Binary trees

Implementations

Node description

```
struct node
{
        int data_element;
        struct node *left, *right;
};
```

Create New Node function

```
struct node *new_node(int data_element)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    // Allocating memory to the node
    temp->data_element = data_element;
    temp->left = temp->right = NULL;
    return temp;
}
```

Binary tree traversal function

```
void display(struct node *root)
// A function for the inroder traversal of the binary tree
{
	if (root != NULL)
	{
		display(root->left);
		printf("%d ", root->data_element);
		display(root->right);
	}
}
```

Binary tree insert node function

```
struct node *insert(struct node *node, int data_element) // Function to insert a new node
    {
        int k;
        if (node == NULL) return new_node(data_element); // Return a new node if the tree if empty
        else {
                printf("Elementul curent: %d\n ", node->data_element);
                if (node->left != NULL) printf("La stanga: %d\n ", node->left->data_element);
                else printf("La stanga: Liber\n ");
                if (node->right != NULL) printf("La dreapta: %d\n ", node->right->data_element);
                else printf("La dreapta: Liber\n ");
                printf("\nAdaugare 1- stanga, 2 - dreapta\n"); scanf("%d", &k);
                switch(k)
                {      case 1: node->left = insert(node->left, data_element); break;
                    case 2: node->right = insert(node->right, data_element); break;
                }
            }
        return node;
}
```

Remove from RIGHT

Binary tree remove node function

```
struct node* excludere_dreapta (struct node *root)
{   struct node *temp = root, *tz;
    if (temp == NULL) { printf("\n Nimic de lichidat \n"); return; }
    else if(temp->right == NULL)
    {   tz = root; root = root->left; free(tz); return root; }
    else
        {  printf("Elementul curent: %d\n ", temp->data_element);
           if (temp->right != NULL && temp->right->right != NULL)
            {   printf("La dreapta: %d\n ", temp->right->data_element);
                excludere_dreapta(temp->right);
            }
            else if (temp->right != NULL && temp->right->right == NULL)
                {   // lichidam urmasul drept conectam din stanga
                    tz = temp->right; temp->right = tz->left; free(tz);
                }
        }
        return root;
 }
```

Binary tree processing main function

```
int main()
{
    int k, val;
    struct node *root = NULL;
    while(1)
    {
        printf("\n1 - adaugare nod\n");
        printf("2 - parcurgere preordine\n");
        printf("3 - parcurgere inordine\n");
        printf("4 - parcurgere postordine\n");
        printf("5 - excludere -ultimul nod\n");
        printf("6 - sfarsit executie\n");
        scanf("%d", &k);
```

Binary tree processing main function

```
switch(k){ case 1:
                {   printf("Valoare de adaugat : "); scanf("%d", &val);
                    if (root == NULL) root = insert(root, val);
                    else insert (root, val);
                    printf("\n %d \n", root->data_element);
                    break;
                }
           case 2: display (root); break;
           case 5: root = excludere_dreapta(root); break;
           case 6: return 0;
         }
    }
}
```