BST to Max Heap Conversion (C++ Implementation)

Idea / Approach

1. **Step 1: Reverse Inorder Traversal**
   - Reverse inorder (`Right → Root → Left`) traversal ek BST ka **descending order** elements deta hai.
   - Example: Agar BST me values `{1, 5, 8, 10, 12, 16, 20}` hain, to reverse inorder se `{20, 16, 12, 10, 8, 5, 1}` milega.
2. **Step 2: Preorder Traversal for Value Placement**
   - Preorder traversal (`Root → Left → Right`) me node-by-node jaa ke values overwrite karte hain descending array se.
   - Isse ensure hota hai ki **root sabse bada** hoga, left aur right subtree me bhi Max Heap property maintain hogi.
3. **Why it works?**
   - Reverse inorder se sorted descending list milti hai (Max Heap ka root sabse bada hota hai).
   - Preorder traversal ensure karta hai ki pehle root ko assign ho, fir uske children ko — exactly Max Heap ka structure.

Code

```
#include <iostream>
#include <vector>
using namespace std;

class Node {
public:
```

```
    int val;
    Node* left;
    Node* right;
    Node(int val) {
        this->val = val;
        left = NULL;
        right = NULL;
    }
};

// Step 1: Reverse inorder traversal (Right -> Root -> Left)
void inorder(Node* root, vector<int>& arr) {
    if (root == NULL) return;
    inorder(root->right, arr);
    arr.push_back(root->val);
    inorder(root->left, arr);
}

// Step 2: Preorder traversal to replace node values
void preorder(Node* root, vector<int>& arr, int* i) {
    if (root == NULL) return;
    root->val = arr[(*i)++];
    preorder(root->left, arr, i);
    preorder(root->right, arr, i);
}

// Utility: Find height of tree
int level(Node* root) {
    if (root == NULL) return 0;
    return 1 + max(level(root->left), level(root->right));
}

// Utility: Print nodes at a given level
void nth_level_display(Node* root, int lvl, int t_lvl) {
    if (root == NULL) return;
    if (lvl == t_lvl) {
```

```
        cout << root->val << " ";
        return;
    }
    nth_level_display(root->left, lvl + 1, t_lvl);
    nth_level_display(root->right, lvl + 1, t_lvl);
}

// Utility: Print tree level-wise
void level_wise_display(Node* root) {
    int n = level(root);
    for (int i = 0; i < n; i++) {
        for (int j = n - 1 - i; j > 0; j--) {
            cout << " ";
        }
        nth_level_display(root, 0, i);
        cout << endl;
    }
}

int main() {
    // Creating BST
    Node* a = new Node(10);
    Node* b = new Node(5);
    Node* c = new Node(16);
    Node* d = new Node(1);
    Node* e = new Node(8);
    Node* f = new Node(12);
    Node* g = new Node(20);

    a->left = b;
    a->right = c;
    b->left = d;
    b->right = e;
    c->left = f;
    c->right = g;
```

```
    cout << "Original BST (Level-wise):\n";
    level_wise_display(a);

    // Step 1: Get descending order elements
    vector<int> arr;
    inorder(a, arr);

    // Step 2: Assign values in preorder manner
    int i = 0;
    preorder(a, arr, &i);

    cout << "\nMax Heap (Level-wise):\n";
    level_wise_display(a);

    return 0;
}
```

Dry Run Example

BST:

```
   10
  /  \
 5    16
/ \   / \
1  8 12  20
```

Step 1: Reverse Inorder → `arr = {20, 16, 12, 10, 8, 5, 1}`

Step 2: Preorder Replacement:

1. Root (10) → replace with 20 → `i=1`
2. Left child (5) → replace with 16 → `i=2`
3. Left child (1) → replace with 12 → `i=3`
4. Right child (8) → replace with 10 → `i=4`

5. Right subtree root (16) → replace with 8 → `i=5`
6. Left child (12) → replace with 5 → `i=6`
7. Right child (20) → replace with 1 → `i=7`

Max Heap Result:

```
    20
   /  \
  16   8
 / \   / \
12 10 5   1
```

Output

```
Original BST (Level-wise):
   10
  5 16
 1 8 12 20

Max Heap (Level-wise):
   20
  16 8
 12 10 5 1
```

✅ Logic Check

- Max Heap property: **Every node ≥ its children** — maintained ✔
- Structure remains same as original tree ✔
- Time Complexity: **O(n)** (both traversals take O(n)) ✔
- Space Complexity: **O(n)** (extra array) ✔