EX.NO : 9(A)	B TREE
DATE :	

PROGRAM STATEMENT:

To write the findMin module of the B tree in CPP.

ALGORITHM:

1. Start the program.
2. Define the function findMin: Accept the root node (myNode) of a B-Tree as input.
3. Traverse the Leftmost Path: Initialize a pointer curr to the root node and traverse down the leftmost child (link[0]) until reaching a leaf node (i.e., a node where link[0] is NULL).
4. Print Minimum Value: Once the leftmost node is found, print its second value (val[1]), which is the minimum value in the node's key array.
5. End the function.
6. End the program.

PROGRAM:

```
voidfindMin(BTreeNode*myNode){
   BTreeNode *curr = myNode;
   while(curr->link[0]!=NULL)
   {
   curr =curr->link[0];
   }
   cout<<endl<<"Min="<<curr->val[1];
}
```

OUTPUT:

	Input	Expected	Got	
✔	5 23 45 1 2 5	1 2 5 23 45 Min=1	1 2 5 23 45 Min=1	✔
✔	9 12 45 78 44 2 3 909 123 567	2 3 12 44 45 78 123 567 909 Min=2	2 3 12 44 45 78 123 567 909 Min=2	✔
✔	4 123 567 345 90	90 123 345 567 Min=90	90 123 345 567 Min=90	✔

Passed all tests! ✔

RESULT:

Thus, the C++ program to write the findMin module of the B tree in CPP is created successfully.

EX.NO : 9(B)	B+ TREE
DATE :	

PROGRAM STATEMENT:

To write the findParent Module of the BPlus Tree in CPP.

ALGORITHM:

1. Start the program.
2. Define the findParent function: Accept the cursor (current node) and child (the child node whose parent is to be found) as input.
3. Base Case Check: If the cursor is a leaf node or if the first child node is a leaf, return NULL because the parent cannot be found further.
4. Search for the Child: Loop through all child pointers of the current node (cursor). If the current child pointer matches the child node, set parent to the cursor and return it.
5. Recursive Search: If the child is not found in the current node, recursively call findParent on each child node of the cursor to search deeper in the tree.
6. Return Parent: If a parent is found, return the parent node; otherwise, return NULL if no parent is found.
7. End the program.

PROGRAM:

```
Node *BPTree::findParent(Node*cursor, Node*child)
{
Node *parent ;
   if (cursor->IS_LEAF || (cursor->ptr[0])->IS_LEAF){
      return NULL;
   }
   for (int i= 0; i< cursor->size+1; i++)
   {
    if(cursor->ptr[i]==child)
      {
      parent = cursor;
         returnparent;
      }
      else
      {
      parent =findParent(cursor->ptr[i], child);
         if (parent != NULL){
            returnparent;
         }
```

```
        }
      }
      return parent;
}
```

OUTPUT:

	Input	Expected	Got	
✔	5 1 2 3 45 20 20	3 1 2 3 20 45 Found	3 1 2 3 20 45 Found	✔
✔	10 1 3 5 50 10 20 100 75 80 15 5	5 20 75 1 3 5 10 15 20 50 75 80 100 Found	5 20 75 1 3 5 10 15 20 50 75 80 100 Found	✔
✔	15 10 1 5 20 30 15 100 90 75 80 120 150 110 25 130 1	90 10 20 1 5 10 15 20 25 30 90 120 75 80 90 100 110 120 130 150 Found	90 10 20 1 5 10 15 20 25 30 90 120 75 80 90 100 110 120 130 150 Found	✔

Passed all tests! ✔

RESULT:

Thus, the C++ program to write the findParent Module of the BPlus Tree in CPP is created successfully.

EX.NO : 9(C)	RED-BLACK TREE
DATE :	

PROGRAM STATEMENT:

To write the Right rotate module of the red black tree in CPP.

ALGORITHM:

1. Start the program.
2. Define rightrotate: Perform a right rotation on temp.
3. Reassign left and right children: Set left to temp->l and update temp->l to left->r. Update the parent pointer of temp->l if necessary.
4. Update parent of left: Set left->p to temp->p. If temp->p is NULL, set root to left.
5. Update parent's child pointer: Adjust temp->p->l or temp->p->r to point to left.
6. Complete rotation: Set left->r to temp, and temp->p to left.
7. End the program.

PROGRAM:

```
void rightrotate(node* temp)
{ node*left= temp->l;
  temp->l=left->r;
  if(temp->l)
    temp->l->p=temp;
    left->p=temp->p;
        if(!temp->p)
             root=left;
        else
        if(temp==temp->p->l)
        temp->p->l=left;
    else
         temp->p->r=left;
         left->r=temp;
         temp->p=left;
}
```

OUTPUT:

	Input	Expected	Got	
✔	5 50 10 20 5 65	Inorder Traversal of Created Tree Data=5 Color=1 Data=10 Color=0 Data=20 Color=0 Data=50 Color=0 Data=65 Color=1	Inorder Traversal of Created Tree Data=5 Color=1 Data=10 Color=0 Data=20 Color=0 Data=50 Color=0 Data=65 Color=1	✔
✔	9 50 12 24 15 7 9 60 55 52	Inorder Traversal of Created Tree Data=7 Color=1 Data=9 Color=0 Data=12 Color=0 Data=15 Color=0 Data=24 Color=0 Data=50 Color=0 Data=52 Color=1 Data=55 Color=1 Data=60 Color=0	Inorder Traversal of Created Tree Data=7 Color=1 Data=9 Color=0 Data=12 Color=0 Data=15 Color=0 Data=24 Color=0 Data=50 Color=0 Data=52 Color=1 Data=55 Color=1 Data=60 Color=0	✔
✔	15 100 50 65 75 10 20 35 5 120 110 115 105 102 112 140	Inorder Traversal of Created Tree Data=5 Color=1 Data=10 Color=0 Data=20 Color=0 Data=35 Color=1 Data=50 Color=0 Data=65 Color=0 Data=75 Color=0 Data=100 Color=1 Data=102 Color=0 Data=105 Color=1 Data=110 Color=0 Data=112 Color=1 Data=115 Color=0 Data=120 Color=0 Data=140 Color=1	Inorder Traversal of Created Tree Data=5 Color=1 Data=10 Color=0 Data=20 Color=0 Data=35 Color=1 Data=50 Color=0 Data=65 Color=0 Data=75 Color=0 Data=100 Color=1 Data=102 Color=0 Data=105 Color=1 Data=110 Color=0 Data=112 Color=1 Data=115 Color=0 Data=120 Color=0 Data=140 Color=1	✔

Passed all tests! ✔

RESULT:

Thus, the C++ program to write the Right rotate module of the red black tree in CPP is created successfully.

EX.NO : 9(D)	SPLAY TREE
DATE :	

PROGRAM STATEMENT:

To write the right rotate module of splay tree in CPP.

ALGORITHM:

1. Start the program.
2. Define rightRotate function: It takes a node x as input.
3. Set y as the left child of x: Assign y = x->left.
4. Reassign the left child of x: Set x->left = y->right.
5. Update the right child of y: Set y->right = x.
6. Return y: The new root after the rotation is y

PROGRAM:

```
node *rightRotate(struct node *x)
{ node*y=x->left;
x->left=y->right;
y->right=x;
return y;


}
```

OUTPUT:

	Input	Expected	Got	
✔	5 10 20 30 5 6 30	Preorder traversal of the BST is 10 5 6 20 30 Preorder traversal of the modified Splay tree is 30 20 10 5 6	Preorder traversal of the BST is 10 5 6 20 30 Preorder traversal of the modified Splay tree is 30 20 10 5 6	✔
✔	7 10 20 30 5 7 9 99 30	Preorder traversal of the BST is 10 5 7 9 20 30 99 Preorder traversal of the modified Splay tree is 30 20 10 5 7 9 99	Preorder traversal of the BST is 10 5 7 9 20 30 99 Preorder traversal of the modified Splay tree is 30 20 10 5 7 9 99	✔
✔	9 100 120 140 105 50 60 55 51 10 51	Preorder traversal of the BST is 100 50 10 60 55 51 120 105 140 Preorder traversal of the modified Splay tree is 51 50 10 100 55 60 120 105 140	Preorder traversal of the BST is 100 50 10 60 55 51 120 105 140 Preorder traversal of the modified Splay tree is 51 50 10 100 55 60 120 105 140	✔

RESULT:

Thus, the C++ program to write the right rotate module of splay tree in CPP is created successfully.