1) 위로 올라올 수는 없으므로 이동이 되지 않았음을 볼 수 있다.



만들고자 했던 최종 목표를 달성하면 축하 문구가 나오고, 게임을 다시 시작할 것인지 물어본다. 여기서 재시작을 의미하는 'Y'를 입력하면 모든 값은 초기화되고 게임이 처음부터 다시 시작된다.



만약 이때 올바르지 않은 값(0)을 입력하면 잘못된 답변이라는 내용과 함께 값을 재입력하길 기다리는 모습을 볼 수 있다. 게임 종료를 의미하는 'N'을 입력하면 프로그램이 종료된다.

3. 추가한 기능 (하노이 탑 수정형)

하노이 탑 기본형은 원판의 개수가 3개로 정해져 있는데 수정형을 그 개수를 마음대로 지정할 수 있다. 원판의 개수를 변수로 저장하여 연산에 이용하므로, 기본형과 변수 초기화 방법이 다르다는 차이점만 존재한다. 단 막대 역할을 하는 3개의 자리는 수정이 불가하다.

III. 결론

1. 어려웠던 점

프로그램을 처음 구현할 때는 이중 벡터에 실질적인 값만 넣고 그 외의 자리는 빈자리로 남겨두었었다. 그 결과 새로운 값을 추가하고 기존에 있던 값을 없애는 것은 명령어를 이용해 쉽게 할 수 있었으나, 연산 과정에서 벡터의 빈자리 탓에 오류가 종종 발생했다. 설령 당장의 문제를 해결하더라도 플레이어의 입력에 따라 다른 곳에서 동일한 오류가 발생하기에 해당 오류를 완벽하게 해결할 수 없겠다 판단이 됐다.

이 문제를 해결하고자 벡터의 빈자리를 0으로 채우는 방식으로 바꿨다. 덕분에 벡터의 빈자리로 인해 생기는 오류는 해결할 수 있었지만 원판을 이동할 수 있는지 판단하고, 이동시키는 과정을 어떻게 구현해야 할지가 어려워졌다.

또한 두 개 이상의 함수에서 필요한 값들은 인자로 넘기면서 작업을 하다 보니 함수의 인

자가 4-5개씩 늘어나는 문제도 발생했었다. 이것에 대해 구조체를 이용하는 방식으로 문제를 해결하게 됐다.

2. 부족한 점

완성한 명령어를 더 단순하게 만드는 방법이 분명 있을 것이라 생각된다. 오류를 해결하기 위해 이중 벡터에 0이라는 불필요한 값들을 채워 넣었으나, 빈자리를 채우지 않고도 프로그램을 원활하게 돌아가게 만드는 방법은 존재하지 않았을까 싶다. 방법을 바꾸게 되면서 명령어로 간단하게 해결할 일도 긴 명령어로 연산을 진행해야 했으니, 원판을 간단하게 이동시킬 수는 없었을까 아쉬움이 든다.

원판의 이동이 가능한지 판단하는 부분에서도 내가 놓친 이동 불가능한 상황이 있으리라 생각된다. 현재 완성한 프로그램도 실행하는 중 숫자가 아닌 다른 값(ex, 알파벳)을 입력하면 무한 루프에 빠지는 문제점이 발견되었다. 플레이어의 입력 값이 숫자인 경우에만 원판 이동이 이루어지도록 수정해보고 싶었으나 해결법을 찾지 못했다.

또한 원인을 알아내진 못했으나 간혹 프로그램 종료가 제대로 되지 않는 경우가 있다. 기본형으로 게임을 진행하고, 이어서 수정형으로 게임을 하고 나면 프로그램이 바로 종료되지 않는다.

이와 관련해 무한 루프에 빠지는 등 무언가 문제가 생겼을 때를 대비해 error 함수를 활용해보고자 했으나 프로그램에 적용이 되지 않았다. 함수를 이용하는 방법을 몰라 생긴 문제라 생각된다. error 함수와 같이 수업에서 배운 것들을 마음껏 이용해보지 못한 것 같아 아쉬움이 든다.