Date: / /

Lab - 5

```
Struct btreenode {
        int val[MAX+1], count;
        btreeNode *link[MAX+1];
};

btreeNode *root = NULL;

btreeNode *createNode (int val, btreeNode *child)
{
    btreeNode *newNode = new btreeNode;
    newNode -> val[1] = val;
    newNode -> count = 1;
    newNode -> link[0] = root;
    newNode -> link[1] = child;
      return newNode;
}

void addvaltoNode (int val, int pos, btreeNode *node, btreeNode *child)
{
    int j = node -> count;
    while (j > pos) {
        node -> val[j+1] = node -> val[j];
        node -> val[j+1] = node -> link[j];
        j--;
    }
    node -> val[j+1] = val;
    node -> link[j+1] = child;
    node -> count++;
}
```

```
void splitnode (int val, int *pval, int pos, btree Node *node,
                btree Node *child, btree Node **newnode){

    int median, j;
    if (pos > MIN)
        median = MIN + 1;
    else
        median = MIN;
    *newNode = new btree Node;
    j = median + 1;
    while (j <= MAX){
     (*newNode) -> val[j - median] = node -> val[j];
     (*newNode) -> link[j - median] = node -> link[j];
     j++;
    }
    node -> count = median;
    (*newNode) -> count = MAX - median;
    if (pos <= MIN){
        addValToNode (val, pos, node, child);
    }
    else{ addValToNode (val, pos - median, *newNode, child);
    }
    *pval = node -> val[node -> count];
    (*newNode) -> link[0] = node -> link[node -> count];
    node -> count--;
}
int SetValueInNode (int val, int *pval, btree Node *node,
                    btree Node **child){
    int pos;
    if (!node){
        *pval = val;
```

```
    *child = NULL;
    return 1;
  }
 if (val < node->val[i])
      pos = 0
  else
    for (pos = node->count; val < node->val[pos] && pos > 1; pos--)
      if (val == node->val[pos])
          cout << "Duplication not allowed";
        return 0;
  if (setValueInNode(val, pval, node->link[pos], child)) {
    if (node->count < MAX){
        addValToNode(*pval, pos, node, *child)
    } else
    { splitNode(*pval, pval, pos, node, *child)
      return 1;
    }
  } return 0;
}
void insertion(int val) {
    int flag, i;
    struct Node *child;
  flag = setValueInNode(val, &i, root, &child);
   if (flag)
      root = createNode(i, child);
}
void copySuccessor(struct Node *myNode, int pos){
    struct Node *dummy;
    dummy = myNode->link[pos];

    for(; dummy->link[0] != NULL;)
        dummy = dummy->link[0];
```

```
        myNode->val[pos] = dummy->val[i];
}
void removeVal (btreeNode *myNode, int pos)
{   int i = pos+1;
    while(i<= myNode->count){
      myNode->val[i-1] = myNode->val[i];
      myNode->link[i-1] = myNode->link[i];
      i++;
    }

  myNode->count--;
}
void doRightShift (btreeNode *myNode, int pos)
{
    btreeNode *x = myNode->link[pos];
    int j = x->count;
    while (j>0) {
    x->val[j+1] = x->val[j];
    x->val[j+1] = x-> link[j];
    }

    x->val[1] = myNode->val[pos];
    x->link[1] = x-> link[0];
    x->count++;

    x = myNode->link(pos-1);
    my node->val[pos] = x-> val[x->count];
    myNode-> link[pos] = x-> link[x->count];
    x->count--;
    return;
}
void doLeftShift (btreeNode *myNode, int pos){
    int j = 1;
```

```
btreeNode *x = myNode -> link[pos-1];

x -> count++;
x -> val[x -> count] = myNode -> val[pos];
x -> link[x -> count] = myNode -> link[pos] -> link[0];

x = myNode -> link[pos];
myNode -> val[pos] = x -> val[1];
x -> link[0] = x -> link[1];
x -> count--;
while (j <= x -> count){
  x -> val[j] = x -> val[j+1];
  x -> link[j] = x -> link[j+1];
  j++;
}
return;
}
void mergeNodes(btreeNode, myNode, int pos)
  int j = 1;
btreeNode *x = myNode -> link[pos], *x2 = myNode -> link
                                          [pos-1];
x2 -> count++;
x2 -> val[x2 -> count] = myNode -> val[pos];
x2 -> link[x2 -> count] = myNode -> link[0];
while (j <= x1 -> count){
    x2 -> count++;
    x2 -> val[x2 -> count] = x1 -> val[j];
    x2 -> link[j] = myNode -> link[j+1];
    j++;
}
myNode -> count--;
free(x1);
}
```