1. Running Environment

Window11, Visual Studio 2019에서 Debug x86으로 실행

2. Code Snippets

과제의 순서는 다음과 같다.

1. heap operation 명령어를 키보드 입력으로 받아옴
2. rotated tree 출력
3. not-rotated tree(original tree) 출력
4. H-tree 출력

main문의 내용은 다음과 같다.

```
string token;                           // 공백을 기준으로 구분될 문자열 토큰
string input = "";                      // 입력받은 문자열을 저장할 변수
int number = 0;                         // 트리 원소개수
int depth = 0;                          // 트리의 깊이
int row_90 = 0, col_90 = 0;             // 90도 돌아간 트리를 저장할 배열의 행, 열
int row = 0, col = 0;                   // 트리를 저장할 배열의 행, 열
const int operation_num = 200;          // 최대 연산 횟수
int operation_cnt = 0;                  // 연산 횟수 카운트
```

각 변수의 역할은 우측에 주석과 같다.

```
// 키보드 입력을 받는 부분(token으로 받은 줄별 입력을 input에 한줄짜리 문자열로 누적)
while (true) {
    getline(cin, token);
    // 한 줄의 마지막 입력이 EOI일때
    if (token[token.size() - 3] == 'E' && token[token.size() - 2] == 'O' && token[token.size() - 1] == 'I') {
        if (input == "") {  // input의 첫 입력이라면 공백을 띄우지 않고 input에 바로 추가
            for (unsigned int i = 0; i < token.size() - 3; i++) {
                input += token[i];
            }
        }
        else {
            input += " ";   // 앞에 이미 다른 값이 존재하면 아니라면 공백을 띄우고 input에 추가
            for (unsigned int i = 0; i < token.size() - 3; i++) {
                input += token[i];
            }
        }
        break;
    }
    // 한 줄의 마지막 입력이 EOI가 아닐 때
    else {  // 첫 입력이라면 공백을 띄우지 않고 input에 바로 추가
        if (input == "") {
            input += token;
        }
        else { // 앞에 이미 다른 값이 존재하면 아니라면 공백을 띄우고 input에 추가
            input += ' ';
            input += token;
        }
    }
}
```

getline함수를 통해 줄단위로 token을 받아오며, 받아온 마지막 입력이 EOI면 while 문을 탈출해 더 이상 입력을 받지 않는다. 이때, 현재 입력받은 줄이 첫 번째 줄이면

띄어쓰기를 하지 않고 input에 추가, 그렇지 않다면 띄어쓰기를 하고 input에 추가한다. 마지막 입력이 EOI가 아니라면 위와 같은 경우를 고려하여 input에 받아온 줄(token)들을 추가한다.

```
vector<string> command;          // 입력받은 heap operation과 값을 저장할 벡터
command = split(input, ' ');     // 공백을 기준으로 저장
vector<string> v;                // heap 벡터 v
make_heap(v.begin(), v.end());   // v로 heap을 생성
```

위에서 input은 “INS 1 INS 2 INS 3 DEL EOI” 이런 식의 문자열인데, 여기서 heap 연산과 그 값을 담을 벡터 command를 생성하고, input을 공백을 기준으로 넣는다. 그리고 heap 연산을 진행할 벡터 v를 만들고, make_heap을 해준다. 이때, 공백을 기준으로 쪼개주는 함수 split은 아래와 같다.

```
// delimiter를 기반으로 문자열을 쪼개는 함수
vector<string> split(string input, char delimiter)
{
    vector<string> result;
    stringstream ss(input);
    string temp;

    while (getline(ss, temp, delimiter)) {
        result.push_back(temp);
    }
    return result;
}
```

```
for (unsigned int i = 0; i < command.size(); i++) {
    if (operation_cnt > 200) {
        cout << "Error! Number of operations exceeded 200" << endl;
        return 0;
    }
    else {
        // INS가 들어오면 바로 다음 값을 push_heap 수행
        if (command[i] == "INS") {
            if (insert_list_check(command[i + 1], insert_list, sizeof(insert_list)/sizeof(string))) {
                v.push_back(command[i + 1]);
                push_heap(v.begin(), v.end());
                i++;
                number++;
                operation_cnt++;
            }
            else {
                return 0;
            }
        }
        // DEL이 들어오면 pop_heap 수행
        else if (command[i] == "DEL") {
            pop_heap(v.begin(), v.end());
            v.pop_back();
            number--;
            operation_cnt++;
        }
        // 다른 이상한 입력이 들어올 경우
        else {
            cout << "Error! Heap Operation Command is not correct. Try again!" << endl;
            return 0;
        }
    }
}
```

heap 연산을 진행하는 for문이다. 연산 횟수가 200을 초과할 시 에러 메시지를 출력 및 프로그램을 종료한다. INS 명령어가 들어오면, 그 다음의 값을 v에 push_heap하고, DEL이 들어오면 v에 pop_heap을 수행한다. 기타 입력(잘못된 명령어)이 들어올 경우, 에러를 출력하고 프로그램을 종료한다. 이때, 과제ppt에서 입력 가능한 범위가 정해지는데 이는 아래와 같다.

```
// 삽입가능한 문자 리스트
string insert_list[63] = {
    "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "?",
    "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z",
    "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"
};
```

위 for문에서는 입력받은 값이 insert_list에 속하는지를 파악하는 bool 함수 insert_list_check를 사용하여 예외처리를 추가로 진행하였다.

```
// 입력받은 값이 정해진 입력값에 속하는지 확인
bool insert_list_check(string input, string* list, int size) {
    for (int i = 0; i < size; i++) {
        if (input == list[i]) {
            return true;
        }
    }
    cout << "Error! Value is not on the insert list!" << endl;
    return false;
}
```

```
depth = log2(number + 1);   // 따른 트리의 깊이
row_90 = number;
col_90 = depth+1;
// 트리의 행, 열을 바꿈(회전 없는 트리)
row = col_90;
col = row_90;
```

트리 관련 변수를 선언해준다.

```
// rotated form 2차원 배열 동적할당 및 0으로 초기화
string** rotated_tree_arr = new string* [row_90];
for (int i = 0; i < row_90; i++) {
    rotated_tree_arr[i] = new string[col_90];
}

for (int i = 0; i < row_90; i++) {
    for (int j = 0; j < col_90; j++) {
        rotated_tree_arr[i][j] = '0';
    }
}

// not-rotated form 2차원 배열 동적할당 및 0으로 초기화
string** original_tree = new string * [row];
for (int i = 0; i < row; i++) {
    original_tree[i] = new string[col];
}

for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
        original_tree[i][j] = '0';
    }
}
```

회전된 트리와 회전되지 않은 트리(원본 트리)를 입력받을 2차원 배열을 동적할당 및 0으로 초기화 해준다.

```
// 1. rotated form 출력
cout << "1. rotated form" << endl;
print_90(v, 0, 0, rotated_tree_arr);    // 회전 트리를 그림


// rotated form 반시계 방향으로 90도 회전
for (int i = 0; i < row_90; i++) {
    for (int j = 0; j < col_90; j++) {
        original_tree[j][row_90 - i - 1] = rotated_tree_arr[i][j];
    }
}

// 2. not-rotated form 출력
cout << "2. not-rotated form" << endl;
// complete binary tree 출력, 0의 경우 공백으로 출력처리
for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
        if (original_tree[i][j] == "0") {
            cout << " ";
        }
        else {
            cout << original_tree[i][j];
        }
    }
    cout << endl;
}
```

non-rotated form은 rotated form을 반시계 방향으로 90도 회전시켜서 만들어주고, 2개의 트리를 출력한다. rotated form은 print_90 함수를 사용하여 출력하는데, 그 함수는 아래와 같다.

```
// 회전 트리를 그리는 함수
void print_90(vector<string> k, int index, int depth, string**& arr) {
    if (index < k.size()) {
        // 왼쪽 자식 노드 출력
        print_90(k, 2 * index + 2, depth + 1, arr);
        // 자기 자신 출력
        cout << setw(depth * 2 + 1) << k[index] << endl;
        if (depth == 0) {
            arr[a][0] = k[index];
        }
        else {
            arr[a][depth] = k[index];
        }
        a++;
        // 오른쪽 자식 출력
        print_90(k, 2 * index + 1, depth + 1, arr);
    }
}
```

```
// H-Tree
string h_tree_str;

for (int i = 0; i < v.size(); i++) {
    h_tree_str += v[i];
}

int h_tree_center = 0;
int h_tree_depth = 0;

h_tree_center = h_center(h_tree_str.length());
h_tree_depth = h_depth(h_tree_str.length());

int h_tree_row = h_tree_center * 2 + 1;
int h_tree_col = h_tree_row; // h-tree를 담을 배열은 정방배열
```

다음으로, 힙정렬된 v벡터값을 1자로 나열하는 h_tree_str 문자열을 만든다. h-tree를 담을 2차원 배열은 정방배열이며 row, col은 'h_tree_center * 2 + 1'값과 같다. h_tree_center, h_tree_depth를 구하는 함수는 아래와 같다.

```
int h_center(int number) { return number <= 1 ? 0 : 2 * h_center(number / 4) + 1; }
int h_depth(int number) { return number <= 7 ? 1 : 2 * h_depth(number / 4); }
```

```
// H-Tree를 담을 2차원 동적배열 생성
string** H_tree = new string * [h_tree_row];
for (int i = 0; i < h_tree_row; i++) {
    H_tree[i] = new string[h_tree_col * 2 + 1];
}

for (int i = 0; i < h_tree_row; i++) {
    for (int j = 0; j < h_tree_col; j++) {
        H_tree[i][j] = '0';
    }
}

make_htree(h_tree_str, 1, h_tree_center, h_tree_center, h_tree_depth, N, S, E, W, H_tree);
```

h-tree를 담을 2차원 동적배열을 생성해주고, make_htree함수를 실행하여 배열 H_tree에 h-tree를 만들어 준다. make_htree 함수의 입력은 순차적으로 힙정렬된 문자열, 2번째 인덱스(여기서는 1), h-tree의 중심, h-tree의 중심, h-tree의 깊이, 방향(북, 남, 동, 서), h-tree를 담을 배열 순이다. 방향은 다음과 같이 정의해 주었다.

```
// 좌표 이동: 2차원 배열의 경우이므로,
// 아래쪽(x): row증가, 오른쪽(y): col증가, 함수 좌표와 다름! 신경쓸 것
const int V[4][2] = { {-1, 0}, {1, 0}, {0, 1}, {0, -1} };
const int U = 0, D = 1, L = 3, R = 2;
const int N = U, S = D, E = R, W = L;
```

이때, h-tree는 배열이므로, 행, 열의 움직임을 잘 생각해서 선언한다.
h-tree를 만드는 함수 make_htree는 아래와 같다.

```
// H-Tree 만드는 함수
void make_htree(string str, int index, int i, int j, int depth, int U, int D, int R, int L, string**& H_tree_arr) {
    if (index > str.length()) return;  // 예외처리
    H_tree_arr[i][j] = str[index - 1];  // 0 인덱스 값을 센터점 출력
    // 좌
    if (2 * index <= str.length()) {
        H_tree_arr[i + depth * V[L][0]][j + depth * V[L][1]] = str[2 * index - 1];  // 좌 출력
        make_htree(str, 4 * index, i + depth * (V[L][0] + V[U][0]), j + depth * (V[L][1] + V[U][1]), depth / 2, D, U, L, R, H_tree_arr);        // 좌상 출력
        make_htree(str, 4 * index + 1, i + depth * (V[L][0] + V[D][0]), j + depth * (V[L][1] + V[D][1]), depth / 2, U, D, R, L, H_tree_arr);    // 좌하 출력
    }
    // 우
    if (2 * index + 1 <= str.length()) {
        H_tree_arr[i + depth * V[R][0]][j + depth * V[R][1]] = str[2 * index];  // 우 출력
        make_htree(str, 4 * index + 2, i + depth * (V[R][0] + V[D][0]), j + depth * (V[R][1] + V[D][1]), depth / 2, U, D, R, L, H_tree_arr);    // 우하 출력
        make_htree(str, 4 * index + 3, i + depth * (V[R][0] + V[U][0]), j + depth * (V[R][1] + V[U][1]), depth / 2, D, U, L, R, H_tree_arr);    // 우상 출력
    }
}
```

주석에 달린 바와 같이, make_tree 함수내에서 make_htree 함수를 4번 호출하며, 중심이 되는 노드의 좌상, 좌하, 우하, 우상 노드를 위치를 맞춰서 출력해준다.

출력 결과

입력에 따른 프로그램 출력 결과는 다음과 같다.

```
INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8 INS 9 EOI
1. rotated form
    5
  6
    2
9
    3
  8
      4
    7
      1
2. not-rotated form
     9
   8   6
 7  3 2 5
1 4
3. H-tree form

471  5

 8 9 6

 3   2


C:\Users\daida\Desktop\HW3_2018741022\Debug\HW3_2018741022.exe(프로세스 33212개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

```
INS 1 INS 5 INS 7 INS 3 DEL INS ? INS B INS 7 INS 9 INS b INS q
DEL INS x INS X EOI
1. rotated form
    7
  9
    1
x
    X
      5
  b
      ?
    B
      3
2. not-rotated form
      x
   b    9
 B   X 1 7
3 ? 5
3. H-tree form

?B3  7

 b x 9

5X   1


C:\Users\daida\Desktop\HW3_2018741022\Debug\HW3_2018741022.exe(프로세스 18104개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

올바르지 못한 입력을 넣었을 때의 예외처리 결과이다.

```
INS 1 INS 2 INS / DEL EOI
Error! Value is not on the insert list!

C:\Users\daida\Desktop\HW3_2018741022\Debug\HW3_2018741022.exe(프로세스 34384개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
|
```

```
INS 2 INS 6 INK EOI
Error! Heap Operation Command is not correct. Try again!

C:\Users\daida\Desktop\HW3_2018741022\Debug\HW3_2018741022.exe(프로세스 3848개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
|
```

```
INS 1 INS 5 INS 7 INS 3 DEL INS ? INS B INS 7 INS 9 INS b INS q
DEL INS x INS X
INS a INS b INS c INS d INS e INS f INS A INS B INS C INS D INS F INS G
INS 1 INS 5 INS 7 INS 3 DEL INS ? INS B INS 7 INS 9 INS b INS q
DEL INS x INS X
INS a INS b INS c INS d INS e INS f INS A INS B INS C INS D INS F INS G
INS 1 INS 5 INS 7 INS 3 DEL INS ? INS B INS 7 INS 9 INS b INS q
DEL INS x INS X
INS a INS b INS c INS d INS e INS f INS A INS B INS C INS D INS F INS G
INS 1 INS 5 INS 7 INS 3 DEL INS ? INS B INS 7 INS 9 INS b INS q
DEL INS x INS X
INS a INS b INS c INS d INS e INS f INS A INS B INS C INS D INS F INS G
INS 1 INS 5 INS 7 INS 3 DEL INS ? INS B INS 7 INS 9 INS b INS q
DEL INS x INS X
INS a INS b INS c INS d INS e INS f INS A INS B INS C INS D INS F INS G
INS 1 INS 5 INS 7 INS 3 DEL INS ? INS B INS 7 INS 9 INS b INS q
DEL INS x INS X
INS a INS b INS c INS d INS e INS f INS A INS B INS C INS D INS F INS G
INS 1 INS 5 INS 7 INS 3 DEL INS ? INS B INS 7 INS 9 INS b INS q
DEL INS x INS X
INS a INS b INS c INS d INS e INS f INS A INS B INS C INS D INS F INS G
INS 1 INS 5 INS 7 INS 3 DEL INS ? INS B INS 7 INS 9 INS b INS q
DEL INS x INS X
INS a INS b INS c INS d INS e INS f INS A INS B INS C INS D INS F INS G
EOI
Error! Number of operations exceeded 200

C:\Users\daida\Desktop\HW3_2018741022\Debug\HW3_2018741022.exe(프로세스 13780개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
|
```

3. Discussion(고찰)

이번 과제에서는 Heap 정렬된 트리를 출력해 보았다. 과제를 하면서 느낀 장점으로는 Heap 트리(최대 힙)는 우선 순위에 따라서 위에서 아래로 정렬되기 때문에, 빠르게 자료를 검색할 수 있을 것 같았다. 실제로 Heap 트리의 높이는 완전 이진트리이므로 $\log_2 n$이므로 원소를 힙에 삽입 및 삭제 시 $\log_2 n$의 시간이 소요된다. 그리고 원소가 총 n개이기 때문에, 전체적으로 $O(nlog_2 n)$의 시간복잡도를 지닌다. 다른 정렬들에 비해 빠른 편인 것 같다. 특히, 최대, 최솟값을 찾을 때는 $O(1)$의 시간복잡도만 지니기 때문에, 여러 자료들이 있을 때 최대 최소검색에 있어서 속도적으로 굉장히 시간을 단축할 수 있다고 본다. H-tree의 경우, 재귀를 사용하여 fractal을 생성하므로써, 트리가 매우 커지더라도 빈 공간을 효율적으로 채울 수 있다는 점이 큰 장점인 것 같다. 하지만, visualize하였을 때 일반적인 트리보다는 바로 알아보기가 어렵다는 단점이 있다.