NAME :- Manish Shashikant Jadhav
UID :- 2023301005.
BRANCH :- Comps -B. **BRANCH:** B.
EXPERIMENT 5: Implement ADT for storing a BST and performing operations on it.
SUBJECT :- DS (DATA STRUCTURES).

CODE :-

```c
#include <stdio.h>
#include <stdlib.h>

// Define the structure for a binary tree node
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

// Function to create a new node
struct TreeNode* createNode(int data) {
    struct TreeNode* newNode = (struct
TreeNode*)malloc(sizeof(struct TreeNode));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to insert a node into the binary tree
struct TreeNode* insertNode(struct TreeNode* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }

    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        root->right = insertNode(root->right, data);
    }

    return root;
}
```

```c
// Function to find the minimum value node in the binary tree
struct TreeNode* findMinValueNode(struct TreeNode* root) {
    if (root == NULL || root->left == NULL) {
        return root;
    }
    return findMinValueNode(root->left);
}

// Function to find the maximum value node in the binary tree
struct TreeNode* findMaxValueNode(struct TreeNode* root) {
    if (root == NULL || root->right == NULL) {
        return root;
    }
    return findMaxValueNode(root->right);
}

// Function to display the binary tree using inorder traversal
void displayTree(struct TreeNode* root) {
    if (root != NULL) {
        displayTree(root->left);
        printf("%d ", root->data);
        displayTree(root->right);
    }
}

// Function to delete a node with a specific value
struct TreeNode* deleteNode(struct TreeNode* root, int data) {
    if (root == NULL) {
        return root;
    }

    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        // Node with only one child or no child
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
```

```
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }

        // Node with two children: Get the in-order successor
(smallest in the right subtree)
        struct TreeNode* temp = findMinValueNode(root->right);

        // Copy the in-order successor's data to this node
        root->data = temp->data;

        // Delete the in-order successor
        root->right = deleteNode(root->right, temp->data);
    }

    return root;
}

// Function to find a node with a specific value
struct TreeNode* findNode(struct TreeNode* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }

    if (data < root->data) {
        return findNode(root->left, data);
    } else {
        return findNode(root->right, data);
    }
}

// Function to swap values of two nodes with specific values
void swapNodeValues(struct TreeNode* root, int value1, int
value2) {
    struct TreeNode* node1 = findNode(root, value1);
    struct TreeNode* node2 = findNode(root, value2);
```

```
    if (node1 == NULL || node2 == NULL) {
        printf("One or both values not found in the tree.\n");
        return;
    }

    int temp = node1->data;
    node1->data = node2->data;
    node2->data = temp;
}

int main() {
    struct TreeNode* root = NULL;

    root = insertNode(root, 28);
    root = insertNode(root, 9);
    root = insertNode(root, 13);
    root = insertNode(root, 2);
    root = insertNode(root, 11);
    root = insertNode(root, 7);
    root = insertNode(root, 18);
    root = insertNode(root, 10);

    printf("Binary Tree Inorder Traversal: ");
    displayTree(root);
    printf("\n");

    struct TreeNode* minNode = findMinValueNode(root);
    printf("Minimum Value Node: %d\n", minNode->data);

    struct TreeNode* maxNode = findMaxValueNode(root);
    printf("Maximum Value Node: %d\n", maxNode->data);

    int valueToFind = 18;
    struct TreeNode* nodeToFind = findNode(root, valueToFind);
    if (nodeToFind != NULL) {
        printf("Node with value %d found in the tree.\n",
valueToFind);
    } else {
```

```
        printf("Node with value %d not found in the tree.\n",
valueToFind);
    }

    int valueToDelete = 7;
    root = deleteNode(root, valueToDelete);
    printf("Binary Tree after deleting node with value %d: ",
valueToDelete);
    displayTree(root);
    printf("\n");

    int valueToSwap1 = 28;
    int valueToSwap2 = 13;
    swapNodeValues(root, valueToSwap1, valueToSwap2);
    printf("Binary Tree after swapping values %d and %d: ",
valueToSwap1, valueToSwap2);
    displayTree(root);
    printf("\n");

    // Clean up memory (optional)
    free(root);

    return 0;
}
```

Output:

```
PS D:\Manish\DS SPIT>  & 'c:\Users\manis\.vscode\extensions\ms-vscode.cpptools-1.17.5-win32-x64\debugAdapt
in=Microsoft-MIEngine-In-txvmdaq1.btr' '--stdout=Microsoft-MIEngine-Out-qsj1djkp.spc' '--stderr=Microsoft-
osoft-MIEngine-Pid-ipjvqtet.gc4' '--dbgExe=C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-r
mi'
Binary Tree Inorder Traversal: 2 7 9 10 11 13 18 28
Minimum Value Node: 2
Maximum Value Node: 28
Node with value 18 found in the tree.
Binary Tree after deleting node with value 7: 2 9 10 11 13 18 28
Binary Tree after swapping values 28 and 13: 2 9 10 11 28 18 13
PS D:\Manish\DS SPIT>
```

Algorithm:

1. Define the structure for a binary tree node (struct TreeNode).

2. Implement a function to create a new node:

- createNode(int data) -> struct TreeNode*

- Allocate memory for a new node.
- Initialize the data, left, and right pointers.
- Return the new node.

3. Implement a function to insert a node into the binary tree:

- insertNode(struct TreeNode* root, int data) -> struct TreeNode*
  - If the root is NULL, create a new node with the given data and return it.
  - If data is less than the root's data, recursively insert into the left subtree.
  - If data is greater than the root's data, recursively insert into the right subtree.
  - Return the updated root.

4. Implement a function to find the minimum value node in the binary tree:

- findMinValueNode(struct TreeNode* root) -> struct TreeNode*
  - Recursively navigate to the left child until you reach a leaf node.
  - Return the leaf node (minimum value node).

5. Implement a function to find the maximum value node in the binary tree:

- findMaxValueNode(struct TreeNode* root) -> struct TreeNode*
  - Recursively navigate to the right child until you reach a leaf node.
  - Return the leaf node (maximum value node).

6. Implement a function to display the binary tree using inorder traversal:

- displayTree(struct TreeNode* root) -> void
  - If the root is not NULL:
    - Recursively call displayTree on the left subtree.
    - Print the data of the current node.
    - Recursively call displayTree on the right subtree.

7. Implement a function to delete a node with a specific value from the binary tree:

- deleteNode(struct TreeNode* root, int data) -> struct TreeNode*
  - If the root is NULL, return it.
  - If data is less than the root's data, recursively delete from the left subtree.
  - If data is greater than the root's data, recursively delete from the right subtree.
  - If the data matches the root's data:
    - Handle cases where the node has one or no children.
    - For nodes with two children, replace the data with the minimum value from the right subtree and delete that node.
  - Return the updated root.

8. Implement a function to find a node with a specific value:

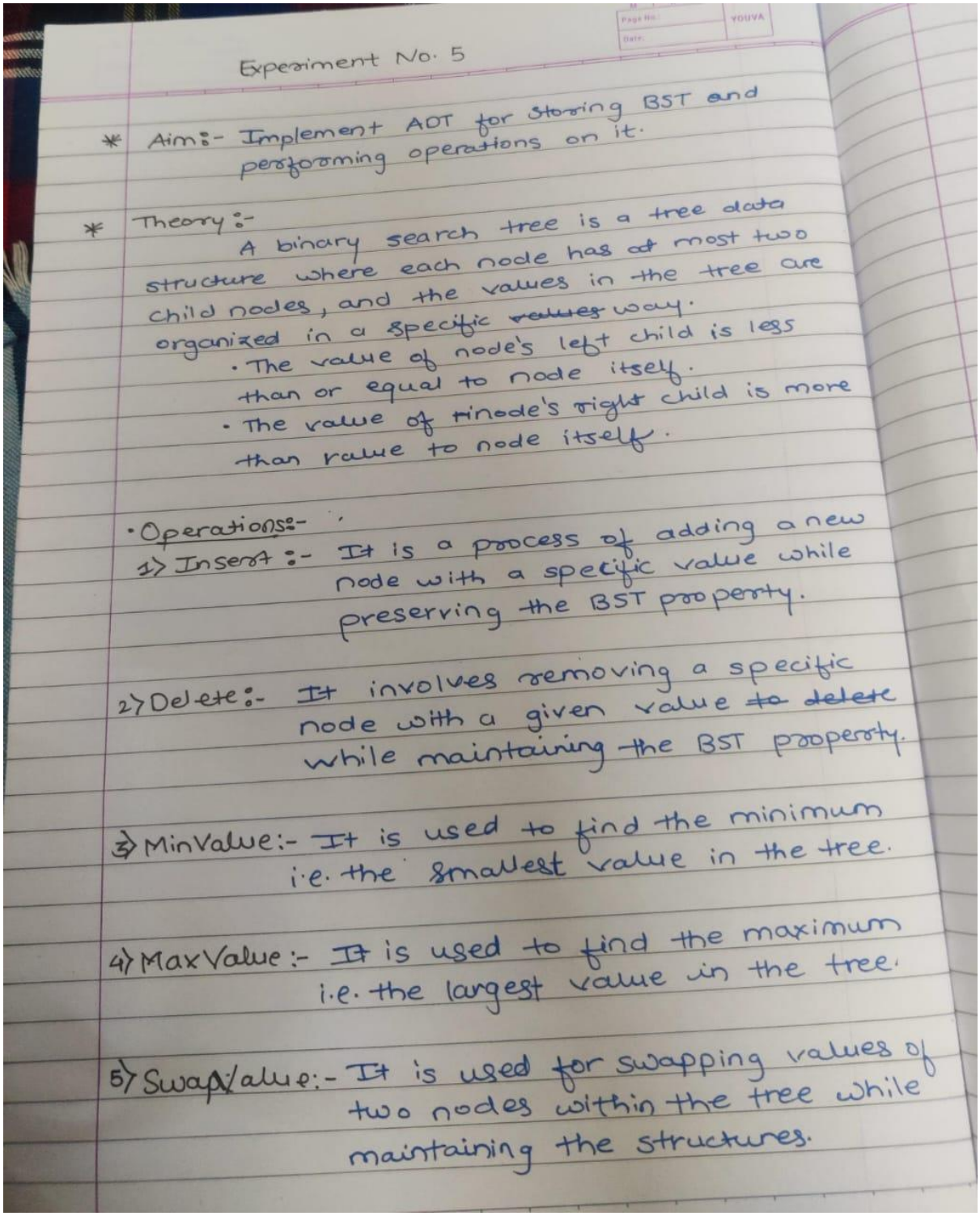
- findNode(struct TreeNode* root, int data) -> struct TreeNode*
  - If the root is NULL or its data matches the search data, return it.
  - If data is less than the root's data, recursively search in the left subtree.
  - If data is greater than the root's data, recursively search in the right subtree.

9. Implement a function to swap values of two nodes with specific values:

- swapNodeValues(struct TreeNode* root, int value1, int value2) -> void
  - Find the nodes with value1 and value2 in the tree.
  - If either of the nodes is not found, display an error message.
  - Swap the values of the two nodes.

10. In the main function:

- Create an empty binary tree (root is NULL).
- Insert several nodes with values.
- Display the tree using inorder traversal.
- Find and print the minimum and maximum value nodes.
- Find and print whether a specific node with a value exists in the tree.
- Delete a node with a specific value and display the tree.
- Swap the values of two nodes and display the tree.
- Optionally, clean up memory by freeing allocated nodes.



Experiment No. 5

* Aim:- Implement ADT for storing BST and performing operations on it.

* Theory:-

A binary search tree is a tree data structure where each node has at most two child nodes, and the values in the tree are organized in a specific ~~values~~ way.

- The value of node's left child is less than or equal to node itself.
- The value of ~~n~~node's right child is more than value to node itself.

- Operations:-

1) Insert :- It is a process of adding a new node with a specific value while preserving the BST property.

2) Delete:- It involves removing a specific node with a given value ~~to delete~~ while maintaining the BST prooperty.

3) MinValue:- It is used to find the minimum i.e. the smallest value in the tree.

4) MaxValue :- It is used to find the maximum i.e. the largest value in the tree.

5) SwapValue:- It is used for swapping values of two nodes within the tree while maintaining the structures.

Conclusion:

Hence, by completing this experiment I came to know about implement ADT for storing a BST and performing operations on it.