



6.S096 | January IAP 2013 | Undergraduate

Introduction To C And C++

Menu

More Info

Feedback

Lectures and Assignments

Data Structures, Debugging

Topics: Using structs, unions, typedef, and enums, and how to debug with Valgrind and GDB.

Lecture Notes

[Lecture 4: Data Structures, Debugging_\(PDF\)](#)

Lab Exercises

The primary goal of this lab period is to introduce debugging tools, and use of unions/structs.

Exercise 1

Download and install [Valgrind](#) on your system, if it's not already. To test if you have Valgrind, run `valgrind --version`. It should print the version of Valgrind that is installed.

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

void fn()
{
    int* x = malloc(10 * sizeof(int));
    printf("%d", *x);
    x[10] = 0;
}

int main()
{
    fn();
    return 0;
}
```

There are 3 sources of memory errors in this code. Run valgrind to determine what they are (although I suspect you can probably tell from the code anyways).

Exercise 2

Use a union to print the individual bytes of an `int`. (Hint: Recall the size of `ints` and other data types.)

Exercise 3

Determine how much memory is required for each of the `structs` below. How much of that memory is padding between the members?

```
struct X
{
    short s;
    int i;
    char c;
};

struct Y
{
    int i;
    char c;
    short s;
};

struct Z
{
    int i;
    short s;
    char c;
};
```

Assignment 4

Today's assignment combines the material from the past few lectures. Your job is to fill in the skeleton code we provide. I have commented the code with what each section should do.

[Assignment 4 files \(ZIP\)](#). (This ZIP file contains: 2 .c files and 1 .h file.)

You can learn more about binary search trees and find pseudo-code on [the binary search tree page on Wikipedia](#).

Your job is to implement a binary search tree, a data structure of connected nodes with a tree shape. Each node has a node identifier (a number), data (payload), and 2 children (left and right). The children are other nodes referenced with a pointer, with the constraint that the left node's ID is less than the parent node's ID, and the right node's ID is larger than the parent node ID. No two nodes will have the same identifier. A node can have less than two children; in that case, one or more of its child pointers can be `NULL`.

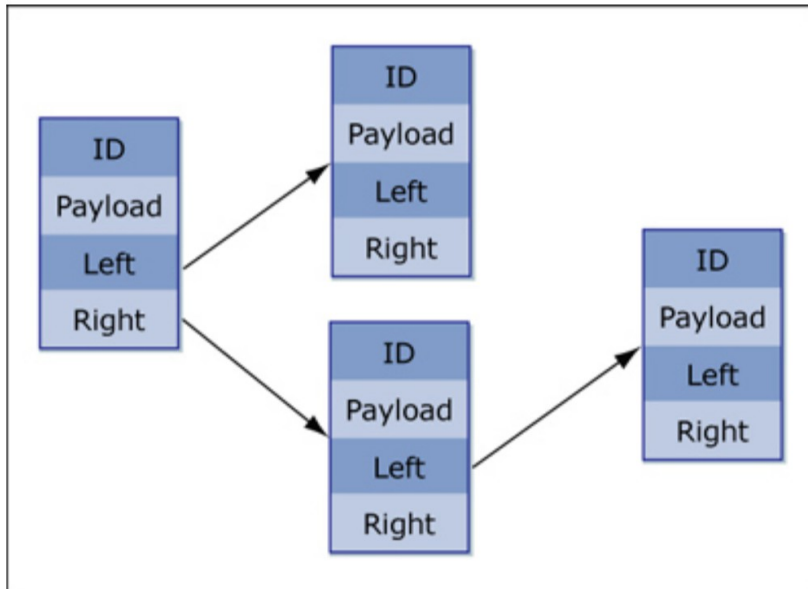


Image by MIT OpenCourseWare.

`user.c` contains the `main()` function. We will replace this function with one for grading. You should use your `main()` function to test that your functions to insert into and search the binary tree work correctly.

This is a C/C++ course, not an algorithms course, but if you want a challenge, try implementing node deletion as well!

Your job is to complete the data structure and function declarations in `bintree.h`, then complete the implementation of your functions in `bintree.c`. If you want to define additional functions to simplify your program, that's fine. You cannot change the return types or argument types of the included functions, though. Even though we don't require the deletion function, make sure to free all memory you allocate!

Make sure your program compiles without warning, runs, and *definitely* use `valgrind` to ensure you have no memory leaks.

```
$ gcc -Wall -std=c99 user.c bintree.c -o bintree
$ ./bintree
<your test output>
```

Solutions

Solutions are not available for this assignment.




Over 2,500 courses & materials

Freely sharing knowledge with learners and educators around the world. [Learn more](#)

[Accessibility](#)

[Creative Commons License](#)

[Terms and Conditions](#)

Proud member of:  **Open Education
GLOBAL**



© 2001–2025 Massachusetts Institute of Technology