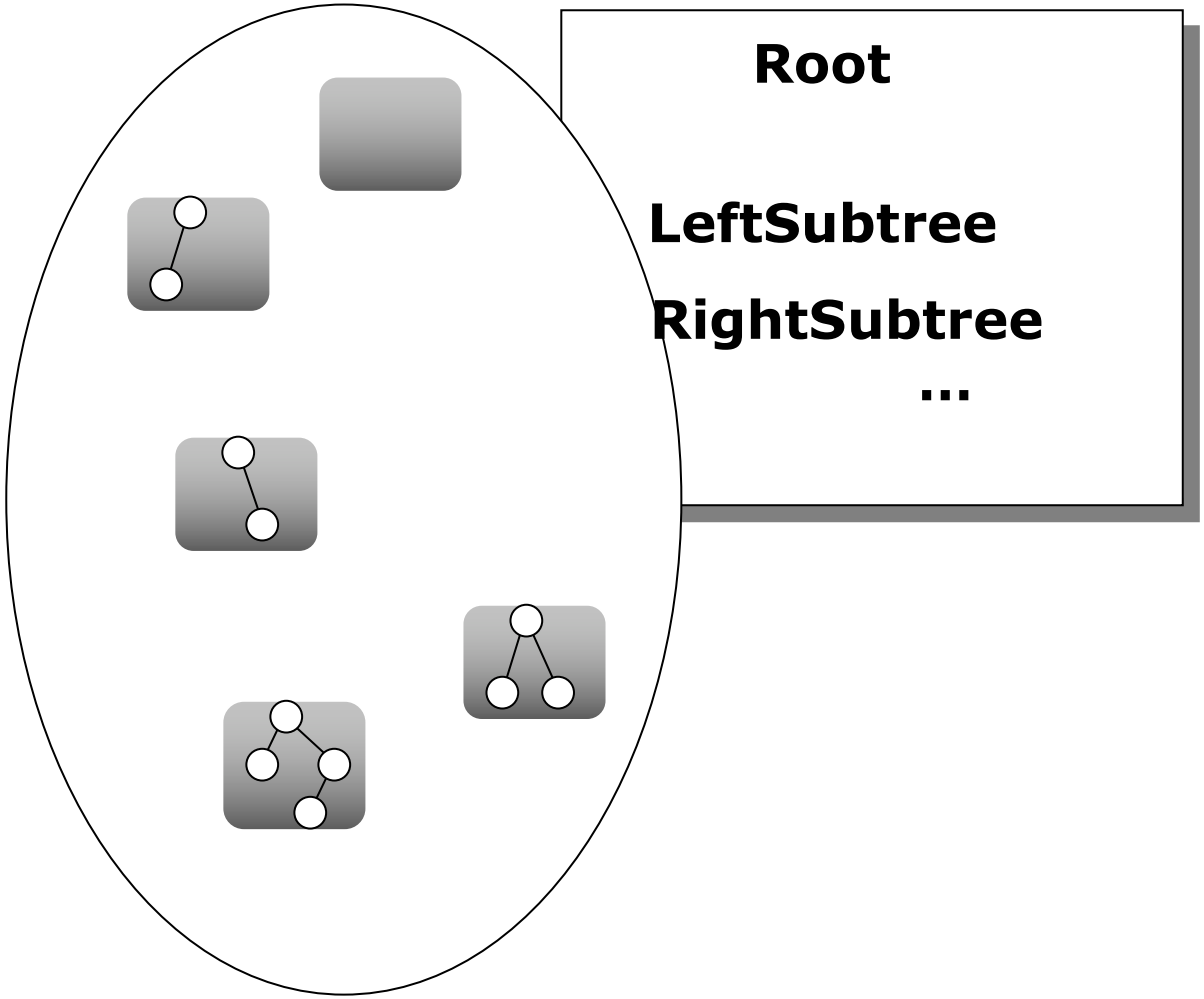
The Binary Tree ADT

As we have been doing, we define Binary Trees and Tree Nodes to be Abstract Data Types.



```
/* File: BinaryTree.h */
#include <stdlib.h>

typedef struct BinaryTreeCDT *BinaryTreeADT;
typedef struct TreeNodeCDT *TreeNodeADT;
#define SpecialErrNode (TreeNodeADT) NULL

BinaryTreeADT NonemptyBinaryTree(TreeNodeADT,
   BinaryTreeADT, BinaryTreeADT);
BinaryTreeADT EmptyBinaryTree(void);
BinaryTreeADT LeftSubtree(BinaryTreeADT);
BinaryTreeADT RightSubtree(BinaryTreeADT);
int TreeIsEmpty(BinaryTreeADT);
TreeNodeADT Root(BinaryTreeADT);
char *GetNodeKey(TreeNodeADT);
```

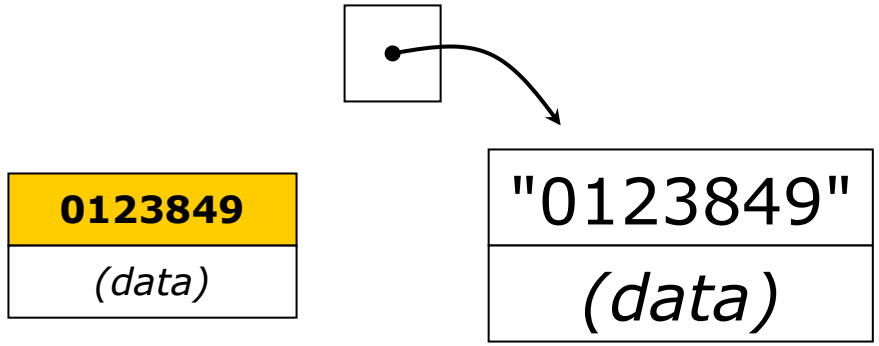
```
/* File: BinaryTree.h */
#include <stdlib.h>
#include <stdbool.h>
typedef struct BinaryTreeCDT *BinaryTreeADT;
typedef struct TreeNodeCDT *TreeNodeADT;
#define SpecialErrNode (TreeNodeADT) NULL

BinaryTreeADT NonemptyBinaryTree(TreeNodeADT,
   BinaryTreeADT, BinaryTreeADT);
BinaryTreeADT EmptyBinaryTree(void);
BinaryTreeADT LeftSubtree(BinaryTreeADT);
BinaryTreeADT RightSubtree(BinaryTreeADT);
bool TreeIsEmpty(BinaryTreeADT);
TreeNodeADT Root(BinaryTreeADT);
char *GetNodeKey(TreeNodeADT);
```

How to implement the Tree Node ADT and the Binary Tree ADT?

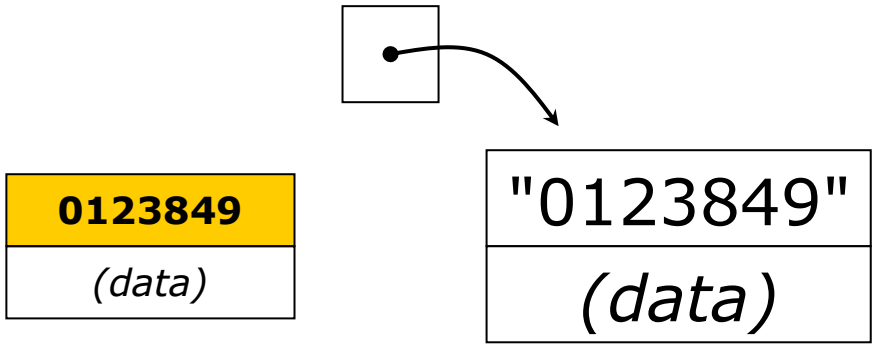
Implementation of the Tree Node ADT

We consider tree nodes with character strings as keys, and integers as data.

```
typedef struct TreeNodeCDT *TreeNodeADT;
struct TreeNodeCDT {
   char* key;
   int nodeData;
}
```



```
typedef struct TreeNodeCDT *TreeNodeADT;
struct TreeNodeCDT {
   char* key;
   int nodeData;
}
```



We need to add the following functions to BinaryTree.h:

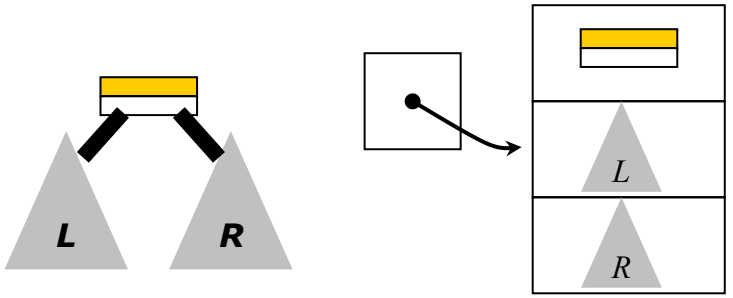
```
TreeNodeADT NewTreeNode(char*, int);
int GetNodeData(TreeNodeADT);
```

```
#include <stdlib.h>
#include <stdbool.h>
typedef struct BinaryTreeCDT *BinaryTreeADT;
typedef struct TreeNodeCDT *TreeNodeADT;
#define SpecialErrNode (TreeNodeADT) NULL
BinaryTreeADT NonemptyBinaryTree(TreeNodeADT,
   BinaryTreeADT, BinaryTreeADT);
BinaryTreeADT EmptyBinaryTree(void);
BinaryTreeADT LeftSubtree(BinaryTreeADT);
BinaryTreeADT RightSubtree(BinaryTreeADT);
bool TreeIsEmpty(BinaryTreeADT);
TreeNodeADT Root(BinaryTreeADT);
TreeNodeADT NewTreeNode(char*, int);
int GetNodeData(TreeNodeADT);
char *GetNodeKey(TreeNodeADT);
```

Implementation of the Binary Tree ADT

A nonempty binary tree consists of three components: the root, the left subtree, and the right subtree.

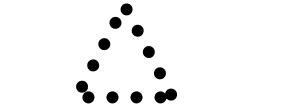
```
struct BinaryTreeCDT {
   TreeNodeADT rt;
   BinaryTreeADT lst;
   BinaryTreeADT rst;
}
```

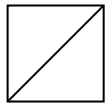


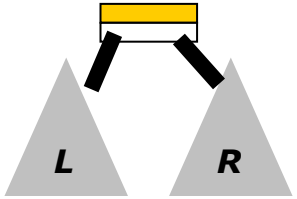
Concrete Implementation of the Binary Tree ADT

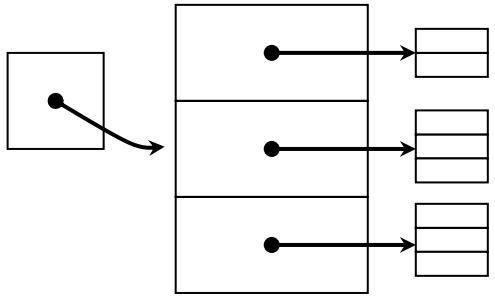
A binary tree consists of three components: the root, the left subtree, and the right subtree.

```
struct BinaryTreeCDT {
   TreeNodeADT rt;
   BinaryTreeADT lst;
   BinaryTreeADT rst;
}
```

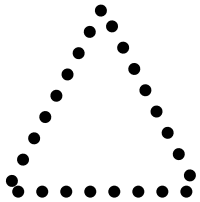


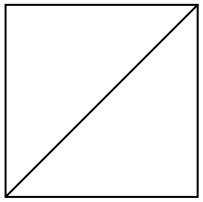






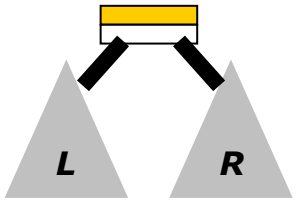
```
BinaryTreeADT EmptyBinaryTree() {
   return (BinaryTreeADT) NULL;
}
```

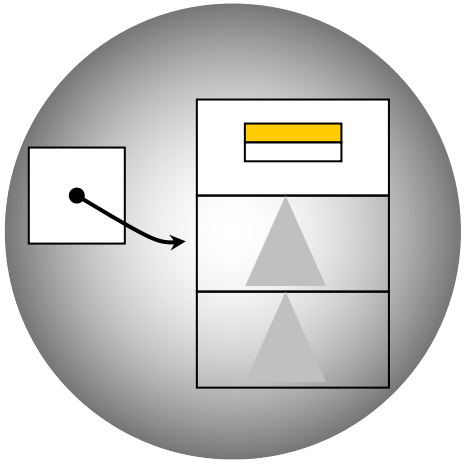


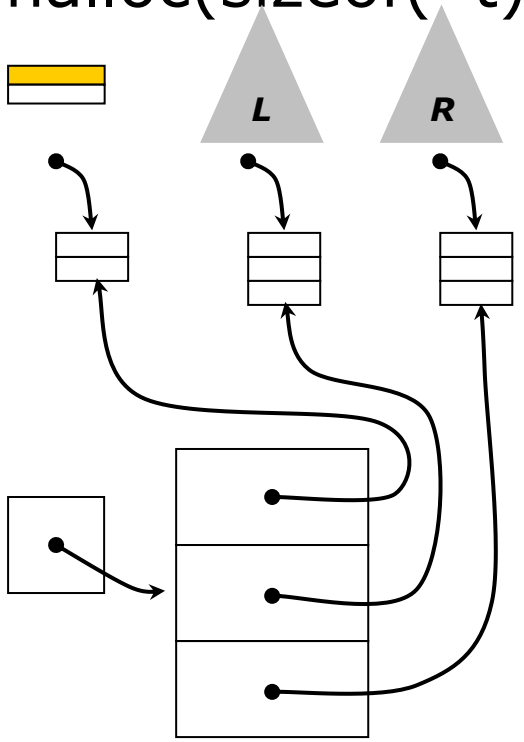


```
BinaryTreeADT t1 = NonemptyBinaryTree(n1, t1, t2);
```

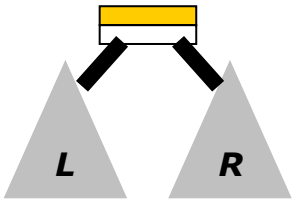
```
BinaryTreeADT NonemptyBinaryTree(TreeNodeADT N,
      BinaryTreeADT L, BinaryTreeADT R) {
   BinaryTreeADT t = (BinaryTreeADT) malloc(sizeof(*t));
   t->rt = N;
   t->lst = L;
   t->rst = R;
   return t;
}
```

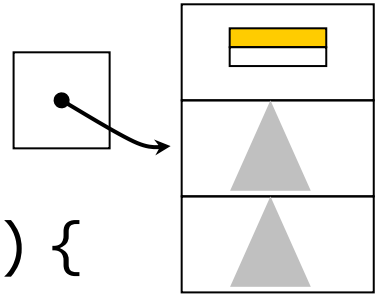


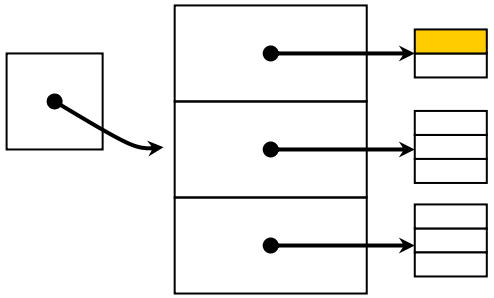




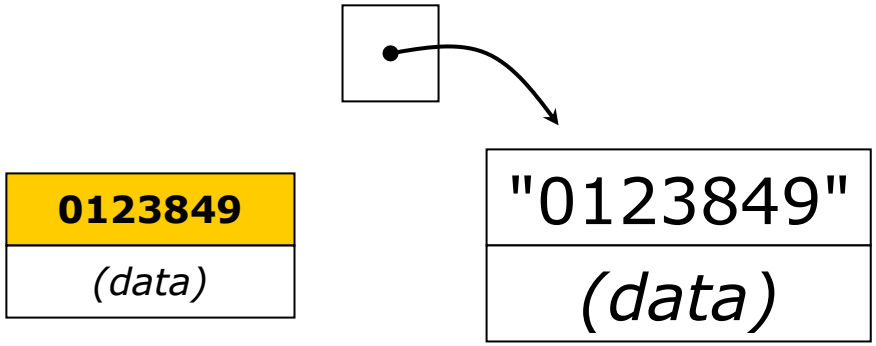
```
TreeNodeADT Root(BinaryTreeADT t) {
   if (TreeIsEmpty(t)) exit(EXIT_FAILURE);
   return t->rt;
}
BinaryTreeADT LeftSubtree(BinaryTreeADT t) {
   if (TreeIsEmpty(t)) exit(EXIT_FAILURE);
   return t->lst;
}
BinaryTreeADT RightSubtree(BinaryTreeADT t) {
   if (TreeIsEmpty(t)) exit(EXIT_FAILURE);
   return t->rst;
}
bool TreeIsEmpty(BinaryTreeADT t) {
   return t == (BinaryTreeADT) NULL;
}
```





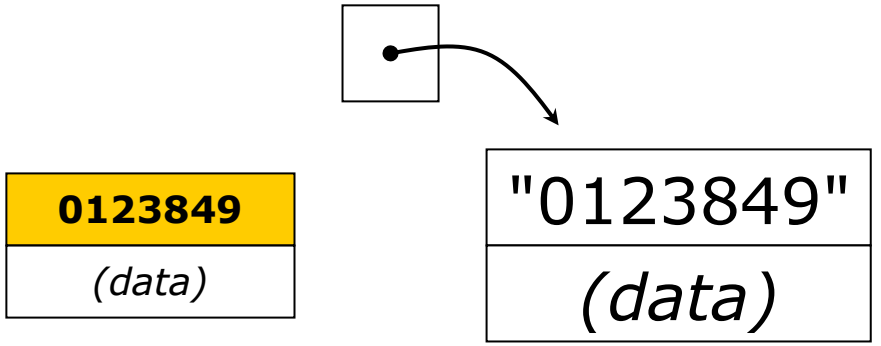


```
struct TreeNodeCDT {
   char* key;
   int nodeData;
}
```




```
TreeNodeADT NewTreeNode(char* k, int d) {
 TreeNodeADT N = (TreeNodeADT) malloc(sizeof(*N));
 N->key = (char*) malloc(sizeof(char)*(strlen(k)+1));
 strcpy(N->key, k); N->nodeData = d;
 return(N);
}
```

```
struct TreeNodeCDT {
   char* key;
   int nodeData;
}
```

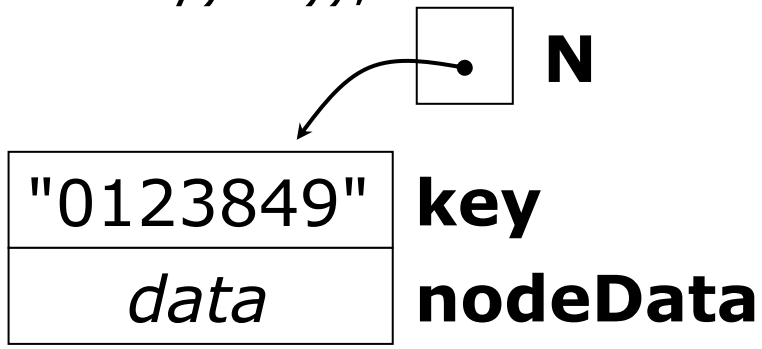



```
#include <string.h>
char *GetNodeKey(TreeNodeADT N) {
   char *k;
   if (N==SpecialErrNode) exit(EXIT_FAILURE);
   k = (char*) malloc(sizeof(char)*(strlen(N->key)+1));
   strcpy(k, N->key);
   return k;
}
```

Class Discussion

compare

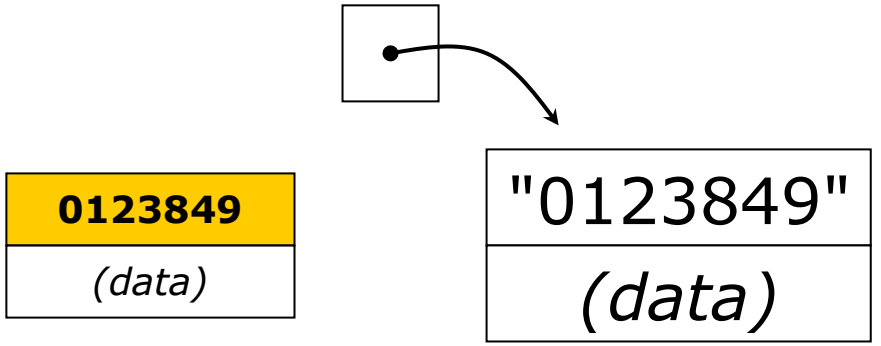
```
char *GetNodeKey(TreeNodeADT N) {
   char* k;
   if (N==SpecialErrNode) exit(EXIT_FAILURE);
   k = (char*) malloc(sizeof(char)*(strlen(N->key)+1));
   strcpy(k, N->key);
   return k;
}
```



with

```
char *GetNodeKey(TreeNodeADT N) {
   if (N== SpecialErrNode) exit(EXIT_FAILURE);
   return N->key;
}
```

```
struct TreeNodeCDT {
   char* key;
   int nodeData;
}
```




```
int GetNodeData(TreeNodeADT N) {
   return N->nodeData;
}
```

We now show the complete listing of the implementations.

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include "BinaryTree.h"

struct BinaryTreeCDT {
   TreeNodeADT rt;
   BinaryTreeADT lst, rst;
};

struct TreeNodeCDT {
   char* key;
   int nodeData;
};
```

```
BinaryTreeADT EmptyBinaryTree() {
   return (BinaryTreeADT) NULL;
}

BinaryTreeADT NonemptyBinaryTree(TreeNodeADT N,
      BinaryTreeADT L, BinaryTreeADT R) {
   BinaryTreeADT t = (BinaryTreeADT) malloc(sizeof(*t));
   t->rt = N; t->lst = L; t->rst = R;
   return t;
}
```

```
TreeNodeADT Root(BinaryTreeADT t) {
   if (TreeIsEmpty(t)) exit(EXIT_FAILURE);
   return t->rt;
}
BinaryTreeADT LeftSubtree(BinaryTreeADT t) {
   if (TreeIsEmpty(t)) exit(EXIT_FAILURE);
   return t->lst;
}
BinaryTreeADT RightSubtree(BinaryTreeADT t) {
   if (TreeIsEmpty(t)) exit(EXIT_FAILURE);
   return t->rst;
}
bool TreeIsEmpty(BinaryTreeADT t) {
   return t == (BinaryTreeADT) NULL;
}
```

```
TreeNodeADT NewTreeNode(char* k, int d) {
   TreeNodeADT N = (TreeNodeADT) malloc(sizeof(*N));
   N->key = (char*) malloc(sizeof(char)*(strlen(k)+1));
   strcpy(N->key, k); N->nodeData = d;
   return N;
}
char *GetNodeKey(TreeNodeADT N) {
   char *k;
   if (N==SpecialErrNode) exit(EXIT_FAILURE);
   k = (char*) malloc(sizeof(char)*(strlen(N->key)+1));
   strcpy(k, N->key); return k;
}
int GetNodeData(TreeNodeADT N) {
   return N->nodeData;
}
```