CS 2211
Systems Programming

Part Twelve:

Trees

Basic Tree Concepts

We begin with a discussion of the terminology used with trees

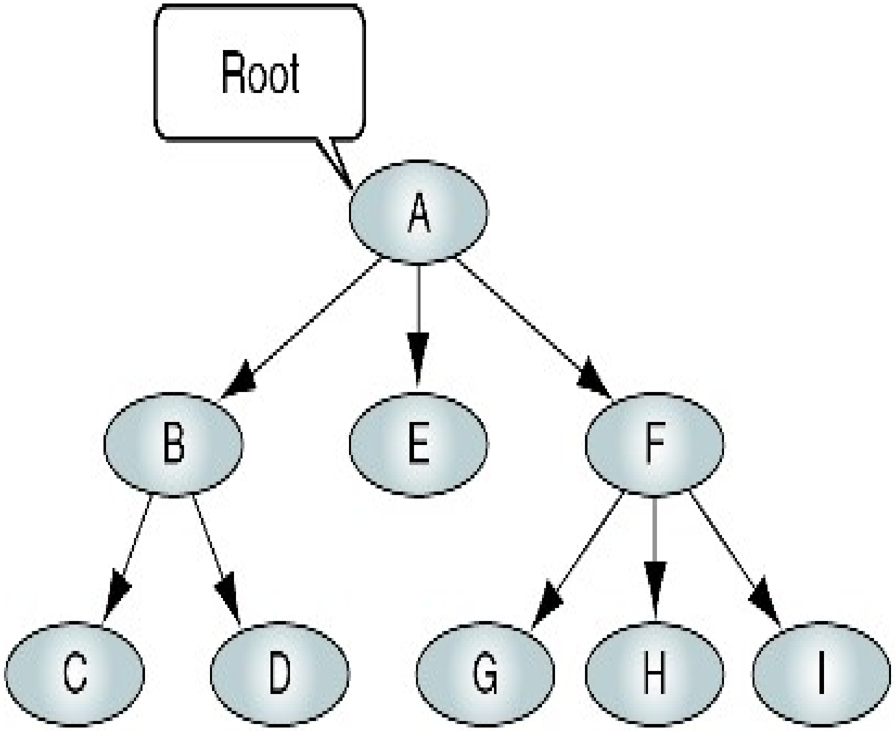
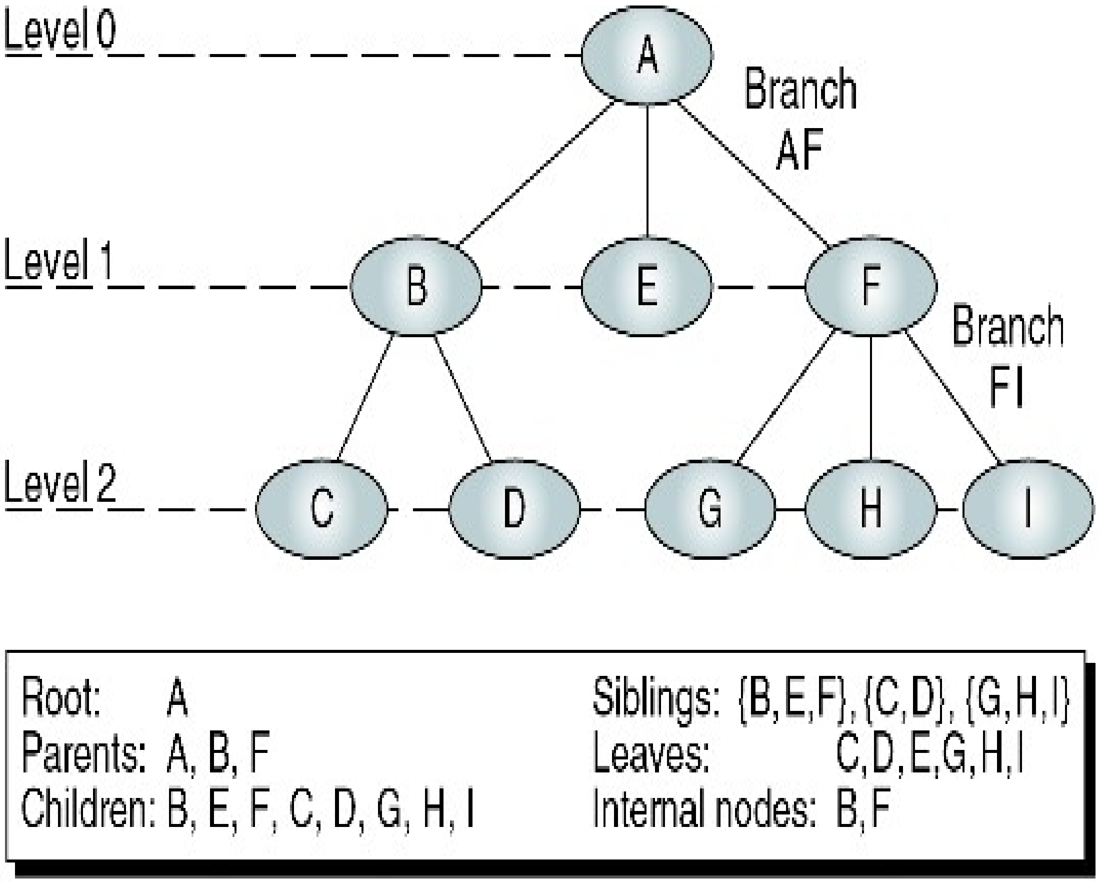


FIGURE 6-1 Tree



Root: A
Parents: A, B, F
Children: B, E, F, C, D, G, H, I

Siblings: {B,E,F}, {C,D}, {G,H,I}
Leaves: C,D,E,G,H,I
Internal nodes: B,F

FIGURE 6-2 Tree Nomenclature

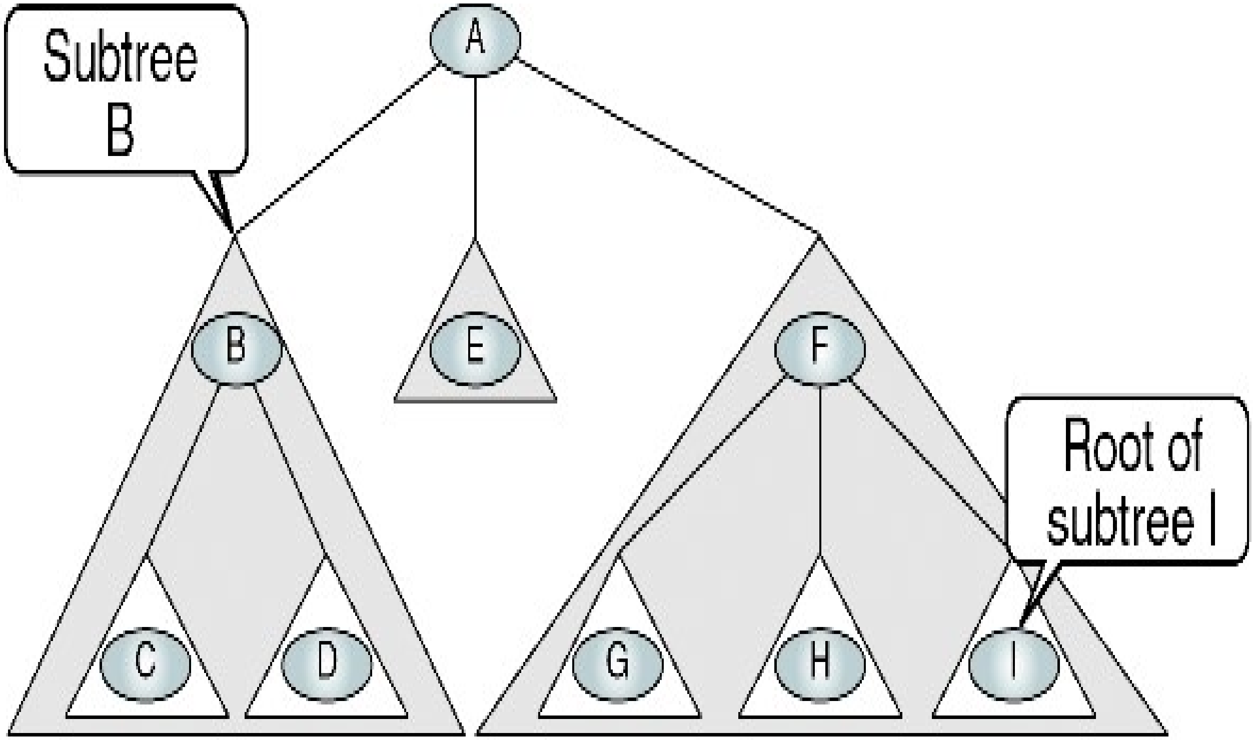


FIGURE 6-3 Subtrees

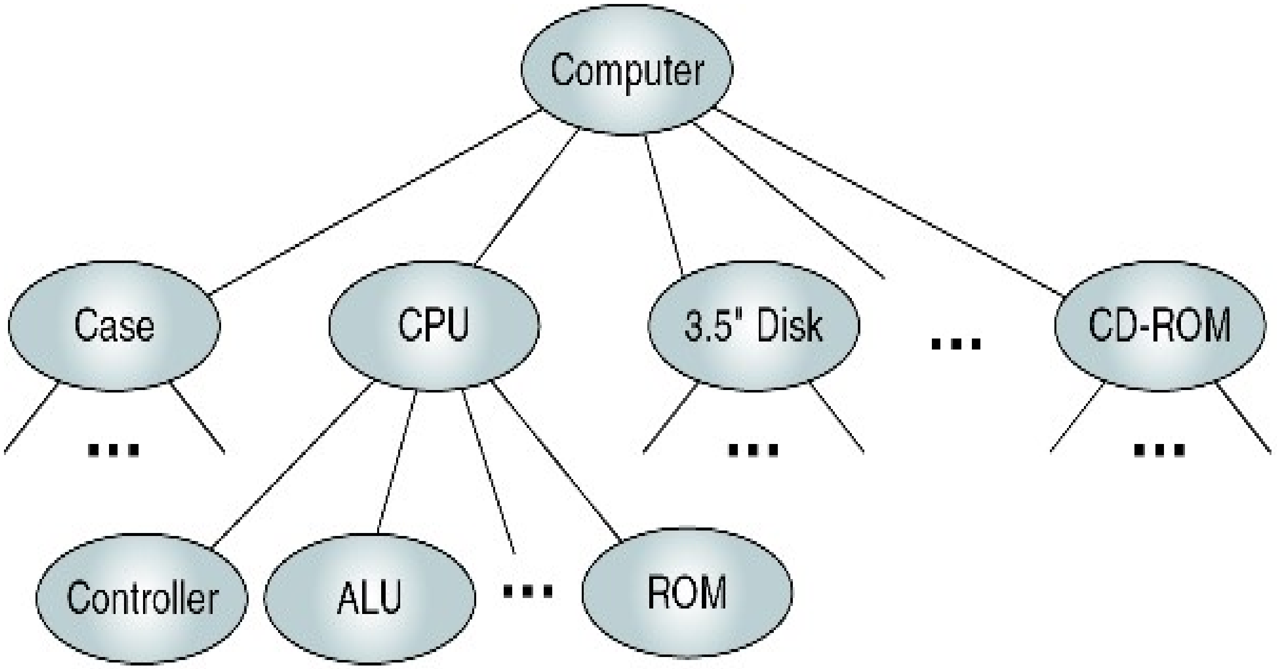


FIGURE 6-4 Computer Parts List as a General Tree

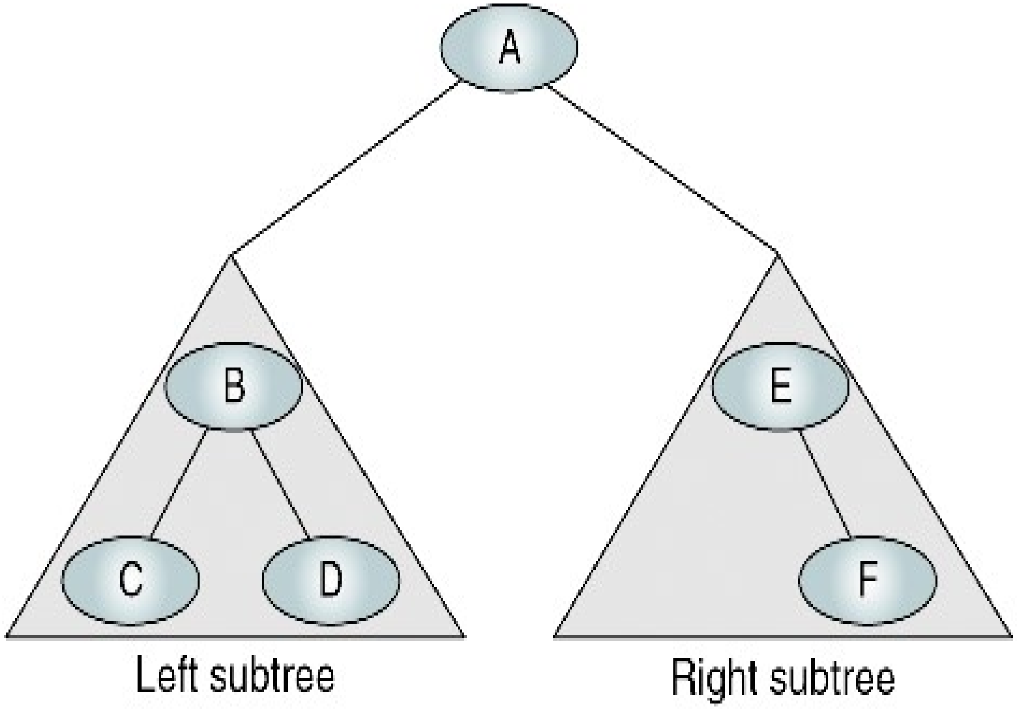


FIGURE 6-5 Binary Tree

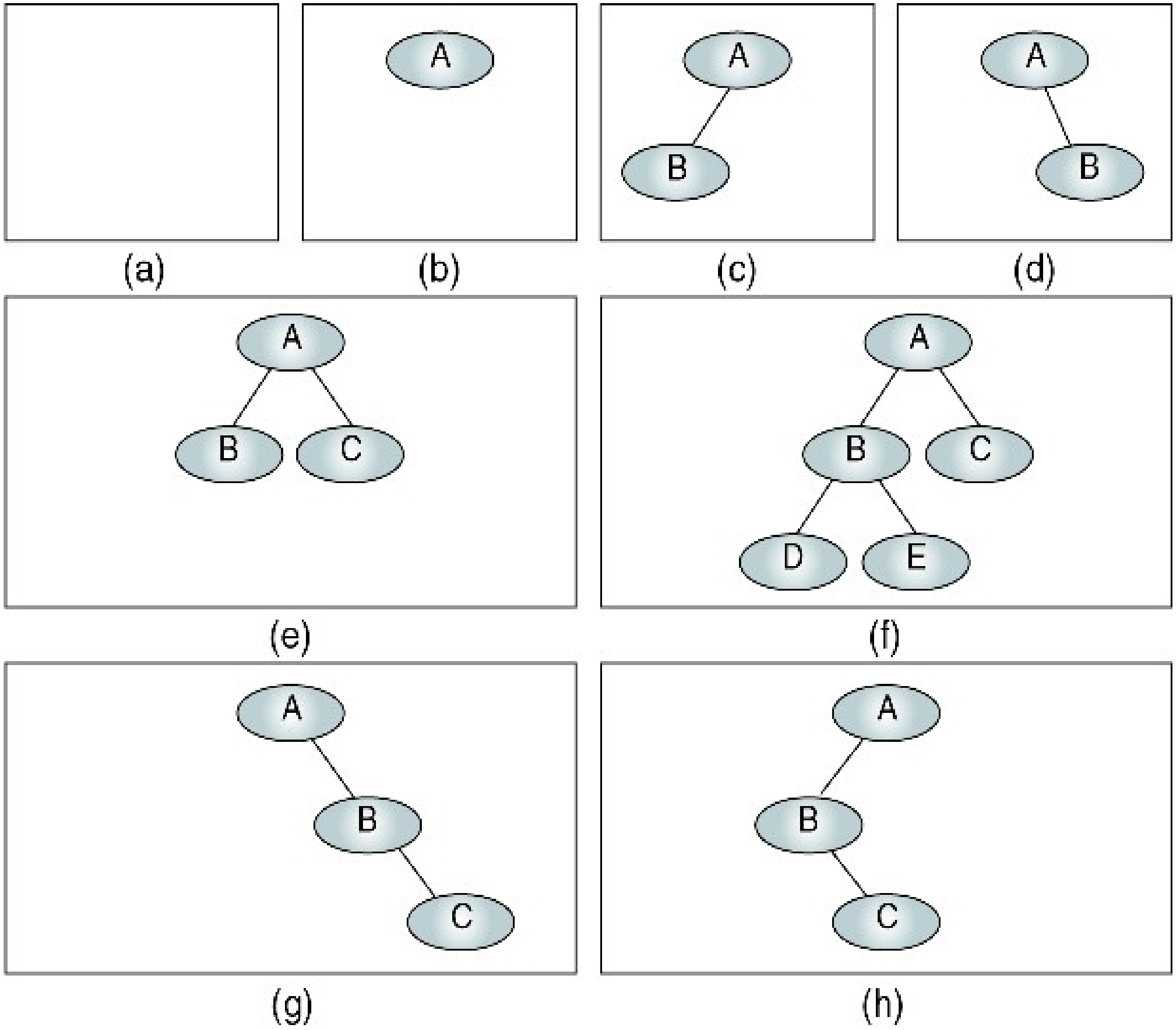
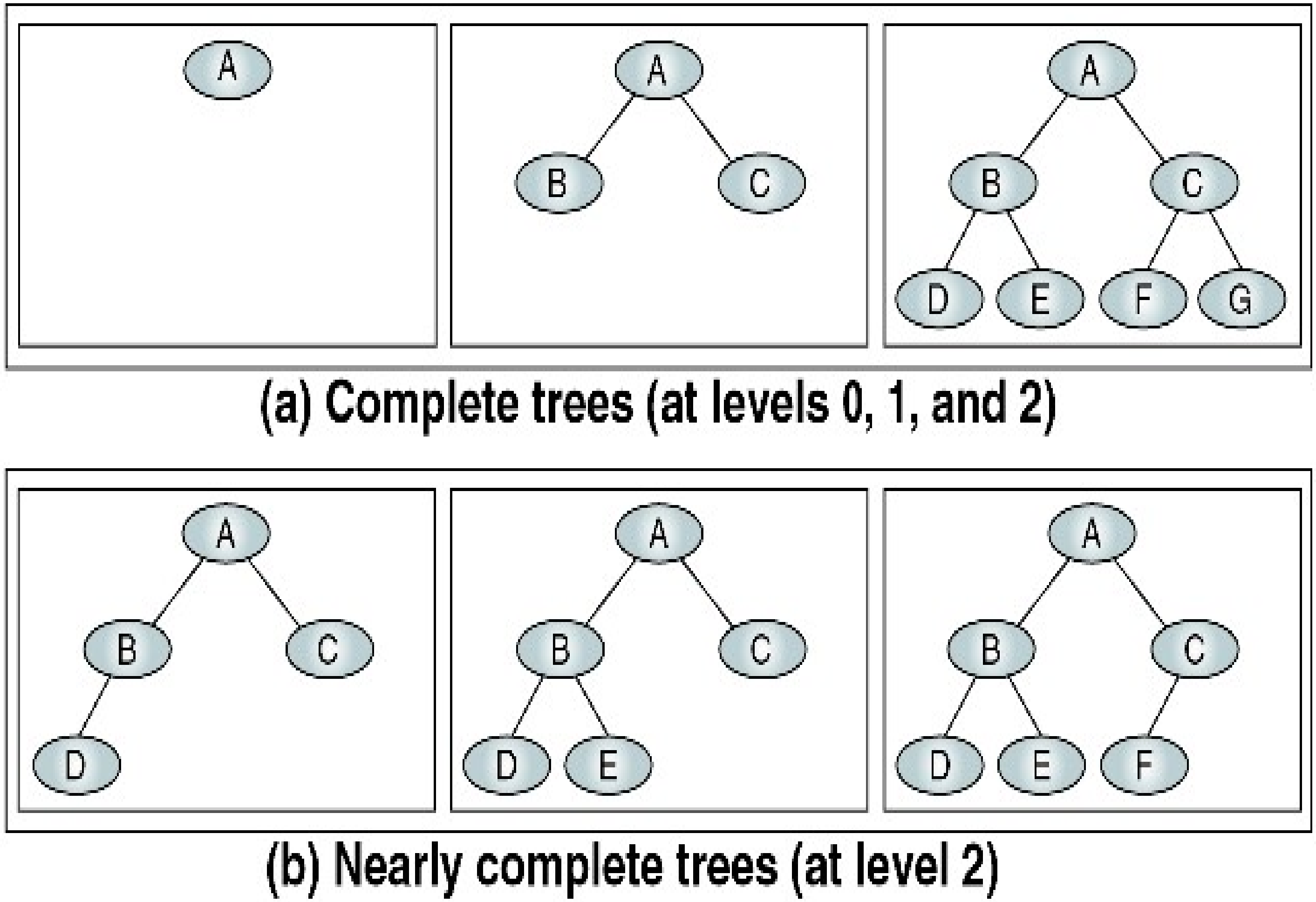


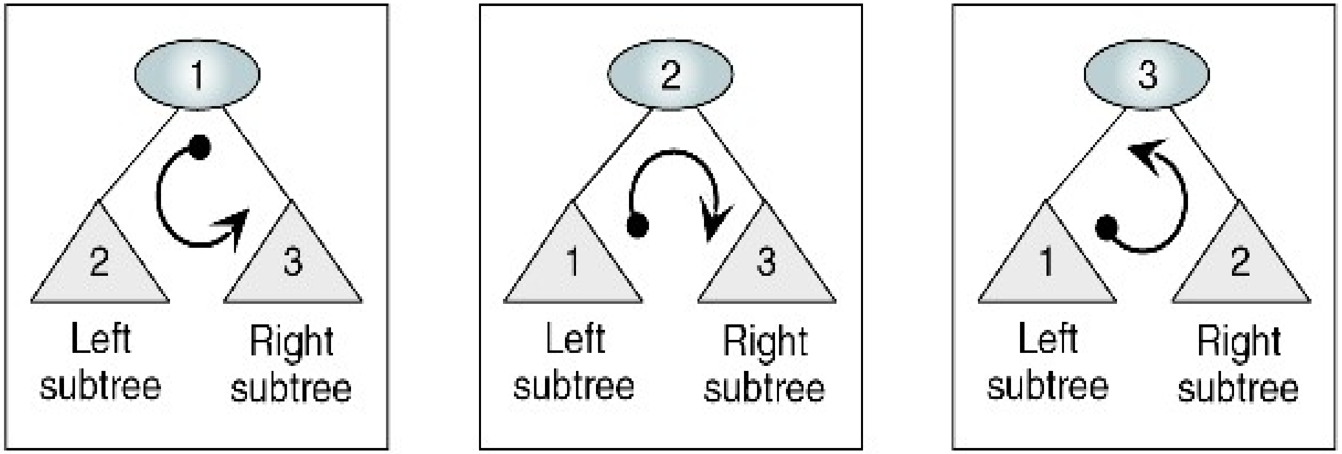
FIGURE 6-6 Collection of Binary Trees



(a) Complete trees (at levels 0, 1, and 2)

(b) Nearly complete trees (at level 2)

FIGURE 6-7 Complete and Nearly Complete Trees



(a) Preorder traversal (b) Inorder traversal (c) Postorder traversal

FIGURE 6-8 Binary Tree Traversals

Binary search trees provide an excellent structure for searching a list
- and -
at the same time for inserting and deleting data into the list.

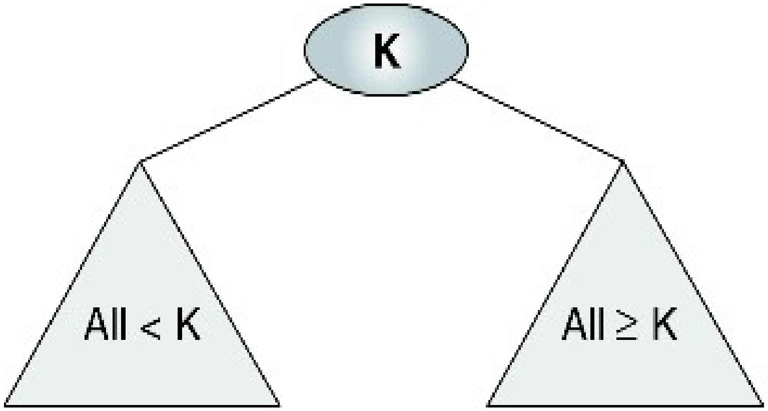


FIGURE 7-1 Binary Search Tree

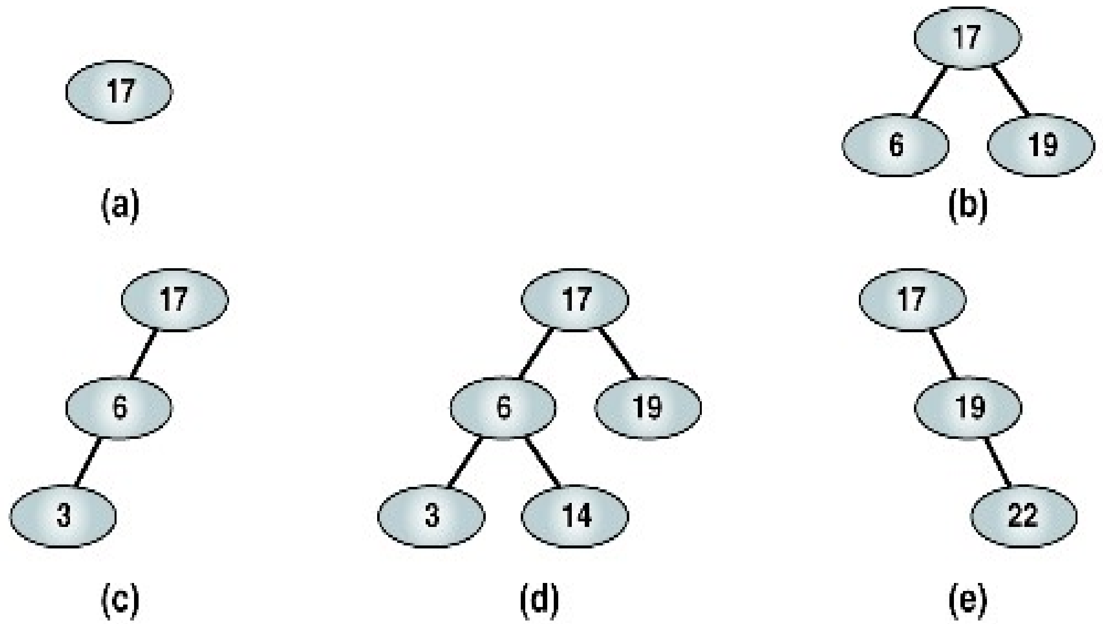


FIGURE 7-2 Valid Binary Search Trees

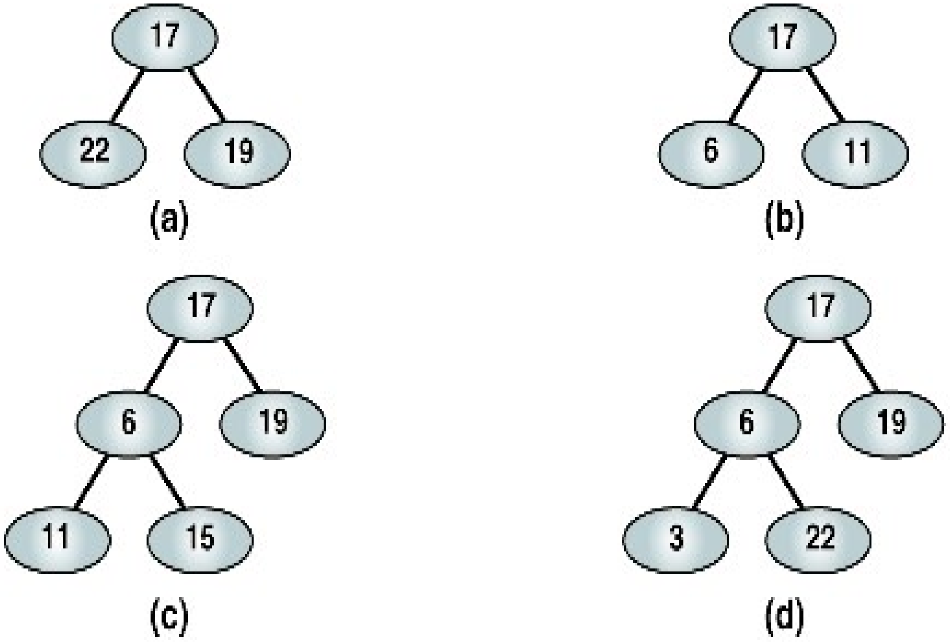


FIGURE 7-3 Invalid Binary Search Trees

Trees in C

END OF PART 1

7-2 BST Operations

We discuss four basic BST operations: traversal, search, insert, and delete; and develop algorithms for searches, insertion, and deletion.

- Traversals
- Searches
- Insertion
- Deletion

We discuss four basic BST operations: traversal, search, insert, and delete; and develop algorithms for searches, insertion, and deletion.

- Insertion
- Traversals
- Searches
- Deletion

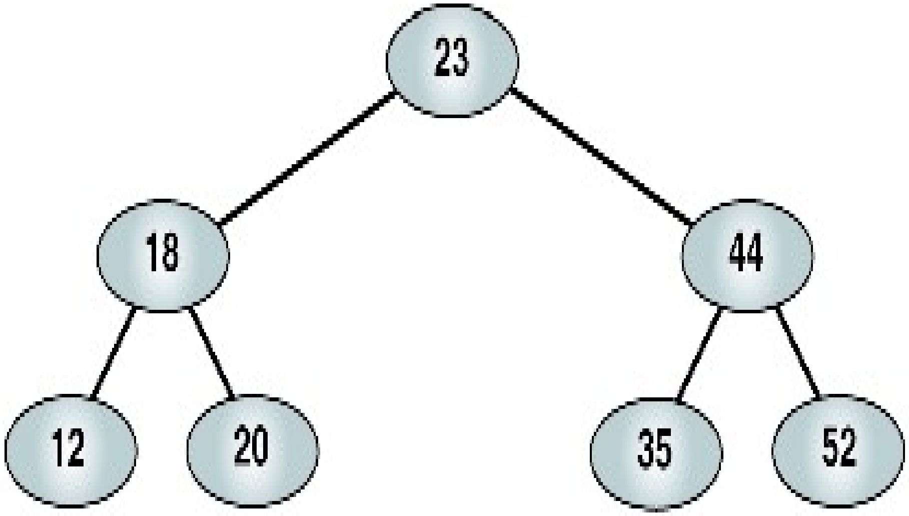
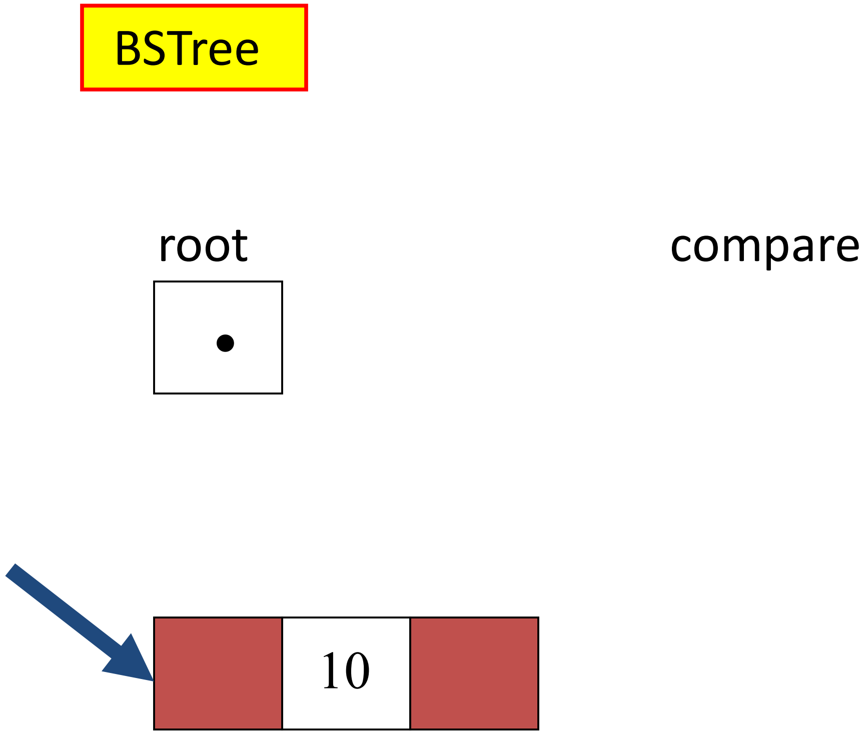


FIGURE 7-4 Example of a Binary Search Tree

To Add an Item to an Empty BST



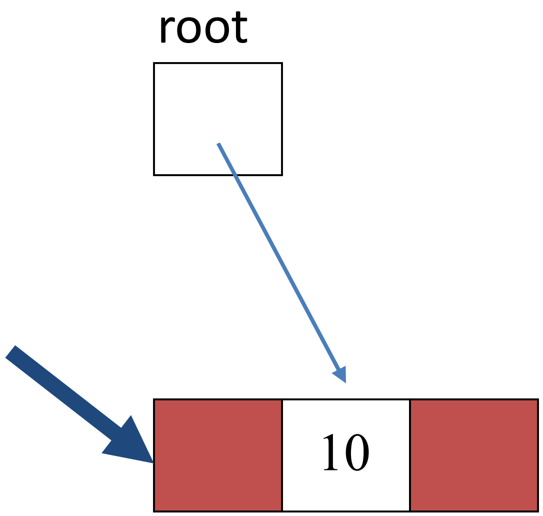
Build the new node, and put the new data item in it

To Add an Item to an Empty BST

BSTree

Build the new node, and put the new data item in it

compare



REMEMBER: the node definition for a doubly linked list was:

```
typedef struct node
{
         int                  data;
         struct node*    next;
         struct node*    prev;
} NODE;
```

… the node definition for a binary search tree is simply:

```
typedef struct node
{
         int                  data;
         struct node*    right;
         struct node*    left;
} NODE;
```

Based on the node definition – to create a new node:

```
typedef struct node
{
         int             data;
         struct node*    right;
         struct node*    left;
} NODE;
```

- to create a new node: :

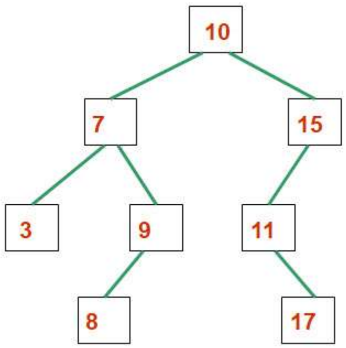
```
struct node *newNode(int item)
{
    struct node *leaf = (struct node *)malloc(sizeof(struct node));
    leaf->data = item;
    leaf->left = leaf->right = NULL;

    return leaf;
}
```

```
#include "defs.h"
```

```
int main()
{
// Local Definitions

   struct node *root = NULL;
   ...
   ...

}   // main
```

defs.h

```
#include<stdio.h>
#include<stdlib.h>

typedef struct
{
    struct node *root;
    int count;
} TREE;

//          Prototype Declarations

struct node *newNode(int );
struct node* insert(struct node*, int );
struct node * minValueNode(struct node* );
struct node* deleteNode(struct node*, int );
void printPostorder(struct node* );
void printInorder(struct node* );
void printPreorder(struct node* );
```



Label	Address	Value
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	

```
#include "defs.h‟


int main()
{
   ...
   struct node *root = NULL;
   root = insert(root, 10);
  ...

}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	

```
#include "defs.h‟


int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
   ...

}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	

```
#include "defs.h"


int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
   ...

}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	
	…	
node	560 - 563	NULL
data	564 - 567	10
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	

```
#include "defs.h”


int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
   ...

}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	
	…	
node	560 - 563	NULL
data	564 - 567	10
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	

```
#include "defs.h“


int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
   ...

}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	
	...	
node	560 - 563	NULL
data	564 - 567	10
	...	
item	620 - 623	10
	...	
	...	
	...	
	...	
	...	
	...	
	...	

```
#include "defs.h"


int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
   ...

}   // main
```

BSTInsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	
	…	
node	560 - 563	NULL
data	564 - 567	10
	…	
item	620 - 623	10
	…	
temp	660 - 663	1010
	…	
{ DM }	1010 - 1021	
	…	
	…	
	…	

```
#include "defs.h"


int main()
{
   ...
   struct node *root = NULL;
   root = insert(root, 10);
  ...

}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	
	…	
node	560 - 563	NULL
data	564 - 567	10
	…	
item	620 - 623	10
	…	
temp	660 - 663	1010
	…	
data	1010 - 1013	10
{ DM }	1014 - 1021	
	…	
	…	

```
#include "defs.h"


int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
   ...

}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	
	…	
node	560 - 563	NULL
data	564 - 567	10
	…	
item	620 - 623	10
	…	
temp	660 - 663	1010
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
	…	

```
#include "defs.h"


int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
   ...

}    // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

1010

Label	Address	Value
root	400 - 403	
	…	
node	560 - 563	NULL
data	564 - 567	10
	…	
item	620 - 623	10
	…	
temp	660 - 663	1010
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
	…	

```
#include "defs.h“


int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
   ...

}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

1010

Label	Address	Value
root	400 - 403	
	…	
node	560 - 563	NULL
data	564 - 567	10
	…	
	…	
	…	
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
	…	

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
   ...

}    // main
```

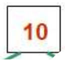
1010

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
	…	

```c
#include "defs.h"

int main()
{
   ...
   struct node *root = NULL;
   root = insert(root, 10);
   root = insert(root, 7);
   root = insert(root, 9);
   root = insert(root, 15);
  ...
}   // main
```



BSTinsert.c

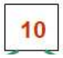
```c
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
	...	

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```



Label	Address	Value
root	400 - 403	1010
	...	
node	560 - 563	1010
data	564 - 567	7
	...	
	...	
	...	
	...	
	...	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
	...	

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}    // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
	…	
	…	
	…	
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
	…	

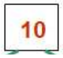
```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
node	590 - 593	NULL
data	594 - 597	7
	…	
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
	…	

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```



BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
node	590 - 593	NULL
data	594 - 597	7
	…	
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
	…	

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
node	590 - 593	NULL
data	594 - 597	7
item	610 -613	7
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
	…	

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
node	590 - 593	NULL
data	594 - 597	7
item	610 -613	7
temp	614 -617	1230
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
{ DM }	1230 - 1041	

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
node	590 - 593	NULL
data	594 - 597	7
item	610 -613	7
temp	614 -617	1230
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
data	1230 - 1233	7
{ DM }	1010 - 1021	

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
node	590 - 593	NULL
data	594 - 597	7
item	610 -613	7
temp	614 -617	1230
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
data	1230 - 1233	7
left	1234 - 1237	NULL
right	1238 - 1241	NULL

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

1230

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
node	590 - 593	NULL
data	594 - 597	7
item	610 -613	7
temp	614 -617	1230
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
data	1230 – 1233	7
left	1234 - 1237	NULL
right	1238 – 1241	NULL

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

1230

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
node	590 - 593	NULL
data	594 - 597	7
	…	
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	NULL
right	1018 - 1021	NULL
data	1230 – 1233	7
left	1234 - 1237	NULL
right	1238 – 1241	NULL

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

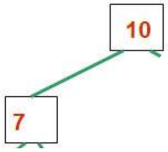
BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

1230

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
	…	
	…	
	…	
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	NULL
data	1230 – 1233	7
left	1234 - 1237	NULL
right	1238 – 1241	NULL

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```



BSTinsert.c

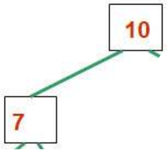
```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

10100

Label	Address	Value
root	400 - 403	1010
	…	
node	560 - 563	1010
data	564 - 567	7
	…	
	…	
	…	
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	NULL
data	1230 – 1233	7
left	1234 - 1237	NULL
right	1238 – 1241	NULL

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

10100

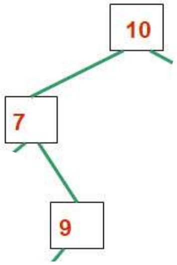


BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	NULL
data	1230 – 1233	7
left	1234 - 1237	NULL
right	1238 – 1241	NULL

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

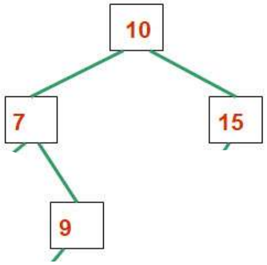


BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	NULL
data	1230 – 1233	7
left	1234 - 1237	NULL
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	NULL
right	1428 – 1431	NULL
	…	
	…	
	…	

```
#include "defs.h"

int main()
{
    ...
    struct node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 7);
    root = insert(root, 9);
    root = insert(root, 15);
   ...
}   // main
```

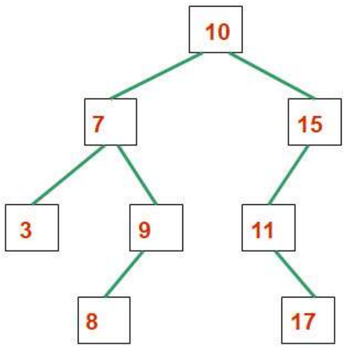


BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
        node->left = insert(node->left, data);
    else
        node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	NULL
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	NULL
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	NULL
right	1758 – 1761	NULL

```
#include "defs.h"

int main()
{
   ...
   root = insert(root, 11);
   root = insert(root, 3);
   root = insert(root, 8);
   root = insert(root, 14);
   ...
}   // main
```



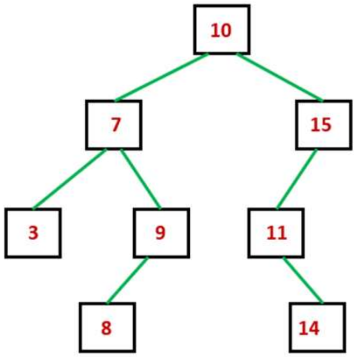
BSTinsert.c

```
// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data < node->data)
         node->left = insert(node->left, data);
    else
         node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}
```

Trees in C

END OF PART 2

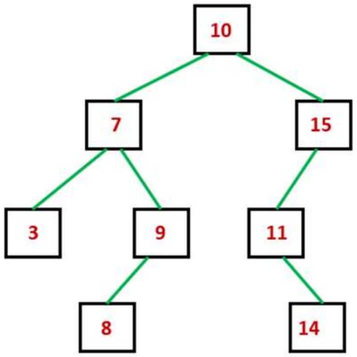
```
#include "defs.h"

int main()
{
    ...
    printf("\nPreorder traversal of binary tree is \n");
          printPreorder(root);
    ...
}   // main
```



BSTTransvers.c

```
/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct node* node)
{
            if (node == NULL)
                        return;

            /* first print data of node */
            printf("%d ", node->data);

            /* then recur on left sutree */
            printPreorder(node->left);

            /* now recur on right subtree */
            printPreorder(node->right);
}
```

```
#include "defs.h"

int main()
{
    ...
    printf("\nPreorder traversal of binary tree is \n");
          printPreorder(root);
    ...
}   // main
```



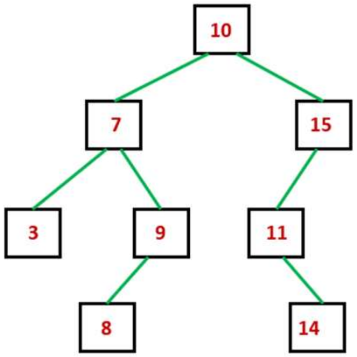
Label	Address	Value
root	400 - 403	1010
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

BSTTransvers.c

```
/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct node* node)
{
            if (node == NULL)
                        return;

            /* first print data of node */
            printf("%d ", node->data);

            /* then recur on left sutree */
            printPreorder(node->left);

            /* now recur on right subtree */
            printPreorder(node->right);
}
```

10 – 7 – 3 – 9 – 8 – 15 – 11 - 14

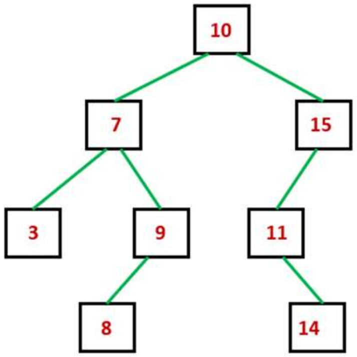
```c
#include "defs.h"

int main()
{
   ...
   printf("\nInorder traversal of binary tree is \n");
          printInorder(root);
   ...
}    // main
```



BSTTransvers.c

```c
void printInorder(struct node* node)
{
            if (node == NULL)
                        return;

            /* first recur on left child */
            printInorder(node->left);

            /* then print the data of node */
            printf("%d ", node->data);

            /* now recur on right child */
            printInorder(node->right);
}
```

```
#include "defs.h"

int main()
{
   ...
   printf("\nInorder traversal of binary tree is \n");
          printInorder(root);
   ...
}   // main
```



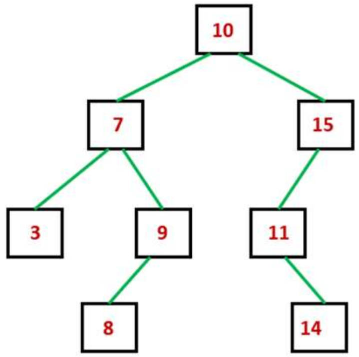
Label	Address	Value
root	400 - 403	1010
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

BSTTransvers.c

```
void printInorder(struct node* node)
{
            if (node == NULL)
                        return;

            /* first recur on left child */
            printInorder(node->left);

            /* then print the data of node */
            printf("%d ", node->data);

            /* now recur on right child */
            printInorder(node->right);
}
```

3 – 7 – 8 – 9 – 10 – 11 – 14 - 15

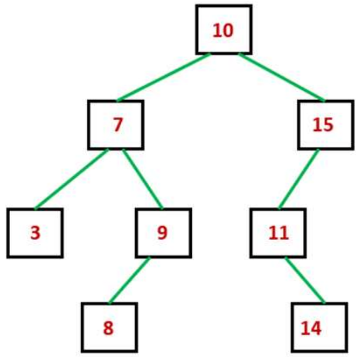
```
#include "defs.h"

int main()
{
   ...
   printf("\nPostorder traversal of binary tree is \n");
        printPostorder(root);
   ...
}   // main
```



BSTTransvers.c

```
/* Given a binary tree, print its nodes according to the
"bottom-up" postorder traversal. */
void printPostorder(struct node* node)
{
            if (node == NULL)
                        return;

            // first recur on left subtree
            printPostorder(node->left);

            // then recur on right subtree
            printPostorder(node->right);

            // now deal with the node
            printf("%d ", node->data);
}
```

```
#include "defs.h"

int main()
{
  ...
  printf("\nPostorder traversal of binary tree is \n");
       printPostorder(root);
  ...
}   // main
```



BSTTransvers.c

```
/* Given a binary tree, print its nodes according to the
"bottom-up" postorder traversal. */
void printPostorder(struct node* node)
{
            if (node == NULL)
                        return;

            // first recur on left subtree
            printPostorder(node->left);

            // then recur on right subtree
            printPostorder(node->right);

            // now deal with the node
            printf("%d ", node->data);
}
```

3 – 8 – 9 – 7 – 14 – 11 – 15 – 10

Label	Address	Value
root	400 - 403	1010
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

Trees in C

END OF PART 3

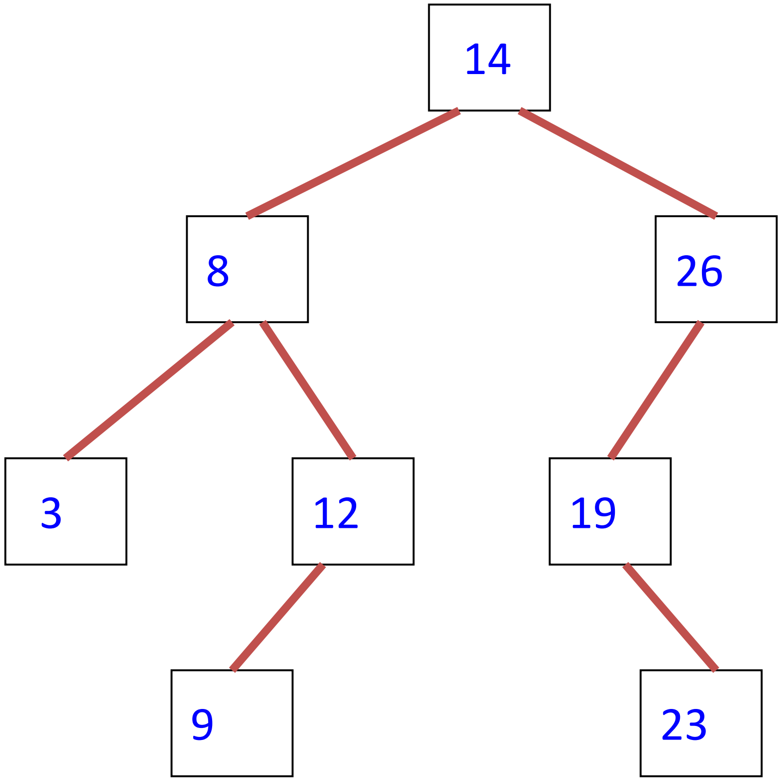
Searching in a BST

- Why is it called a binary ***search*** tree?
  - Data is stored in such a way, that it can be more ***efficiently*** found than in an ordinary binary tree

Searching in a BST

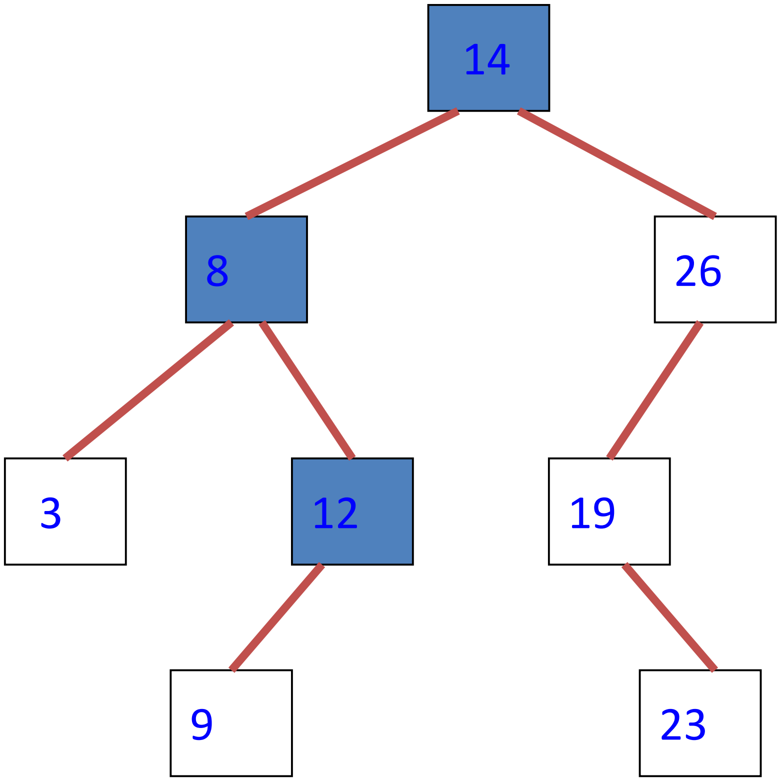
- **Algorithm to *search* for an item in a BST**
  - Compare data item to the root of the (sub)tree
  - If data item = data at root, found
  - If data item < data at root, go to the left; if there is no left child, data item is not in tree
  - If data item > data at root, go to the right; if there is no right child, data item is not in tree

Search Operation – a Recursive Algorithm

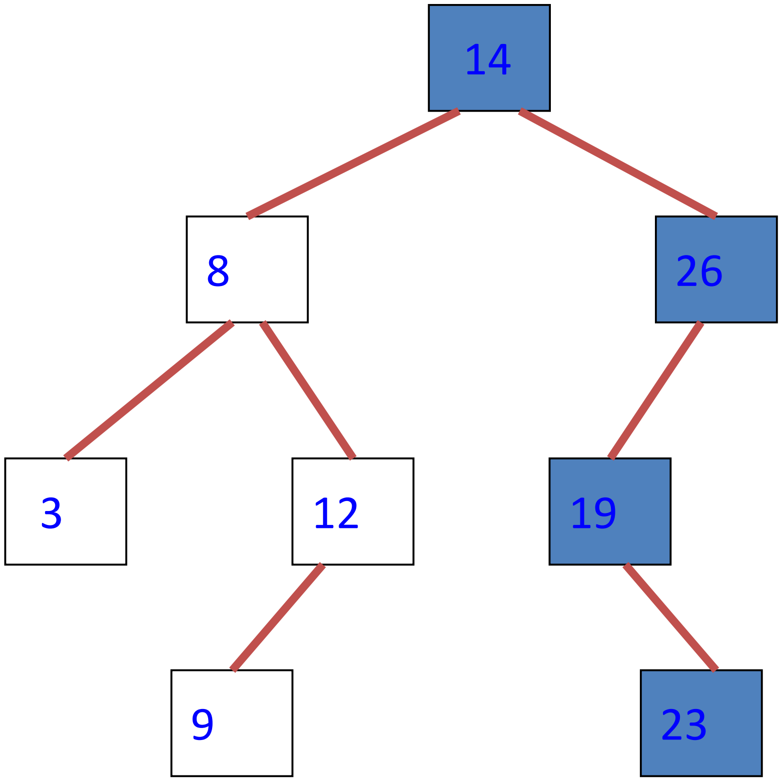


```
To search for a value k;

        returns true if found
    or false if not found

If the tree is empty, return false.

If k == value at root

    return true: we're done.

If k < value at root
    return result from search for k in
              the left subtree

Else
     return result from search for k in
            the right subtree.
```

Search Operation

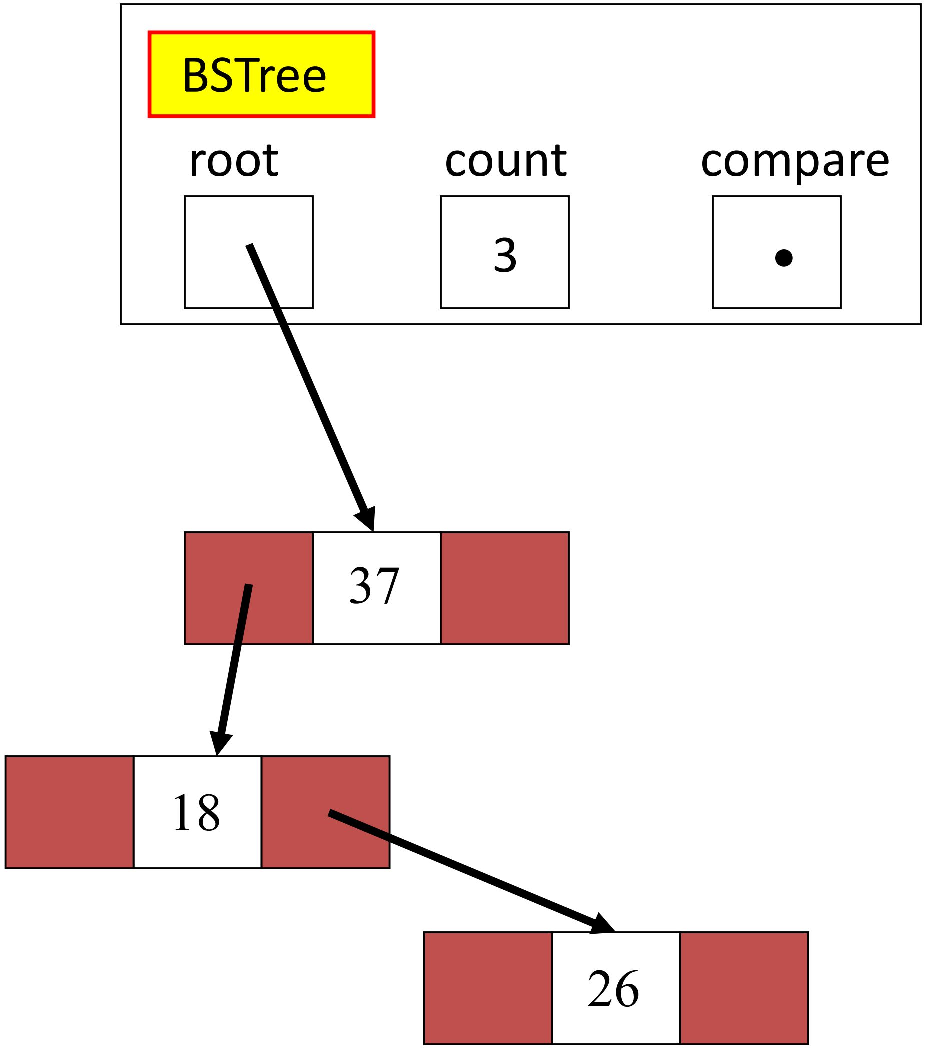


Search for 13: visited nodes are coloured yellow; return false when node containing 12 has no right child



Search for 22: return false when node containing 23 has no left child

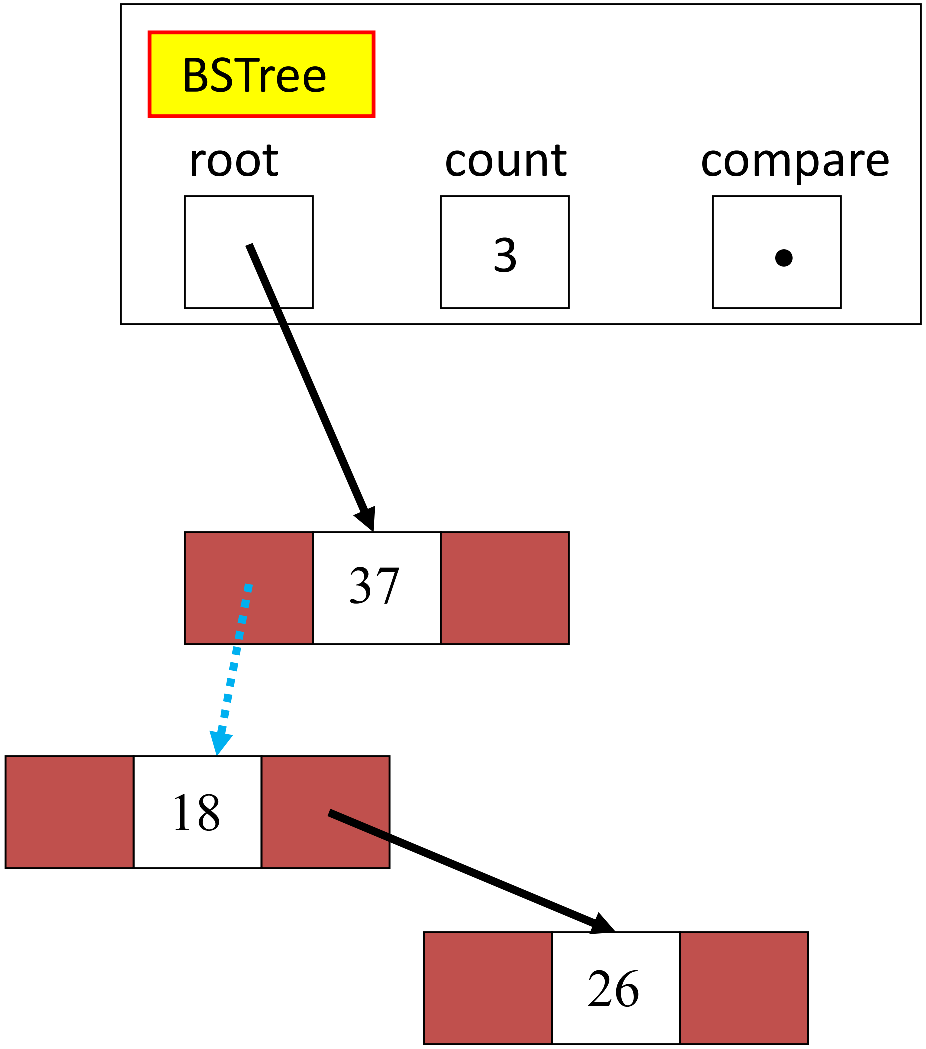
Search for a NODE on the BST



start at the root

If less – search left node
if greater – search right node
if equal – return location
else
return 'not found'

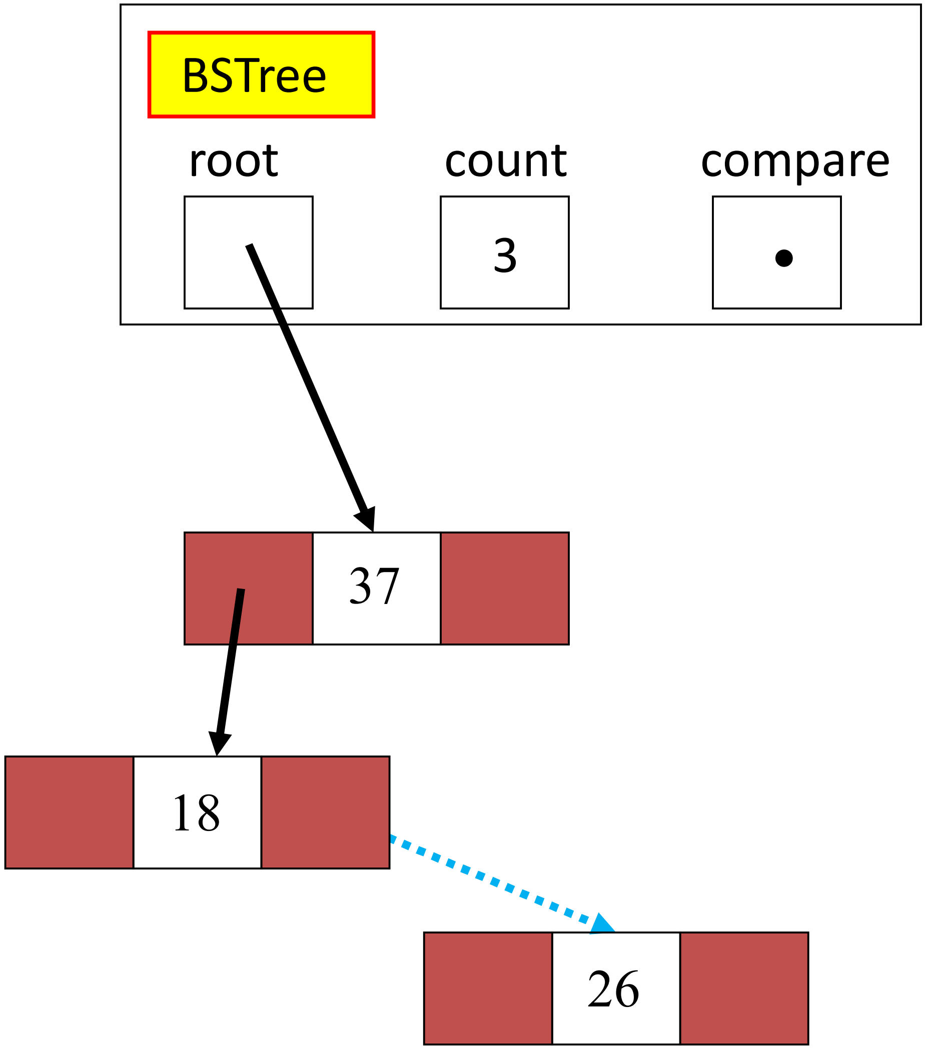
Search for a NODE on the BST



start at the root

If less – search left node
if greater – search right node
if equal – return location
else
return ‘not found’

Search for a NODE on the BST



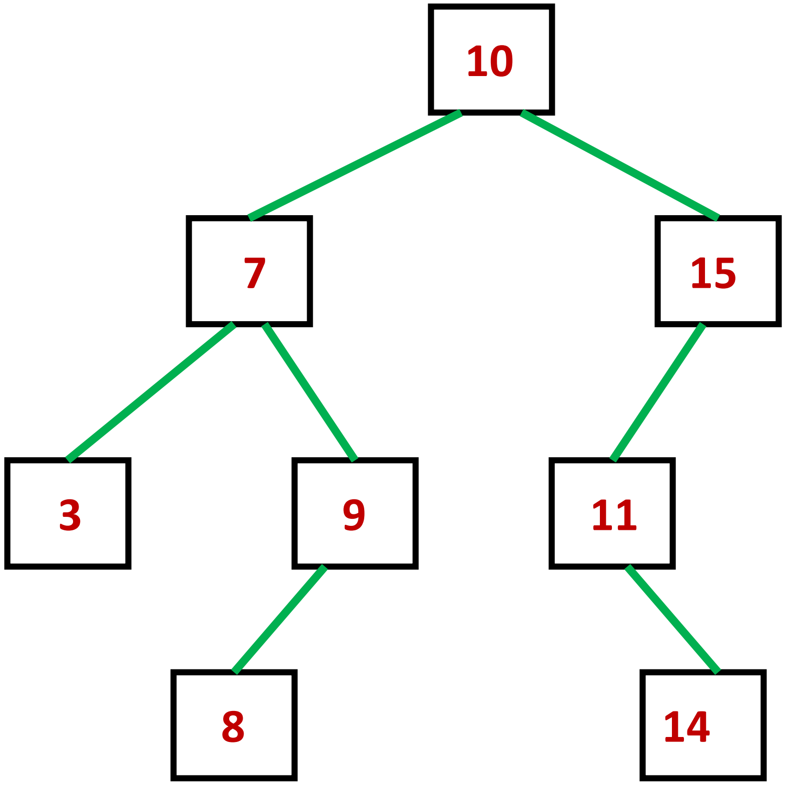
start at the root

If less – search left node
if greater – search right node
if equal – return location
else
return ‘not found’

Trees in C

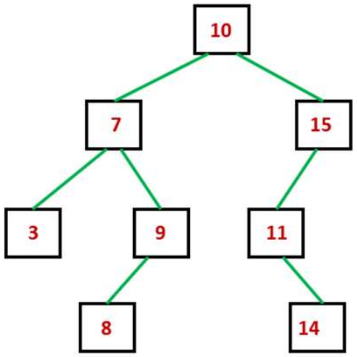
END OF PART 4

Delete Operation – a Recursive Algorithm



```
delete (recursively)
If ( root == NULL)  return root


If (data < root->data);
 delete root->left
else if  (data > root->data);
 delete root->right
else
  if (root->left is NULL);
    *temp = root->right
     free(root)
   else if (root->left is NULL);
      *temp = root->left
      free(root)

find the smallest leaf to the right from this point
switch values
 delete unused node
```

```
...
  printf("\nDelete 7\n");
  root = deleteNode(root, 7);
...
```



BSTDelete.c

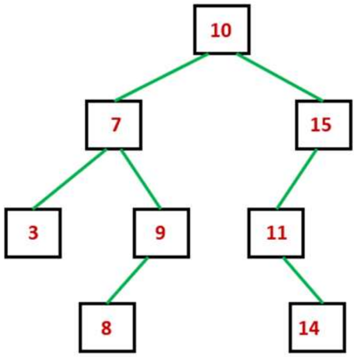
```
/* Given a binary search tree and a data, this function deletes the data
and returns the new root */
struct node* deleteNode(struct node* root, int data)
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
   {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(root);
           return temp;
        }

        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
     }
     return root;
}
```

Label	Address	Value
root	400 - 403	1010
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

```
...
  printf("\nDelete 7\n");
  root = deleteNode(root, 7);
 ...
```

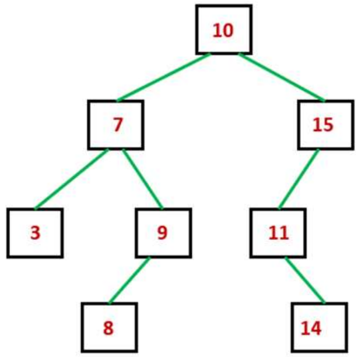
BSTDelete.c

```
/* Given a binary search tree and a data, this function deletes the data
and returns the new root */
struct node* deleteNode(struct node* root, int data)
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
   {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(root);
           return temp;
        }

        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
     }
     return root;
}
```



Label	Address	Value
root	400 - 403	1010
root	500 - 503	10100
data	504 -507	7
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

```
...
  printf("\nDelete 7\n");
  root = deleteNode(root, 7);
 ...
```

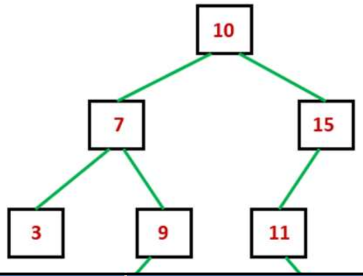


BSTDelete.c

```
/* Given a binary search tree and a data, this function deletes the data
and returns the new root */
struct node* deleteNode(struct node* root, int data)
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
   {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(root);
           return temp;
        }

        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
```

Label	Address	Value
root	400 - 403	1010
root	500 - 503	10100
data	504 -507	7
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

```
...
  printf("\nDelete 7\n");
  root = deleteNode(root, 7);
 ...
```

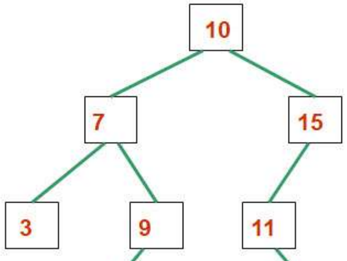


BSTDelete.c

```
/* Given a binary search tree and a data, this function deletes the data
and returns the new root */
struct node* deleteNode(struct node* root, int data)
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
   {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(root);
           return temp;
        }

        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
     }
     return root;
}
```

Label	Address	Value
root	400 - 403	1010
root	500 - 503	10100
data	504 -507	7
root	600 - 603	1230
data	604 - 607	7
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

```
...
 printf("\nDelete 7\n");
 root = deleteNode(root, 7);
  ...
```



BSTDelete.c

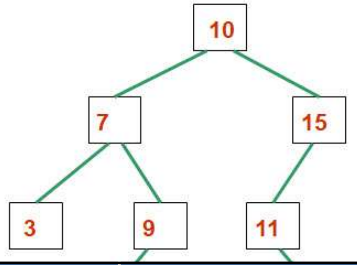
```
/* Given a binary search tree and a data, this function deletes the data
and returns the new root */
struct node* deleteNode(struct node* root, int data)
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
     {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(root);
           return temp;
        }

        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
     }
     return root;
}
```

Label	Address	Value
root	400 - 403	1010
root	500 - 503	10100
data	504 -507	7
root	600 - 603	1230
data	604 - 607	7
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

```
...
  printf("\nDelete 7\n");
  root = deleteNode(root, 7);
 ...
```

BSTDelete.c

```
/* Given a binary search tree and a data, this function deletes the data
and returns the new root */
struct node* deleteNode(struct node* root, int data)
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
   {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(root);
           return temp;
        }

        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
     }
     return root;
}
```



Label	Address	Value
root	400 - 403	1010
root	500 - 503	10100
data	504 -507	7
root	600 - 603	1230
data	604 - 607	7
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

```
/* Given a non-empty binary search tr        rn the node with minimum
data value found in that tree. Note t        entire tree does not
need to be searched. */
struct node * minValueNode(struct node* node)
{
            struct node* current = node;

            /* loop down to find the leftmost leaf */
            while (current && current->left != NULL)
                        current = current->left;

            return current;
}
```

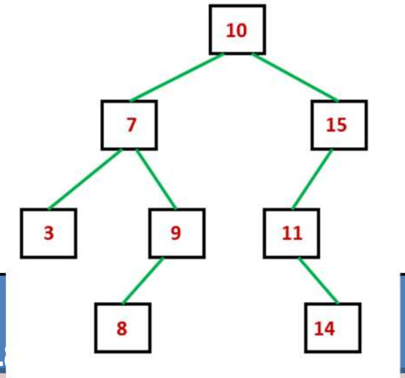
1420

1980

```
/
a
s
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
   {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(r
           retur
        }

        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
     }
     return root;
}
```

1980

1420



L		Value
root	400 - 403	1010
root	500 - 503	10100
data	504 -507	7
root	600 - 603	1230
data	604 - 607	7
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	7
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

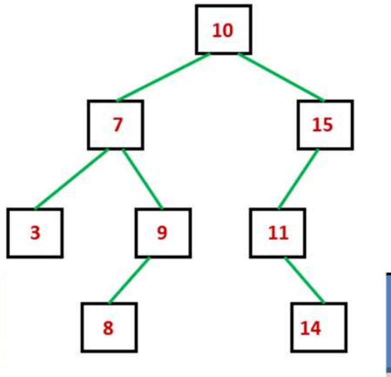
```
...
  printf("\nDelete 7\n");
  root = deleteNode(root, 7);
 ...
```

BSTDelete.c

```
/* Given a binary search tree and a data, this function deletes the data
and returns the new root */
struct node* deleteNode(struct node* root, int data)
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
   {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(root);
           return temp;
        }

        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
     }
     return root;
}
```

8

1980



L		Value
root	400 - 403	1010
root	500 - 503	10100
data	504 -507	7
root	600 - 603	1230
data	604 - 607	7
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	8
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

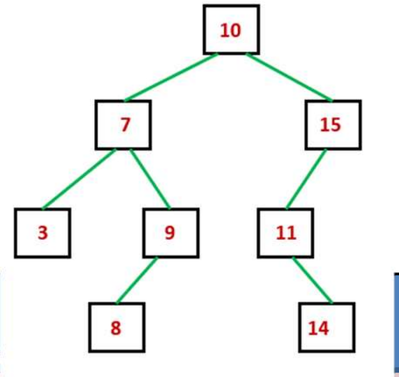
```
...
  printf("\nDelete 7\n");
  root = deleteNode(root, 7);
 ...
```

BSTDelete.c

```
/* Given a binary search tree and a data, this function deletes the data
and returns the new root */
struct node* deleteNode(struct node* root, int data)
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
   {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(root);
           return temp;
        }

        struct node* temp = minValu        t->ri    ;
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
     }
     return root;
}
```

1420

8

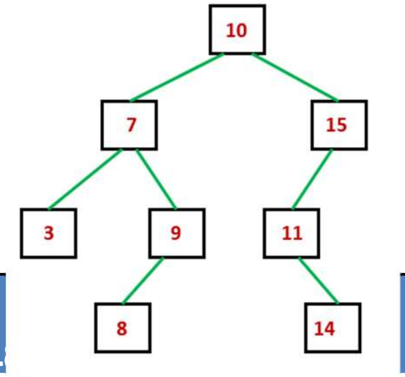


L		Value
root	400 - 403	1010
root	500 - 503	10100
data	504 -507	7
root	600 - 603	1230
data	604 - 607	7
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	8
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

```
...
  printf("\nDelete 7\n");
  root = deleteNode(root, 7);
 ...
```

BSTDelete.c

```
/* Given a binary search tree and a data, this function deletes the data
and returns the new root */
struct node* deleteNode(struct node* root, int data)
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
   {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(root);
           return temp;
        }

        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
     }
     return root;
}
```

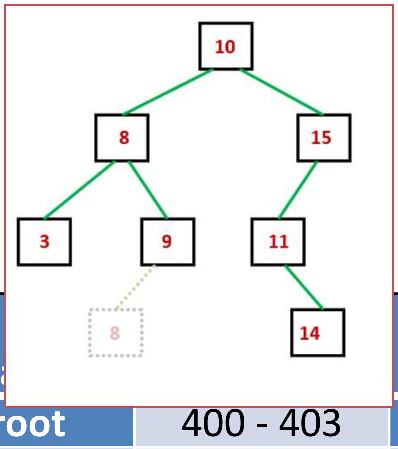


L		Value
root	400 - 403	1010
root	500 - 503	10100
data	504 -507	7
root	600 - 603	1230
data	604 - 607	7
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	8
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

```
...
  printf("\nDelete 7\n");
  root = deleteNode(root, 7);
 ...
```

BSTDelete.c

```
/* Given a binary search tree and a data, this function deletes the data
and returns the new root */
struct node* deleteNode(struct node* root, int data)
{
   if (root == NULL) return root;

   if (data < root->data)
       root->left = deleteNode(root->left, data);
   else if (data > root->data)
           root->right = deleteNode(root->right, data);
   else
   {
        if (root->left == NULL)
           struct node *temp = root->right;
           free(root);
           return temp;
        }
        else if (root->right == NULL)
        {
           struct node *temp = root->left;
           free(root);
           return temp;
        }

        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
     }
     return root;
}
```



La		Value
root	400 - 403	1010
	…	
	…	
	…	
	…	
data	1010 - 1013	10
left	1014 - 1017	1230
right	1018 - 1021	1750
data	1230 – 1233	8
left	1234 - 1237	1850
right	1238 – 1241	1420
data	1420 – 1423	9
left	1424 - 1427	1980
right	1428 – 1431	NULL
data	1750 – 1753	15
left	1754 - 1757	2170
right	1758 – 1761	NULL

Trees in C

END OF PART 6
END OF TREES in C