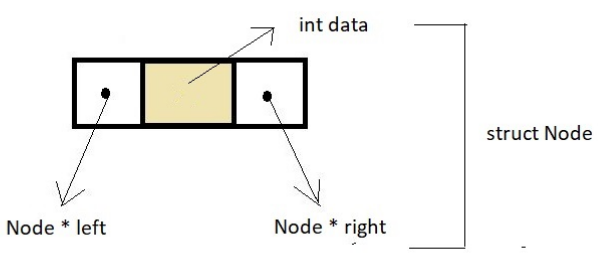
Linked Representation Of Binary Tree in C

Show Course Contents

Overview Q&A Downloads Announcements

Linked Representation Of Binary Tree in C

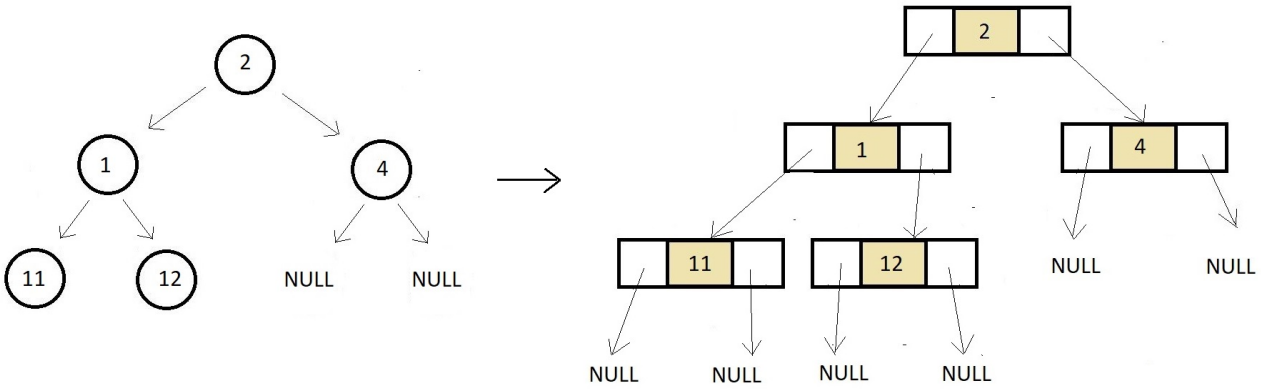
In the last lecture, we saw different representations of a binary tree. We saw its array representation. We saw its linked representation. In contrast to the linked representation of binary trees, we found its array representation to be futile at the end because of its size constraint and the complexity of its implementation.
And since the array representation is not that tough to implement I believe, I'll continue with the linked representation, and a quick revision on that should work.
1. We use the concept of doubly-linked lists to represent a node.



2. The left pointer of this node points to the left child and the right pointer points to

the right child, and if there is no left or right child, we represent that using a NULL.

See how a tree looks in real life, and how efficiently the linked representation of a tree helps us visualize the same.



We also saw the declaration of the structure Node in C. Today we'll see all that in detail. We'll see how these nodes get linked, and the creation of a binary tree via that. I have attached the source code below. Follow it as we proceed.

Understanding the code snippet below:

1. The first thing would be to create the struct Node we discussed yesterday. So, create a struct Node. Inside the structure, we have three embedded elements, first is an integer variable *data* to store the data of the node, second is a struct Node pointer called *left* to store the address of the left child node, and third is again a struct Node pointer called *right* to store the address of the right child node.

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

Code Snippet 1: Creating the struct Node

2. Our next job would be to create the root node. In main, create struct Node pointer *p*, and assign to it the memory in heap using malloc. Don't forget to include the

header file <malloc.h> or <stdlib.h> And then using the pointer operator →, assign some data to its *data* element and NULL to both the left and the right element of the struct Node *p*.

3. And now we can proceed to creating the further nodes. We have different ways to do that. First one is to copy the whole thing we did for the root node twice for both the children and name them p1 and p2. This will create three separate nodes p, p1 and p2. Then just link them together by changing the left element of *p* from NULL to the left node's pointer p1 and the right element of *p* from NULL to the right node's pointer p2.

4. But this is somewhere redundant and not considered a good practice as we are copying the whole thing again and again. So, we would create a dedicated function for creating a node.

Creating the createNode function:

5. Create a struct Node* function *createNode,* and pass the data for the node as the only parameter. Create a struct Node pointer n. Reserve a memory in heap for n using malloc. Basically, do the same thing we did above. Point the left and the right element of the struct n to NULL, and fill the data in the data element. And return the node pointer n you created. This would simply create a node, and you can now very easily link them as per your wish via main to the other nodes.

```
struct node* createNode(int data){
    struct node *n; // creating a node pointer
    n = (struct node *) malloc(sizeof(struct node)); // Allocating m
    n->data = data; // Setting the data
    n->left = NULL; // Setting the left and right children to NULL
    n->right = NULL; // Setting the left and right children to NULL
    return n; // Finally returning the created node
}
```

Code Snippet 2: Creating the function createNode

Here is the whole source code:

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

```c
#include<malloc.h>

struct node{
    int data;
    struct node* left;
    struct node* right;
};

struct node* createNode(int data){
    struct node *n; // creating a node pointer
    n = (struct node *) malloc(sizeof(struct node)); // Allocating m
    n->data = data; // Setting the data
    n->left = NULL; // Setting the left and right children to NULL
    n->right = NULL; // Setting the left and right children to NULL
    return n; // Finally returning the created node
}

int main(){
    /*
    // Constructing the root node
    struct node *p;
    p = (struct node *) malloc(sizeof(struct node));
    p->data = 2;
    p->left = NULL;
    p->right = NULL;

    // Constructing the second node
    struct node *p1;
    p1 = (struct node *) malloc(sizeof(struct node));
    p->data = 1;
    p1->left = NULL;
    p1->right = NULL;

    // Constructing the third node
    struct node *p2;
```

```
    p2 = (struct node *) malloc(sizeof(struct node));
    p->data = 4;
    p2->left = NULL;
    p2->right = NULL;
    */

    // Constructing the root node - Using Function (Recommended)
    struct node *p = createNode(2);
    struct node *p1 = createNode(1);
    struct node *p2 = createNode(4);

    // Linking the root node with left and right children
    p->left = p1;
    p->right = p2;
    return 0;
}
```

Code Snippet 3: Implementing binary trees in C

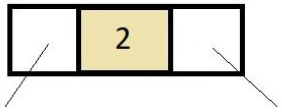
Now we can see if our program doesn't revert any error while we try linking 3 nodes p, p1 and p2.

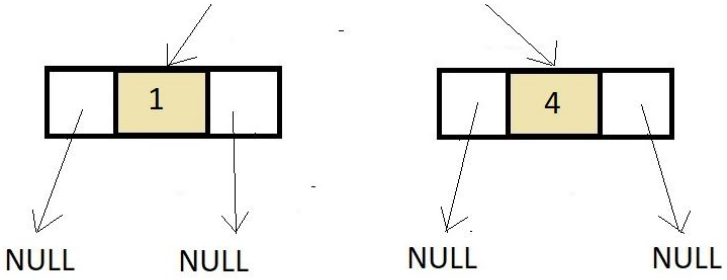
```
    struct node *p = createNode(2);
    struct node *p1 = createNode(1);
    struct node *p2 = createNode(4);

    p->left = p1;
    p->right = p2;
```

Code Snippet 4: Using createNode to create nodes.

No, it didn't show any error, and this is how we created three nodes and linked two of them to the root node. You can imagine how it would have looked.





We can create even more of these nodes, and link them accordingly. We will start further with the operations on these very soon. Stay connected.
I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial where we'll learn our first operation on a binary tree, its traversal. Till then keep coding.

Previous Next

CodeWithHarry