ETS Struktur Data

Nama : Reynold Putra Merdeka
NRP : 5027211034

```
void insertRoot(string parent){
        _root = __createNode(parent);
}
```

Fungsi ini digunakan untuk memasukkan Root (node pertama), didalamnya terdapat fungsi createNode untuk membuat noe baru

```
BSTNode* __createNode(string val) {
    BSTNode *newNode = (BSTNode*) malloc(sizeof(BSTNode));
    newNode->key = val;
    newNode->left = newNode->right = NULL;
    return newNode;
}
```

Fungsi ini untuk membuat node baru yang nantinya dapat dimasukkan ke dalam BST

```
void insertRight(string parent, string child) {
    if (!find(child)) {
        if(find(parent)){
            BSTNode *temp = __search(_root, parent);
            temp->right = __createNode(child);
            _size++;
        }
    }
}

void insertLeft(string parent, string child) {
    if (!find(child)) {
        if(find(parent)){
            BSTNode *temp = __search(_root, parent);
            temp->left = __createNode(child);
            _size++;
        }
    }
}
```

Fungsi insert digunakan untuk memasukkan Node ke dalam BST. Saat memanggil fungsi ini, menggunakan argumen nama child dan parent. Nantinya parent akan dicari terlebih dahulu, kemudian membuat node baru untuk child dan dihubungkan pada parent node.

```
bool find(string val) {
    BSTNode *temp = __search(_root, val);
    if (!temp) return false;
    if (temp->key == val) return true;
    else return false;
}
```

Fungsi find digunakan untuk testing apakah value tersebut sudah ada di dalam BST atau belum

```
BSTNode* __search(BSTNode *root, string val) {
    if (root == 0) return NULL;
    if (root->key == val ) return root;

    BSTNode *newNode1 = __search(root->left, val);
    if(newNode1 != 0) return newNode1;
    BSTNode *newNode2 = __search(root->right, val);
    if(newNode2 != 0) return newNode2;

    return NULL;
}
```

Fungsi Search digunakan untuk menelusuri node yang memiliki key/value tertentu. Fungsi ini berjalan secara recursive menelusuri left tree kemudian right tree

```
void __inorder(BSTNode *root) {
    if (root) {
        __inorder(root->left);
        cout << root->key << endl;
        __inorder(root->right);
    }
}
```

Fungsi print transversal inorder dimana mencetak key/value dari tree paling kiri terlebih dahulu kemudian ke parent dan ke tree kanan dan mencari left tree bila ada

```
void __postorder(BSTNode *root) {
    if (root) {
        __postorder(root->left);
        __postorder(root->right);
```

```
            cout << root->key << endl;
        }
    }
```

Fungsi print transversal postorder dimana mencetak key/value dari paling kiri kemudian mencari ke right tree dan menuju tree paling kiri lagi

```
    void __preorder(BSTNode *root) {
        if (root) {
            cout << root->key << endl;
            __preorder(root->left);
            __preorder(root->right);
        }
    }
```

Fungsi print transversal postorder dimana mencetak key.value parent terlebih dahulu dan kemudian berpindah ke child left tree, apabila sudah diakhir maka pindah ke right tree