COMP20003 Workshop Week 4

1	Some simple & useful C tools
2	Binary Trees & BST, Q 3.1
3	AVL & Rotations, Q 3.2
4	Lab 4.1 BST lookup, insertion with: - valgrind - gdb - make BST rotation (time permitted)

Using argc and argv (parameters of main())

```
int main(int argc, char *argv[])…
```

More than one input or output streams? Use text files.

Programs can read or write to text files. Each text file should have a filename (for example mytext).

	input	output

open file	`FILE *f;`
	`f= fopen("mytext", "r");`
	`assert(f);`
		`FILE *f;`
		`f= fopen("mytext", "w");`
		`assert(f);`

read/write to/from string s and int n	`fscanf(f, "%s %d",s, &n);`	`fprintf(f, "%s %d", s, n);`
close file after use	`fclose(f);`	`fclose(f);`

scanf/fscanf a string that contains spaces or odd characters

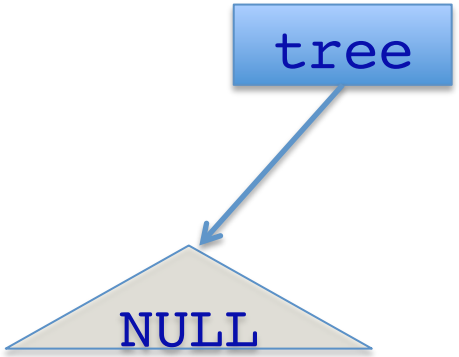
Problems with `scanf("%s %d", name, &age);`

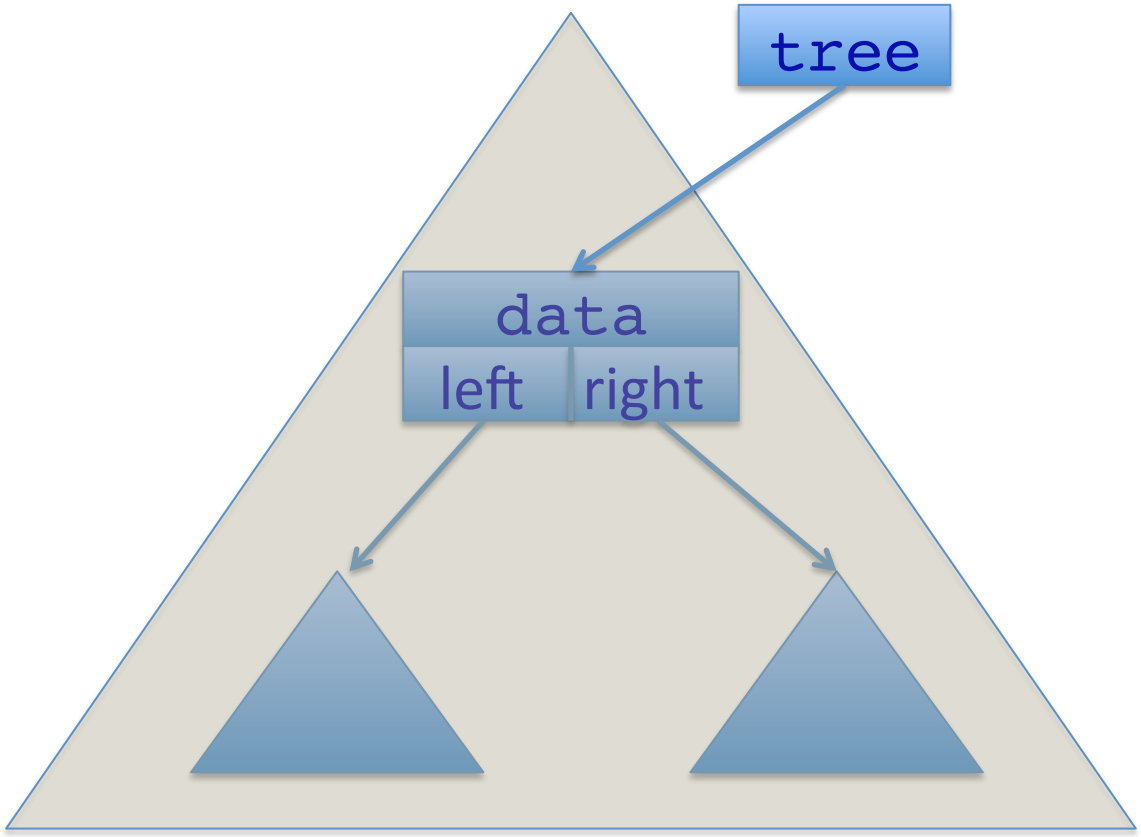
supposing input is **`Donald Trump, 78`**

Example

Binary trees and BST

Binary tree ≡ an *empty tree* (`NULL`), or
a *root node* (with some `data`) that is
connected to a *left sub-tree* and a *right sub-tree*





Declaring trees: examples

Model 1: (employed in the LMS workshop sheets)

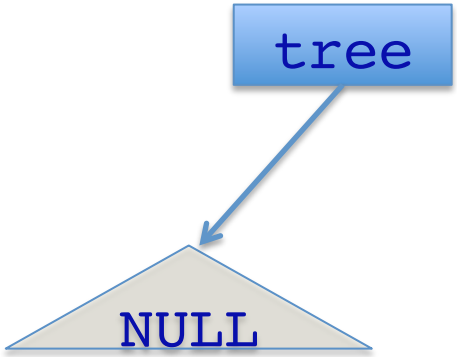
```
struct bst {
   int data;                  //  any data
   struct bst *left;          //  left child
   struct bst *right;         //  right child
};
struct node *mytree= NULL;
```

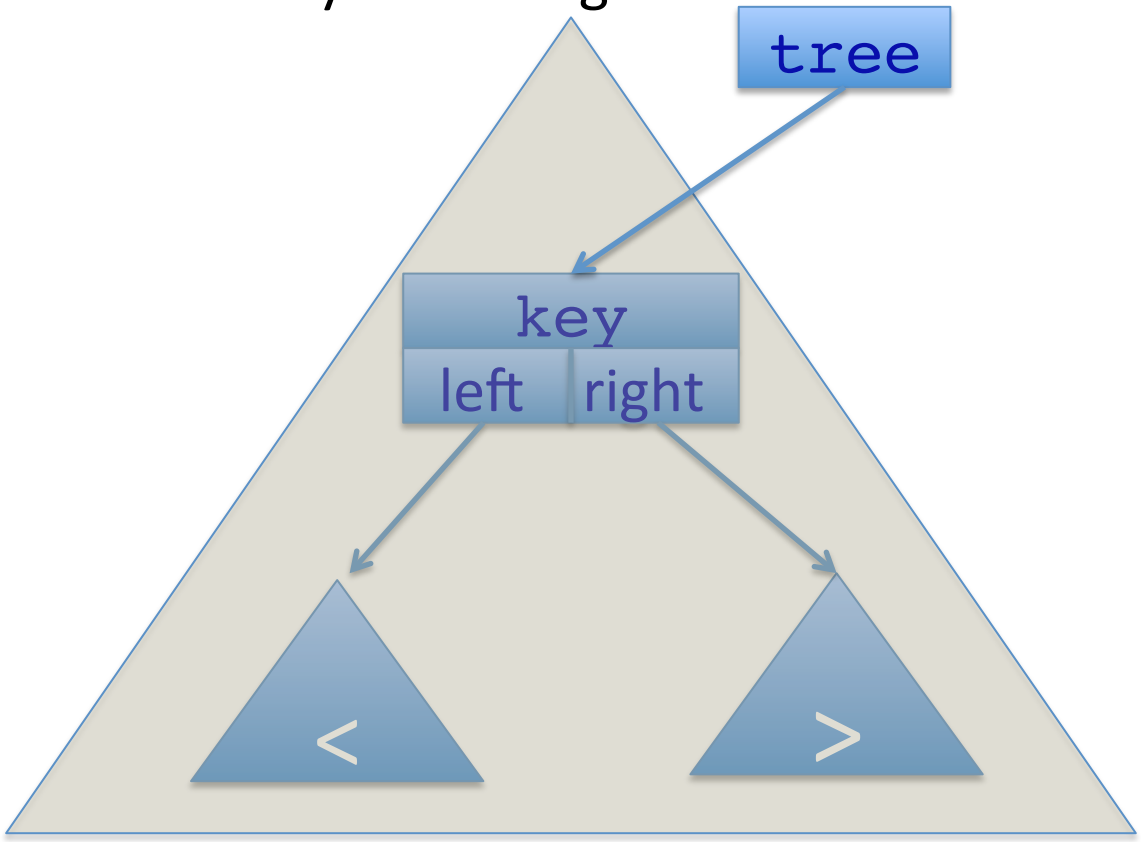
Model 2:

```
typedef struct node_t *tree_t;
struct node_t {
   int key;                //  any data
   tree_t left;            //  left child
   tree_t right;           //  right child
};
tree_t mytree= NULL;
```

BST is a binary tree…

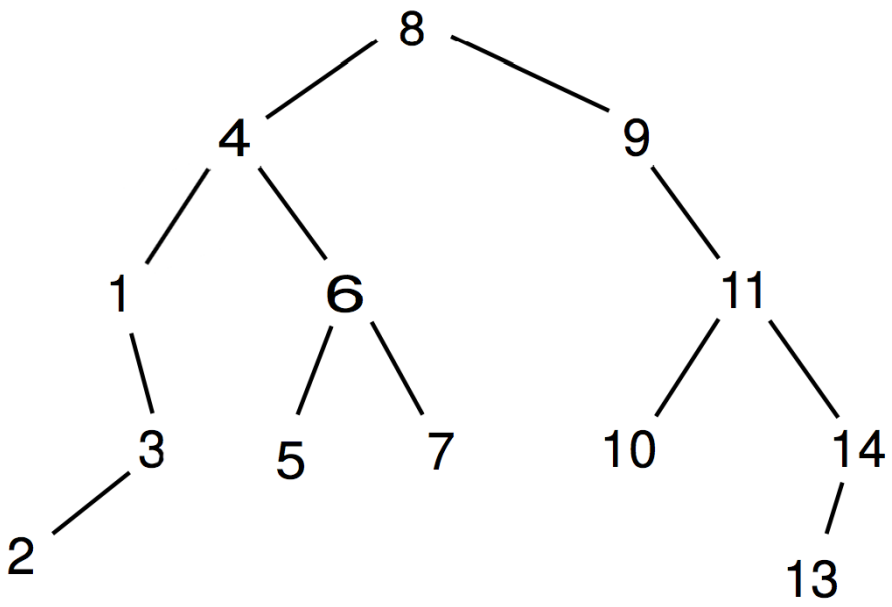
BST ≡ a *binary tree*, where
position of a node depends on the value of key, and
for any sub-tree the key of its *root :*
is *larger* than keys on its left tree, and
is *smaller* than keys on its right tree





Binary Trees: Traversal

in-order, pre-order, post-order



Q3.1: building BST

8 4 9 11 6 7 1 5 3 14 10 13 2

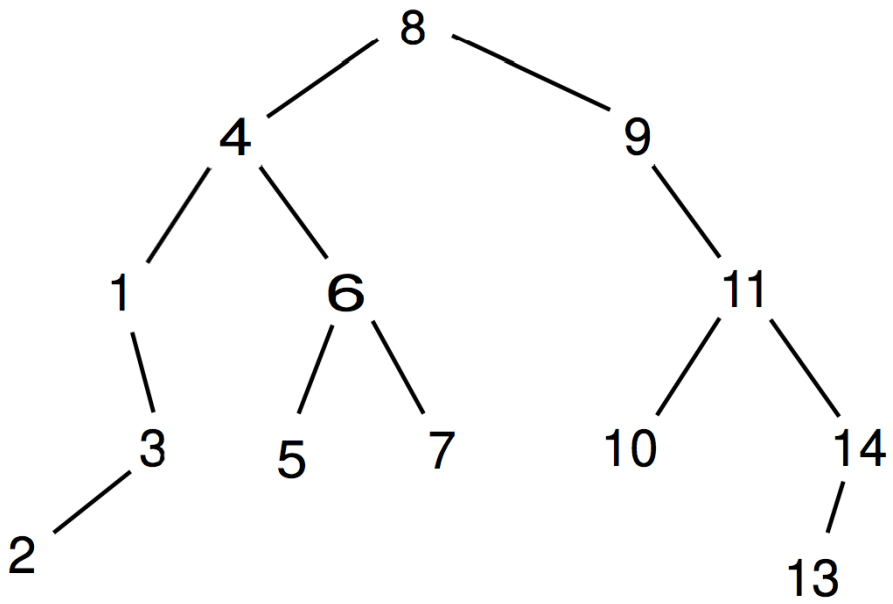
(ii) Draw a new tree after inserting each number into a BST.

Q3.1: building BST

Given the following input:

```
8  4  9  11  6  7  1  5  3  14  10  13  2
```

(i) Write the insertion function used to get the binary search tree shown below.



Q4.1: building BST

(i) Write the insertion function for BST, supposing:

```
struct bst {
   int key;
   struct bst *left;
   struct bst *right;
};
```

Consider using the prototype:

```
struct bst *bst_insert(struct bst *t, int key);
```

Q 4.2

In some circumstances, it might be useful to have a doubly-linked tree, i.e. each node has a parent pointer (root pointer is initialized to null), as well as two child pointers. Assume each node has a counter and a depth variable. After each insertion some counters get updated using the following assignment

```
node.counter = node.left.depth - node.right.depth;
```

Show how can you use the doubly-linked tree to keep the counters updated after a node is inserted.

Q 4.2: data structure

```
typedef struct node_t *tree_t;
struct node_t {
   key_t key;
   int depth;
   int counter;
   tree_t parent;
   tree_t left;
   tree_t right;
};
```

depth = ?

Why depth and counter? Remember AVL tree?

Q 4.2

```
node.counter = node.left.depth –
node.right.depth;
```

Show how can you use the doubly-linked tree to keep the counters updated after a node is inserted.

AVL tree and rotations

AVL tree = ?

Rotations, single & double rotations.

Programming Task today

Method 1: Do programming directly on Notebook

Method 2: download `github.com/anhvir/c203.` 2 versions:

Version 2A: the whole skeleton in Notebook is copied to a single file named `all.c`

Version 2B: the above file is extracted into 6 files:

for `llqueue.h`, `llqueue.c`, `bst.h, bst.c, main.c`, and `bst_data.c`. and a Makefile is added.

You'll learn more if you use version 2B. And you should:

1. `ssh bob@dimefox.eng.unimelb.edu.au`
2. use `gdb` and `valgrind` and try to debug;

Notes:

see `LMS.Resources` for `gdb` and `valgrind` guides.

Assignment 1: released today, due 8AM Mon 03/SEP