## Insertion in BST

```
  New = (struct node *) malloc(sizeof (struct node));
    New -> data = item;
    New -> left = NULL;
    New -> right = NULL;
    if(root == NULL)
    {
        root = New;
        root -> left = NULL;
        root -> right = NULL;
    }
    else
    {
        parentptr = NULL;
        nodeptr = root;
        while(nodeptr != NULL)
        {
            parentptr = nodeptr;
            if(item < nodeptr->data)
            {
                nodeptr = nodeptr -> left;
            }
            else
            {
                nodeptr = nodeptr -> right;
            }
        }
        if(item < parentptr -> data)
        {
            parentptr -> left = New;
        }
        else
        {
            parentptr -> right = New;
        }
    }

    }
}
```

## Deletion in BST

```
void searchKey(Node* &curr, int key, Node* &parent)
{

	while (curr != nullptr && curr->data != key)
	{
		parent = curr;

		if (key < curr->data)
			curr = curr->left;
		else
			curr = curr->right;
	}
}

Node* minimumKey(Node* curr)
{
	while (curr->left != nullptr) {
		curr = curr->left;
	}
	return curr;
}
void deleteNode(Node*& root, int key)
{
	Node* parent = nullptr;

	Node* curr = root;

	searchKey(curr, key, parent);

	if (curr == nullptr)
		return;
```

```
// Case 1: node to be deleted has no children i.e. it is a leaf node
	if (curr->left == nullptr && curr->right == nullptr)
	{
		if (curr != root)
		{
			if (parent->left == curr)
				parent->left = nullptr;
			else
				parent->right = nullptr;
		}
		else
			root = nullptr;


		free(curr);
	}

	// Case 2: node to be deleted has two children
	else if (curr->left && curr->right)
	{
		Node* successor = minimumKey(curr->right);

		int val = successor->data;

		deleteNode(root, successor->data);

		curr->data = val;
	}

	// Case 3: node to be deleted has only one child
	else
	{

		Node* child = (curr->left)? curr->left: curr->right;
```

```
            if (curr != root)
            {
                if (curr == parent->left)
                    parent->left = child;
                else
                    parent->right = child;
            }

            else
                root = child;

            free(curr);
        }
}
```