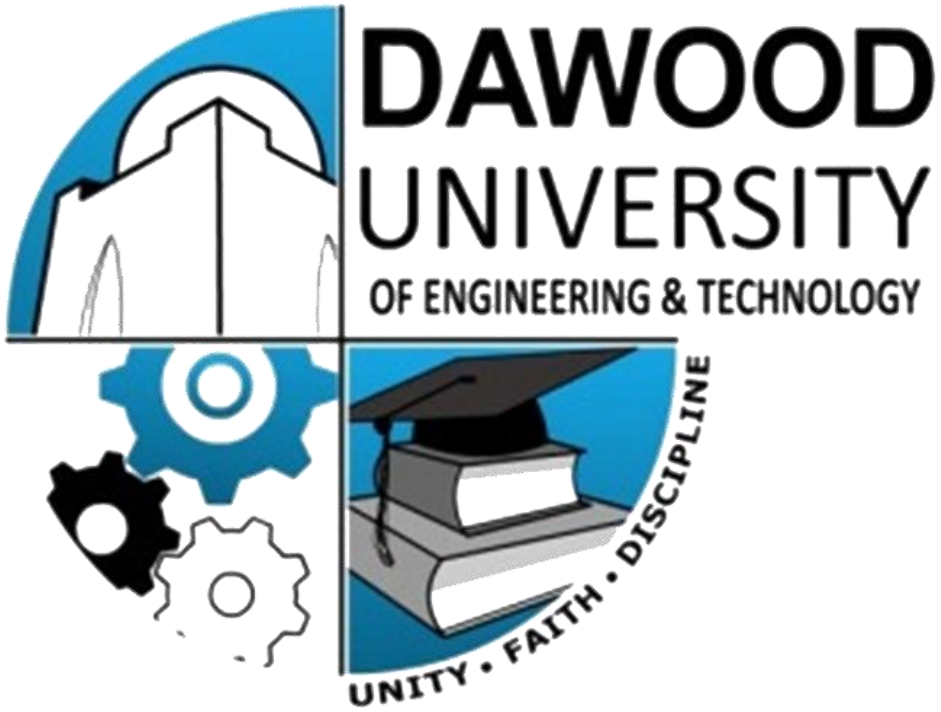
OEL#2



Name:Muhammad Ilyas khan

Roll No: 23-CS-115

Subject: DSA

Q1) A simple text editor needs to implement an Undo feature. Every time a user performs an operation(like typing or deleting text), it should be possible to undo the last action. Use a stack to achieve this functionality.

```cpp
#include <iostream>
#include <stack>
#include <string>

using namespace std;

class TextEditor {
private:
    string text;
    stack<string> history;

public:
    void addText(const string& newText) {
        history.push(text);
        text += newText;
        cout << "Added text: \"" << newText << "\"\n";
    }

    void deleteText(int count) {
        if (count > text.size()) {
            cout << "Cannot delete more characters than present.\n";
            return;
        }
        history.push(text);
        text.erase(text.size() - count);
        cout << "Deleted last " << count << " characters\n";
    }

    void undo() {
        if (!history.empty()) {
            text = history.top();
            history.pop();
            cout << "Undo performed. Current text: \"" << text << "\"\n";
        } else {
            cout << "No operations to undo.\n";
        }
    }

    void displayText() const {
        cout << "Current text: \"" << text << "\"\n";
    }
};

int main() {
    TextEditor editor;

    editor.addText("Hello");
    editor.displayText();

    editor.addText(" World");
    editor.displayText();

    editor.deleteText(5);
    editor.displayText();

    editor.undo();
    editor.displayText();

    editor.undo();
    editor.displayText();

    editor.undo();
    editor.displayText();

    return 0;
}
```

OUTPUT:

Q2) Check the CHILDREN SUM PROPERTY in the Binary tree, i.e. for every node data value must be equal to the sum of data values of the left and right child.

task2.cpp > ...

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

struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = right = nullptr;
    }
};

bool checkChildrenSumProperty(Node* root) {
    if (root == nullptr || (root->left == nullptr && root->right == nullptr)) {
        return true;
    }

    int leftData = (root->left) ? root->left->data : 0;
    int rightData = (root->right) ? root->right->data : 0;

    if (root->data == leftData + rightData &&
        checkChildrenSumProperty(root->left) &&
        checkChildrenSumProperty(root->right)) {
        return true;
    }

    return false;
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(8);
    root->right = new Node(2);
    root->left->left = new Node(3);
    root->left->right = new Node(5);
    root->right->left = new Node(2);

    if (checkChildrenSumProperty(root)) {
        cout << "The tree satisfies the Children Sum Property.\n";
    } else {
        cout << "The tree does not satisfy the Children Sum Property.\n";
```

```cpp
bool checkChildrenSumProperty(Node* root) {
    if (root == nullptr || (root->left == nullptr && root->right == nullptr)) {
        return true;
    }

    int leftData = (root->left) ? root->left->data : 0;
    int rightData = (root->right) ? root->right->data : 0;

    if (root->data == leftData + rightData &&
        checkChildrenSumProperty(root->left) &&
        checkChildrenSumProperty(root->right)) {
        return true;
    }

    return false;
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(8);
    root->right = new Node(2);
    root->left->left = new Node(3);
    root->left->right = new Node(5);
    root->right->left = new Node(2);

    if (checkChildrenSumProperty(root)) {
        cout << "The tree satisfies the Children Sum Property.\n";
    } else {
        cout << "The tree does not satisfy the Children Sum Property.\n";
    }

    return 0;
}
```

OUTPUT;

```
The tree satisfies the Children Sum Property.
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=$
muhammad.ilyas.khan@muhammad.ilyas.khan-Precision-3541:~/Desktop/files/u
```