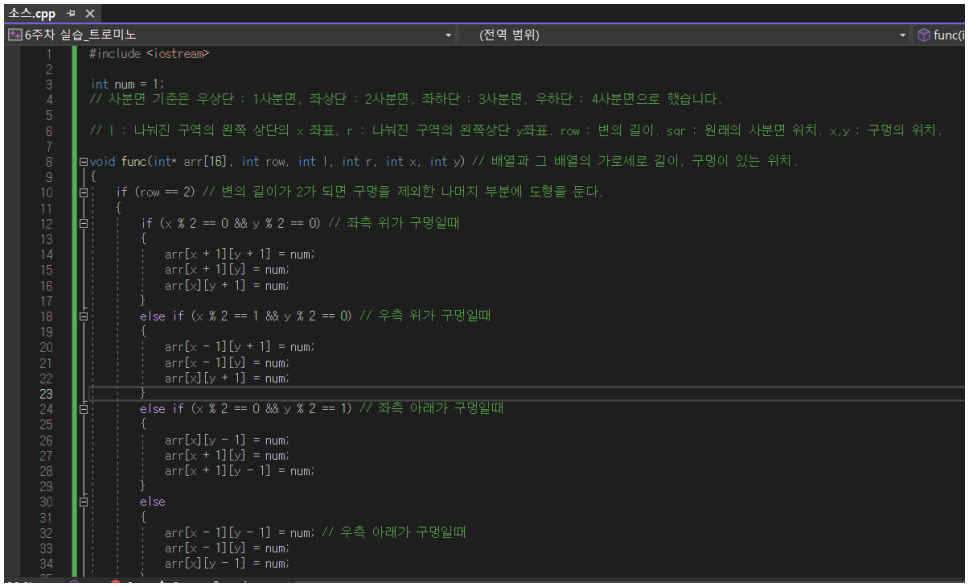
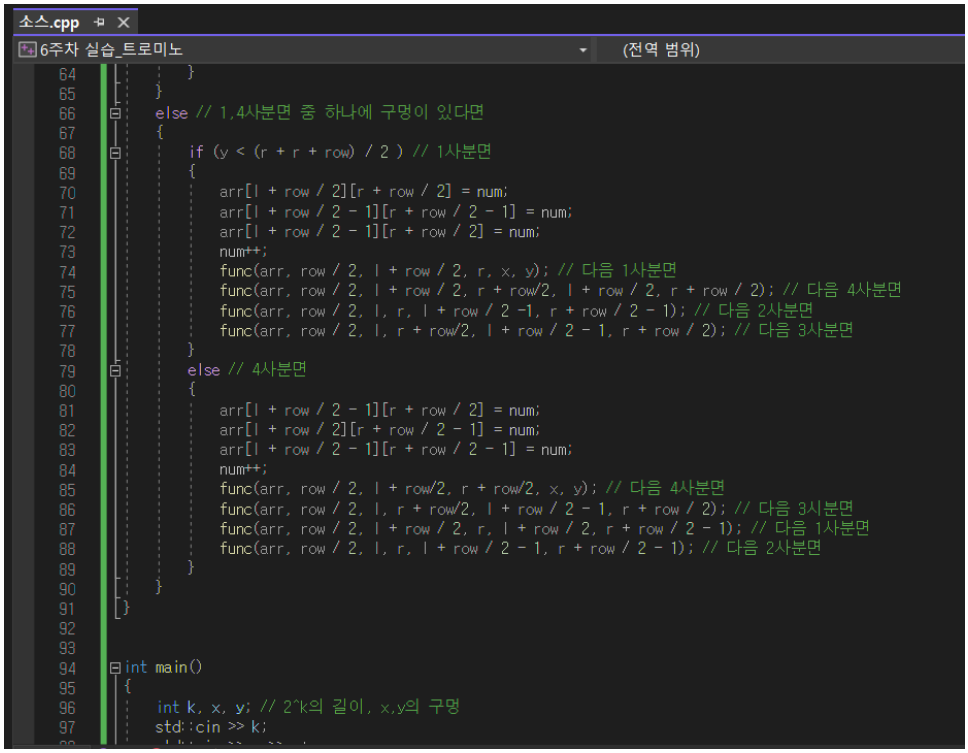
20101576 정재찬

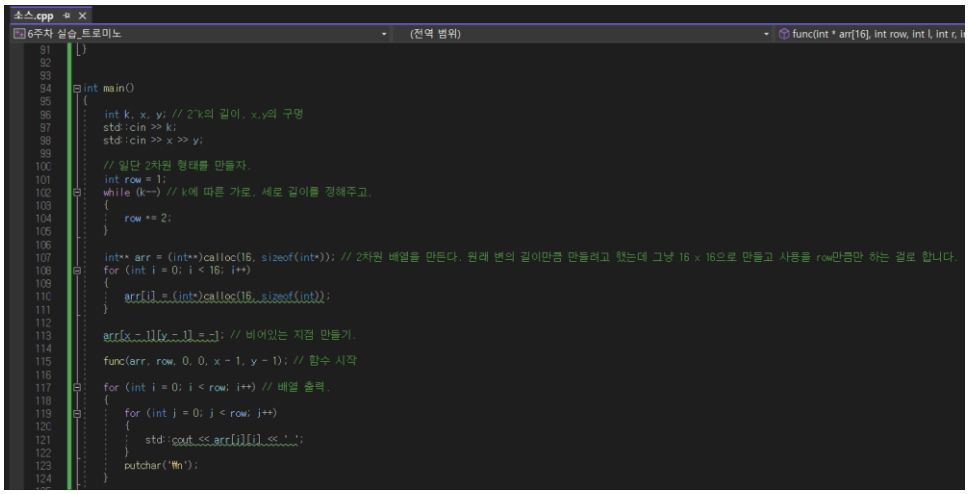
소스코드 및 주석(코드 설명)



```cpp
#include <iostream>

int num = 1;
// 사분면 기준은 우상단 : 1사분면, 좌상단 : 2사분면, 좌하단 : 3사분면, 우하단 : 4사분면으로 했습니다.

// l : 나눠진 구역의 왼쪽 상단의 x 좌표, r : 나눠진 구역의 왼쪽상단 y좌표, row : 변의 길이, sqr : 원래의 사분면 위치, x,y : 구멍의 위치,

void func(int* arr[16], int row, int l, int r, int x, int y) // 배열과 그 배열의 가로세로 길이, 구멍이 있는 위치,
{
    if (row == 2) // 변의 길이가 2가 되면 구멍을 제외한 나머지 부분에 도형을 둔다.
    {
        if (x % 2 == 0 && y % 2 == 0) // 좌측 위가 구멍일때
        {
            arr[x + 1][y + 1] = num;
            arr[x + 1][y] = num;
            arr[x][y + 1] = num;
        }
        else if (x % 2 == 1 && y % 2 == 0) // 우측 위가 구멍일때
        {
            arr[x - 1][y + 1] = num;
            arr[x - 1][y] = num;
            arr[x][y + 1] = num;
        }
        else if (x % 2 == 0 && y % 2 == 1) // 좌측 아래가 구멍일때
        {
            arr[x][y - 1] = num;
            arr[x + 1][y] = num;
            arr[x + 1][y - 1] = num;
        }
        else
        {
            arr[x - 1][y - 1] = num; // 우측 아래가 구멍일때
            arr[x - 1][y] = num;
            arr[x][y - 1] = num;
        }
        num++;
        return;
    }
    // 변의 길이가 4, 8, 16일 때
    // 구멍이 나눠진 구역(사각형 중) 몇 사분면인지 확인하고, 그 사분면을 제외한 다른 세 사분면에 겹치도록 도형을 둔 후 그렇게 생긴 구멍 4개에 대해 각각 이 함수를 재귀 호출한다.
    if (x < (l + l + row)/2) // 구멍이 2,3사분면 중 하나일 때,
    {
        if (y < (r + r + row) / 2) // 구멍이 2사분면 이라면 나머지 사분면에 모두 겹치도록 블록을 둠.
        {
            arr[l + row / 2][r + row / 2] = num;
            arr[l + row / 2][r + row / 2 - 1] = num;
            arr[l + row / 2 - 1][r + row / 2] = num;
            num++;
            func(arr, row / 2, l, r, x, y); // 다음 2사분면에 있는 타겟 점에 대해 반복
            func(arr, row / 2, l + row/2, r + row/2, l + row / 2, r + row / 2); // 다음 4사분면에 있는 타겟 점에 대해 반복
            func(arr, row / 2, l + row/2, r, l + row / 2, r + row / 2 - 1); // 다음 1사분면 ...
            func(arr, row / 2, l, r + row / 2, l + row / 2 - 1, r + row / 2); // 다음 3사분면 ...
        }
        else // 3사분면이라면
        {
            arr[l + row / 2][r + row / 2] = num;
            arr[l + row / 2][r + row / 2 - 1] = num;
            arr[l + row / 2 - 1][r + row / 2 - 1] = num;
            num++;
            func(arr, row / 2, l, r + row / 2, x, y); // 다음 3사분면
            func(arr, row / 2, l + row/2, r + row / 2, l + row / 2, r + row / 2); // 다음 4사분면
            func(arr, row / 2, l + row/2, r, l + row / 2, r + row / 2 - 1); // 다음 1사분면
            func(arr, row / 2, l, r, l + row / 2 - 1, r + row / 2 - 1); // 다음 2사분면
        }
    }
    else // 1,4사분면 중 하나일 때,
    {
        if (y < (r + r + row) / 2) // 구멍이 4사분면 이라면 나머지 사분면에 모두 겹치도록 블록을 둠.
        {
            arr[l + row / 2][r + row / 2] = num;
            arr[l + row / 2][r + row / 2 - 1] = num;
            arr[l + row / 2 - 1][r + row / 2] = num;
            num++;
            func(arr, row / 2, l, r, x, y); // 다음 4사분면에 있는 타겟 점에 대해 반복
            func(arr, row / 2, l + row/2, r, l + row / 2, r + row / 2); // 다음 1사분면에 있는 타겟 점에 대해 반복
            func(arr, row / 2, l + row/2, r + row/2, l + row / 2, r + row / 2 - 1); // 다음 4사분면 ...
            func(arr, row / 2, l, r + row / 2, l + row / 2 - 1, r + row / 2); // 다음 2사분면 ...
        }
        else // 2사분면이라면
        {
            arr[l + row / 2][r + row / 2] = num;
            arr[l + row / 2][r + row / 2 - 1] = num;
            arr[l + row / 2 - 1][r + row / 2 - 1] = num;
            num++;
            func(arr, row / 2, l, r + row / 2, x, y); // 다음 2사분면
            func(arr, row / 2, l + row/2, r, l + row / 2, r + row / 2); // 다음 1사분면
            func(arr, row / 2, l + row/2, r + row/2, l + row / 2, r + row / 2 - 1); // 다음 4사분면
            func(arr, row / 2, l, r, l + row / 2 - 1, r + row / 2 - 1); // 다음 3사분면
        }
    }
}
```



```
소스.cpp
6주차 실습_트로미노 (전역 범위)
64          }
65      }
66      else // 1,4사분면 중 하나에 구멍이 있다면
67      {
68          if (y < (r + r + row) / 2 ) // 1사분면
69          {
70              arr[l + row / 2][r + row / 2] = num;
71              arr[l + row / 2 - 1][r + row / 2 - 1] = num;
72              arr[l + row / 2 - 1][r + row / 2] = num;
73              num++;
74              func(arr, row / 2, l + row / 2, r, x, y); // 다음 1사분면
75              func(arr, row / 2, l + row / 2, r + row/2, l + row / 2, r + row / 2); // 다음 4사분면
76              func(arr, row / 2, l, r, l + row / 2 -1, r + row / 2 - 1); // 다음 2사분면
77              func(arr, row / 2, l, r + row/2, l + row / 2 - 1, r + row / 2); // 다음 3사분면
78          }
79          else // 4사분면
80          {
81              arr[l + row / 2 - 1][r + row / 2] = num;
82              arr[l + row / 2][r + row / 2 - 1] = num;
83              arr[l + row / 2 - 1][r + row / 2 - 1] = num;
84              num++;
85              func(arr, row / 2, l + row/2, r + row/2, x, y); // 다음 4사분면
86              func(arr, row / 2, l, r + row/2, l + row / 2 - 1, r + row / 2); // 다음 3시분면
87              func(arr, row / 2, l + row / 2, r, l + row / 2, r + row / 2 - 1); // 다음 1사분면
88              func(arr, row / 2, l, r, l + row / 2 - 1, r + row / 2 - 1); // 다음 2사분면
89          }
90      }
91  }
92
93
94  int main()
95  {
96      int k, x, y; // 2^k의 길이, x,y의 구멍
97      std::cin >> k;
```



```
소스.cpp
6주차 실습_트로미노 (전역 범위) func(int * arr[16], int row, int l, int r, i
91  }
92
93
94  int main()
95  {
96      int k, x, y; // 2^k의 길이, x,y의 구멍
97      std::cin >> k;
98      std::cin >> x >> y;
99
100     // 일단 2차원 형태를 만들자.
101     int row = 1;
102     while (k--) // k에 따른 가로, 세로 길이를 정해주고.
103     {
104         row *= 2;
105     }
106
107     int** arr = (int**)calloc(16, sizeof(int*)); // 2차원 배열을 만든다. 원래 변의 길이만큼 만들려고 했는데 그냥 16 x 16으로 만들고 사용을 row만큼만 하는 걸로 합니다.
108     for (int i = 0; i < 16; i++)
109     {
110         arr[i] = (int*)calloc(16, sizeof(int));
111     }
112
113     arr[x - 1][y - 1] = -1; // 비어있는 지점 만들기.
114
115     func(arr, row, 0, 0, x - 1, y - 1); // 함수 시작
116
117     for (int i = 0; i < row; i++) // 배열 출력.
118     {
119         for (int j = 0; j < row; j++)
120         {
121             std::cout << arr[j][i] << ' ';
122         }
123         putchar('\n');
124     }
```

시간 복잡도 계산

2차원 배열을 만들고 초기화 하는 시간 : O(2^2k)

구멍을 찾아 -1을 할당하는 시간 : O(1)

하나의 문제는 4개의 부분 문제들로 쪼개질 수 있고 각각의 문제의 크기는 변의 길이 2^k를 기준으로 반절로 줄어든다. k -> k-1 -> k-2 ... : 총 k번의 func호출이 고정적으로 발생하고, 각각의 func함수는 다시 4번의 func함수를 호출하기 때문에 시간 복잡도 : k * O(4^k) = O(k * 2^2k)

부분문제가 작아지다가 변의 길이가 2^k, k=1인 2가 되었을 때 : 연산의 횟수 : 3 (블록 하나를 넣는 연산) == O(3) ~= O(1)의 상수 시간 복잡도를 갖는다.

그러므로 총 시간 복잡도는 O(k * 2^2k)이 됩니다.

풀이 정답 증명사진