Lecture12: Basic Data Structure with Binary Tree
12/10/2012

Slides modified from Yin Lou, Cornell CS2022: Introduction to C

1

Administration

- Assignment #11 is on the webpage.

2

Outline

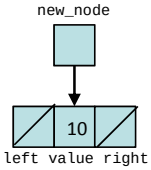
- Binary Tree
- Write large programs with multiple files

3

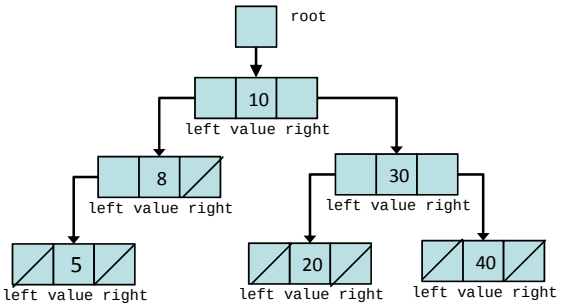
A Binary Tree

- A tree is a data structure that stores data, like linked list.
- Binary tree supports fast add-delete-search.
- Each node in the tree leads to two further nodes.

```
struct node {
   int val;
   struct node *left;
   struct node *right;
};
```



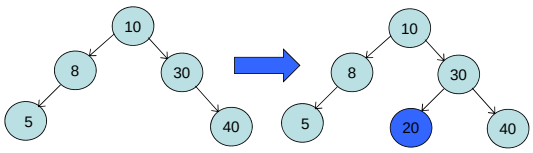
A Binary Tree: fast search (also add & delete)



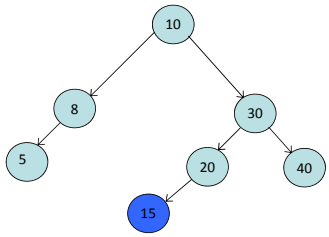
Binary Tree Data Structure

- A tree node includes:
  - The data
  - Pointers to two children nodes (left and right)
    - 2 pointers == binary
- Left node pointer points to a node with data less than the current node
  - All nodes to the left contain data less
- Right node pointer points to a node with data greater than the current node
    - All nodes to the right contain data greater
- A leaf node is a node with no children
- A root node is the top node

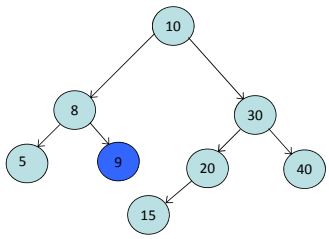
Add a node (20) - now add 15



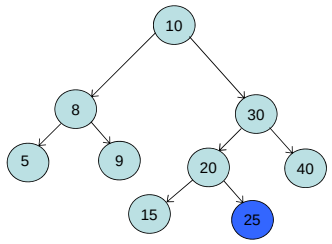
Add a node (15) – now add 9



Add a node (9) – now add 25



Add a node (25)



Binary Tree Implementation in C

```
struct node {
   int value;
   struct node *left;
   struct node *right;
};
```

```
struct node *root = NULL;
```

root 

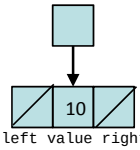
11

Create a node

```
struct node {
   int value;
   struct node *left;
   struct node *right;
};
```

```
new_node = malloc(sizeof(struct node));
```

new_node

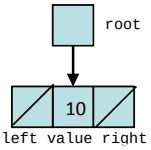



```
new_node->value = 10;
new_node->left = NULL;
new_node->right = NULL;
```

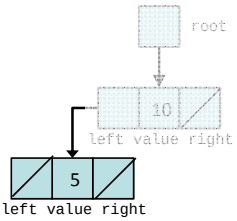
12

Insert a node (n=5)

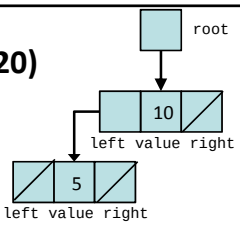
```
struct node *new_node;
new_node = malloc(sizeof(struct node));
new_node->value = n;
new_node->left = NULL;
new_node->right = NULL;
```



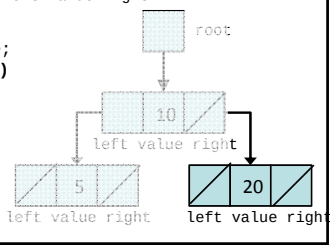

```
if (root->value > n)
   root->right = new_node;
```



Insert a node (n=20)

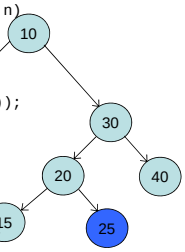



```
if (root->value > n)
   root->right = new_node;
else if (root->value < n)
   root->left = new_node;
```



Insert a node

```
struct node *add_node(struct node *tree, int n)
{
   if (tree == NULL)
   {
         struct node *new_node;
         new_node = malloc(sizeof(struct node));
         new_node->value = n;
         new_node->left = NULL;
         new_node->right = NULL;
         tree = new_node;
   }
   else if (tree->value > n)
         tree->left = add_node(tree->left, n);
   else if (tree->value == n)
         printf("Error: %d already exists\n", n);
   else
         tree->right = add_node(tree->right, n);
    return tree;
}
```



Tree operations (like Linked List)

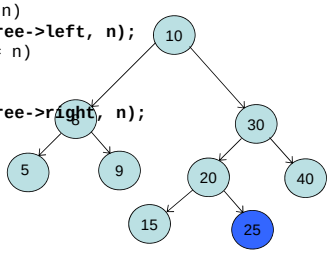
Beside `add_node()`, other possible list operations:

```
struct node *search_node(struct node *tree, int n);
void print_tree(struct node *tree);
struct node* delete_node(struct node *tree, int n);
int sum_tree(struct node *tree);
…
```

16

`struct node *search_node(struct node *tree, int n);`

```
if (tree == NULL)
  return NULL;
else if (tree->value > n)
  return search_node(tree->left, n);
else if (tree->value == n)
  return tree;
else
  return search_node(tree->right, n);
```



17

In-Class Exercise 12.1

Copy and paste the skeleton code from the exercise. Implement the `print_tree()` function that prints values from small to large.

```
void print_tree(struct node* tree);
```

18

In-Class Exercise 12.2

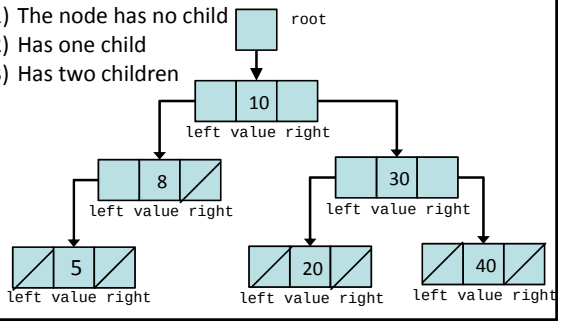
Now reorder two lines of code in your print_tree() function to print values from large to small.

```
void print_tree(struct node* tree);
```

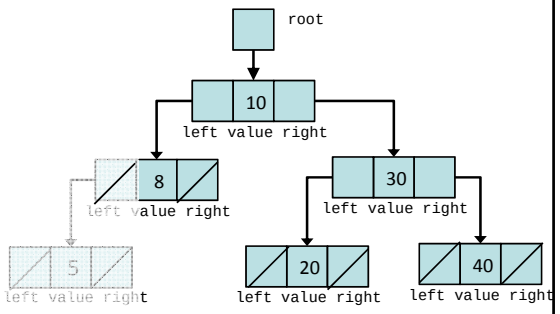
19

Delete a node

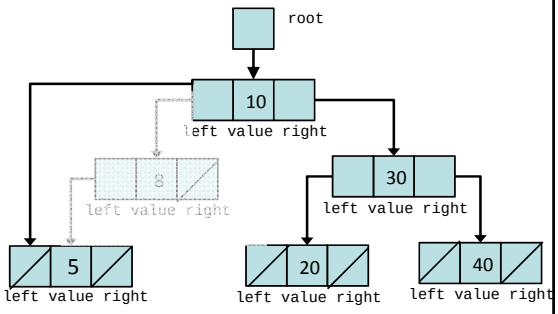
Find the deleted node
(1) The node has no child
(2) Has one child
(3) Has two children



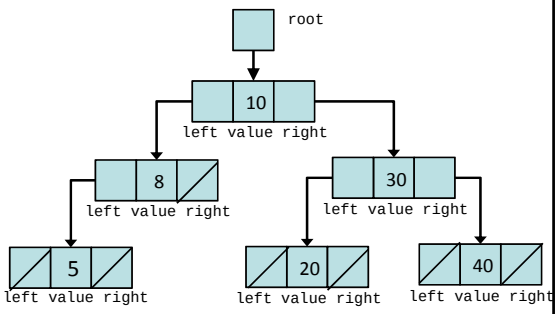
Delete a node (no child: 5)



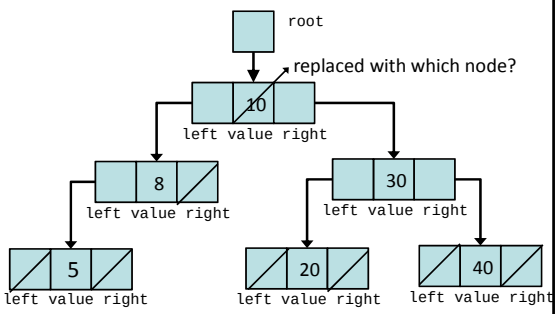
Delete a node (one child: 8)



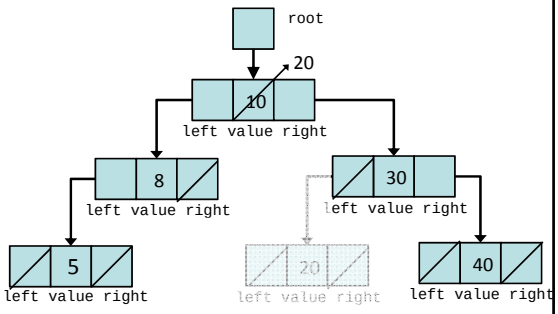
Delete a node (two children: 10)



Delete a node (two children: 10)



Delete a node (two children: 10)



struct node *delete_node(struct node *tree, int n)

```
/* Find the deleted node */
if (tree == NULL);
else if (tree->value > n)
   tree->left = delete_node(tree->left, n);
else if (tree->value < n)
   tree->right = delete_node(tree->right, n);
else if (tree->value == n)
{
   /* three cases:
    * (1) the node has no child
    * (2) the node has one child
    * (3) the node has two children
    */
```

struct node *delete_node(struct node *tree, int n)

```
  /* three cases:
   * (1) the node has no child
   * (2) the node has one child
   * (3) the node has two children
   */
  struct node *temp;
  if (tree->left == NULL) // one right child or none
  {
      temp = tree->right;
      free(tree);
      tree = temp;
  }
  else if (temp->right == NULL) // one left child or none
  {
      temp = tree->left;
      free(tree);
      tree = temp;
  }
```

struct node *delete_node(struct node *tree, int n)

```
  /*
   * (3) the node has two children
   */

  else // two children
  {
  /* find the smallest_node whose value > n
   * replace this smallest_node's value with tree's value
   * delete this smallest_node
   */
      temp = tree->right;
      while (temp->left != NULL)
            temp = temp->left;
      tree->value = temp->value;
      tree->right = delete_node(tree->right, tree->value);
  }
}
return tree;
```

In-Class Exercise 12.3

Implement the following function that returns the depth of a tree (the maximum number of nodes between the root and any leaf node in the tree). Note that this number also counts the root and the leaf node.

```
int get_depth(struct node* tree)
```

29

Write a large multi-file program

- Small C programs can be put in one file.
- Large C programs ought to be split into multiple files. One file implements one related group of functions.
- Two types of files
  - Header files “xxx.h”
  - Source files “xxx.c”

30

Header File

- Place common material to be **shared** in a header file.
  - Shared function declarations (not function definitions)
  - Shared struct, enumeration, union, etc.
  - Shared MACROs (#define …)
  - External variables
- Can be *included* as needed.

Example: "mymath.h"

```
int fact(int n);
int power(int power, int exp);
extern int global_var1;
extern int global_var2;
```

31

Source file: contain function definitions

power.c

```
#include "mymath.h"

int global_var1;
int power(int base, int exp)
{
   int result = 1;
   int i;
   for (i = 1; i <= exp; ++i)
   {
        result *= base;
   }
   return result;
}
```

fact.c

```
#include "mymath.h"

int global_var2;
int fact(int n)
{
   if (n == 0)
   {
        return 1;
   }
   else
   {
        return n * fact(n - 1);
   }
}
```

32

Example

main.c

```
#include <stdio.h>
#include "mymath.h"

void main()
{
  printf("%d\n", power(5, 3));
  printf("%d\n", fact(5));
}
```

33

In-Class Exercise 12.4

Create a multi-file program:
- Move the tree functions (`add_node`, `delete_node`, etc.) into a file "tree.c"
- Create a new header file "tree.h".
- The "main.c" file should have the `main()` function.

34