C++, POINTERS, AND DATA STRUCTURES

BME 121 2016

Jeff Luo

Topics

- C++ Basics
- C++ Pointers
- C++ Methods
- The Stack and Queue Data Structures
- The Tree Data Structure
- Binary Tree
- WA6

A little history

- C# built upon the concepts of C++

- “new” things in C#:
  - Garbage collection
  - Properties
  - Predefined / Implemented LinkedList, ArrayList, Collections…
  - Foreach loop
  - Interfaces

C++ Core Concept: Pointers and References

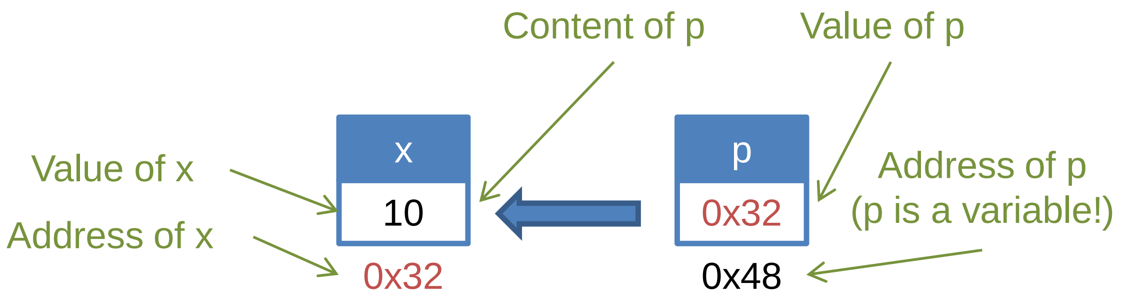
- A pointer is a special variable that only stores memory addresses
- It is given a Type, so that it only points to memory addresses of variables or objects of that Type
  - int x = 10;
  - int * p = &x; (Read: declare p as an pointer to an int, and assign it the address of variable x);
- A Reference in C++ means the memory address
  - &x is the reference of variable x.
  - We stick to the terminology of calling it the address of x instead

C++ Core Concept: Pointers and References

- A few additional terminologies:

int x = 10;

int * p = &x;



- The Value of a pointer is the memory address stored in that pointer
- Eg: cout << p << endl; // shows the memory address of x

- The Content of a pointer is the content stored in the memory address pointed to by that pointer
- EG: cout << *p << endl; // shows the content stored in x, which is 10

- Note that every variable and object in C++ has a memory address. This is a fundamental property! Thus, every pointer also has its own memory address.

Example 1

```
#include <iostream>
using namespace std;
int main ()
{
    int x, y;    // two variables
    int * ptr;   // 1 int pointer

    ptr = &x;    // store the address of x as the value of pointer
    *ptr = 10;   // assign content of pointer to 10 (changes x)
    ptr = &y;    // store the address of y as the value of pointer
    *ptr = 20;   // assign content of pointer to 20 (changes y)
    cout << "x is " << x << endl;        // shows 10
    cout << "y is " << y << endl;        // shows 20
    return 0;
}
```

Example 2

```
#include <iostream>
using namespace std;
int main ()
{
    int x, y;    // two variables
    int * p1, * p2;  // 2 int pointers

    p1 = &x;     // store address of x as value of p1
    p2 = &y;     // store address of y as value of p2
    *p1 = 10;    // assign content of p1 to 10 (changes x to 10)
    *p2 = *p1;   // assign content of p2 the content of p1 (copy content: y is now 10)
    p1 = p2;     // assign value of p1 to value of p2 (copy memory address: ie p1 now points to y)
    *p1 = 20;    // assign content of p1 to 20 (changes y to 20)

    cout << "x is " << x << endl;    // shows 10
    cout << "y is " << y << endl;    // shows 20
    return 0;
}
```

Inspecting the Value & Content of a Pointer

```
#include <iostream>
using namespace std;

int main ()
{
        int x = 10;
        int * p = &x;

        cout << "value of x:   "             << x  << endl;
        cout << "address of x: "             << &x << endl;
        cout << "value of p:   "             << p  << endl;
        cout << "address of p: "             << &p << endl;
        cout << "content of p: "             << *p << endl;
}
```

```
value of   x: 10
address of x: 0x28ff1c
value of   p: 0x28ff1c
address of p: 0x28ff18
content of p: 10
```

- Use these kinds of cout statements to help debug your program

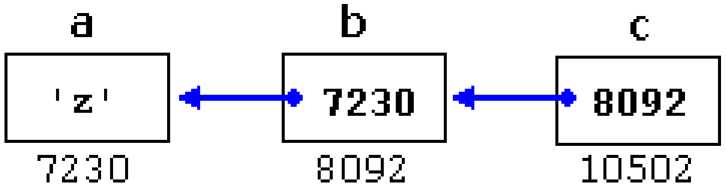
Pointers to Pointers

- The language allows pointers to pointers as well
- Any kind of pointer always store a memory address

```
char a;
char * b;
char ** c;
a = 'z';
b = &a;
c = &b;
```

Generally called a pointer (or specifically single pointer)

Generally called a double pointer



- Protip: Always draw diagrams like the one above to help you see things. Assign random integers as memory addresses, but stick to the ones you choose. If it is an array, assign a starting integer memory address, and every sequential element gets a memory address of 1 + the address of the previous element.

Pointers to Pointers to Pointers…

- These kinds of pointers always point to other pointers of 1 less star (*):
  - int ** p must point to some int * x
  - int *** p must point to some int ** x

- If it is a triple pointer, then ***p will retrieve the content of the variable or object. syntax same as (*(*(*p)))
  - Read: content of content of content of p
- If it is a quadruple pointer, then ****p will retrieve the content
- Etc…

Inspecting a Double Pointer

```
// my first double pointer
int main ()
{
    int x = 10;
    int * p1 = &x;
// p2 stores the address of an int pointer (p1)
    int ** p2 = &p1;

    cout << "value of x: " << x << endl;
    cout << "address of x: " << &x << endl;
    cout << "value of p1: " << p1 << endl;
    cout << "address of p1: " << &p1 << endl;
    cout << "content of p1: " << *p1 << endl;
    cout << "value of p2: " << p2 << endl;
    cout << "address of p2: " << &p2 << endl;
    cout << "content of p2: " << *p2 << endl;
    cout << "content of content of p2: " << **p2 << endl;
}
```

```
value of x: 10
address of x: 0x28ff1c

value of p1: 0x28ff1c
address of p1: 0x28ff18
content of p1: 10

value of p2: 0x28ff18
address of p2: 0x28ff14
content of p2: 0x28ff1c
content of content of p2: 10
```

Pointers & References used in Function Parameters

```
void fx(int * p) // expects an int pointer as input
{
    *p = 8;      // will change the content of whatever was pointed to by the input pointer to 8
}
// called by fx(&x) for some int x, or by passing in a pointer to an int

void fx2(int & p)   // expects an integer as input
{
    p = 8;       // will also change the input int to 8 [from calling code]
}
// called by fx2(x) for some int x
```

- It is preferred to use the top version of syntax for Pass by Reference, so that in the body of the function it is clear that we are modifying the contents of a pointer (clearly understood that modifications have some effect to a variable or object declared outside of the function)

Functions returning Pointers

- Two different ways:

```
int * myFunc()
{
    int x = 20;
    return &x; // returns address of x
}
int * myFunc2()
{
    int y = 30;
    int * ptr = & y;
    return ptr;   // returns value stored in ptr
}

int main ()
{
    int * p = myFunc();      cout << *p << endl;  // shows 20
    int * p2 = myFunc2();    cout << *p2 << endl; // shows 30
}
```

Classes, Objects & Pointers

- Assume we have the following:

```
MyClass myObject;        // created in stack memory (more on this @ Lab)
MyClass * myPtr = & myObject;
```

- There are three ways to invoke a member from an object

```
cout << myObject.getString() << endl;
cout << (* myPtr).getString() << endl;
cout << myPtr->getString() << endl;
```

- We don't normally use the middle line syntax since the language conveniently provides us with the `->` operator
- This works as well for public fields:

```
cout << myPtr->someStringField << endl;
```

CONFUSING C++ TYPES?

Use http://cdecl.org !

Node - The Core of All Advanced Data Structures

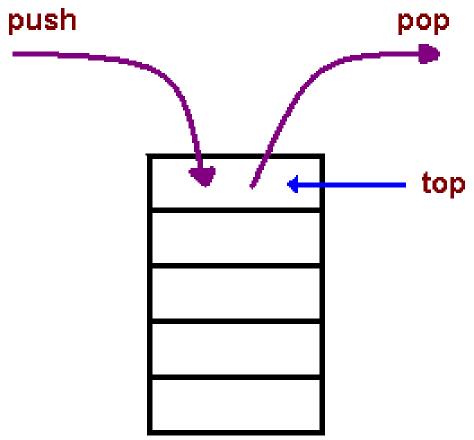
- The Node is the most basic element of a data structure, it is like the atoms of physical materials
- Each node has 1 or more pointers to other nodes.
- The connectivity between a group of nodes form data structures (like compound molecules).
- C++ gives you precise control on pointers, which makes it a more “deep” level language and a theoretically better platform for studying the details of data structures

Stack Data Structure

- A stack is a linear container of objects which are inserted according to a last-in-first-out principle (LIFO).
- EG: stack of books, pancakes

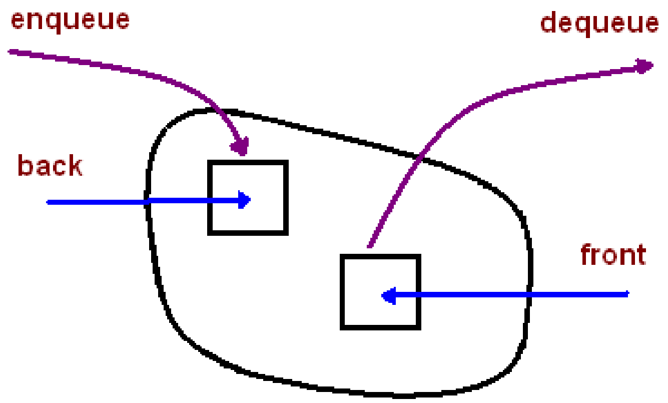
- Only 2 ways of getting an element into or out of a stack:
  - Push (insert) an object to the top of the stack
  - Pop (removed) an object from the top of the stack

- Implemented as an array or list behind the scenes



Queue Data Structure

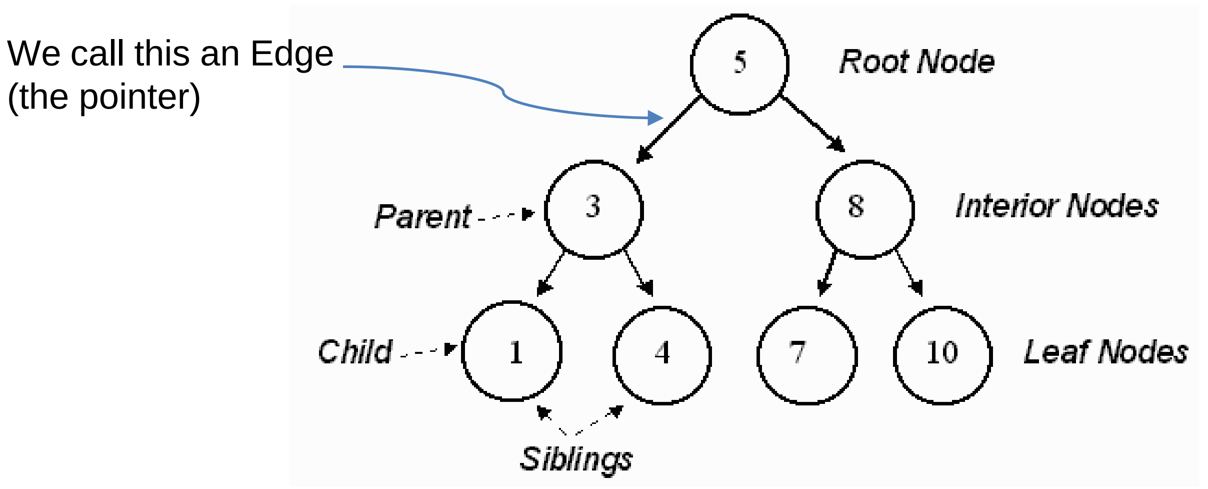
- A queue is a linear container of objects which are inserted according to a first-in-first-out principle (FIFO).
- EG: a line up of students at Starbucks
- Only 2 ways of getting an element into or out of a queue:
  - Enqueue (insert) an object to the back of the queue
  - Dequeue (remove) an object from the front of the queue



- Also implemented as an array or list behind the scenes

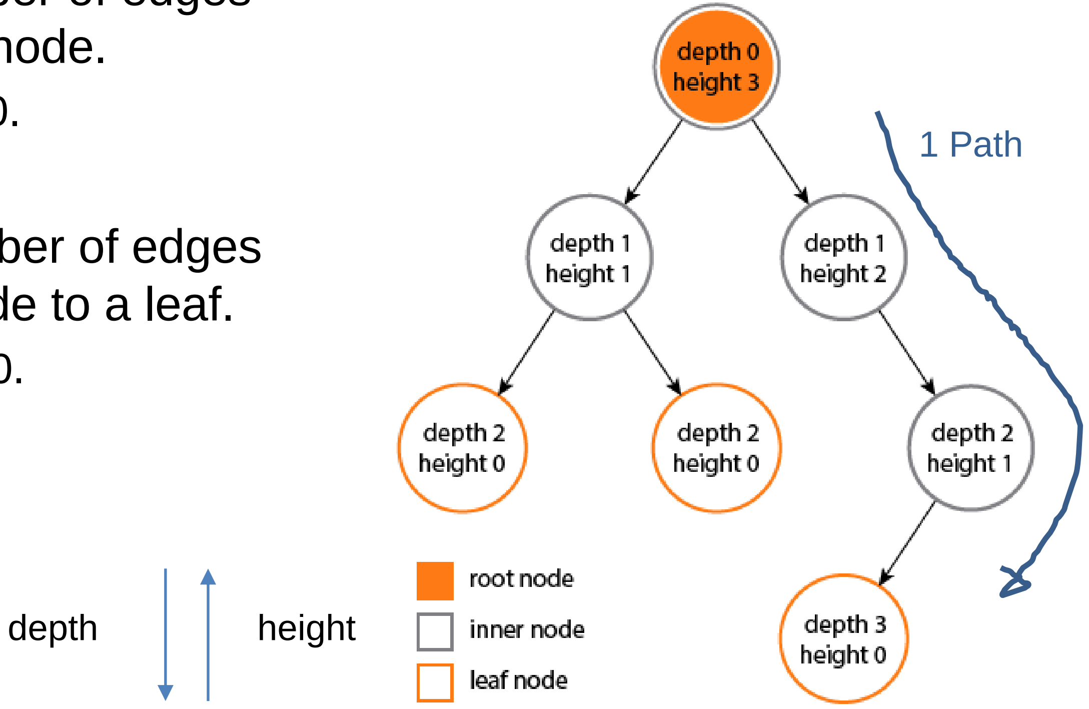
Trees

- Organizes Nodes into the shape of a Tree (literally)
- Always 1 Root Node (drawn at the top)
- Each Node can have 0 or more child Nodes
- Each Node must have exactly 1 parent (except root node)
- A Node which has no children is called a Leaf Node



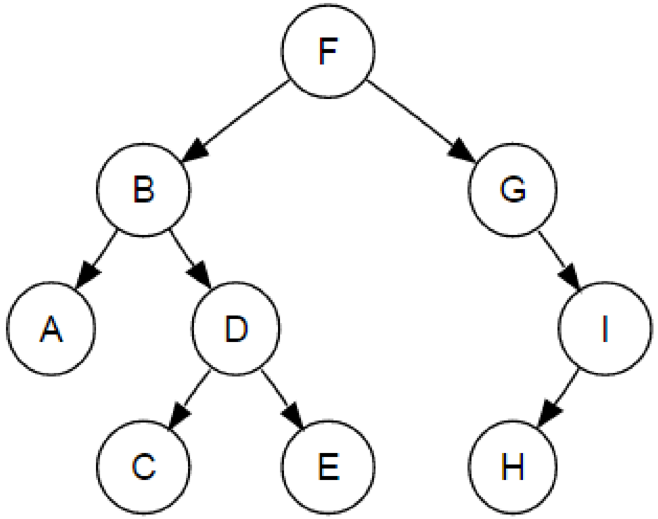
Trees

- The depth of a node is the number of edges from the node to the tree's root node.
  - A root node will have a depth of 0.
- The height of a node is the number of edges on the *longest path* from the node to a leaf.
  - A leaf node will have a height of 0.
- Used mainly for searching an organized collection of data more efficiently than a linear collection



Binary Tree

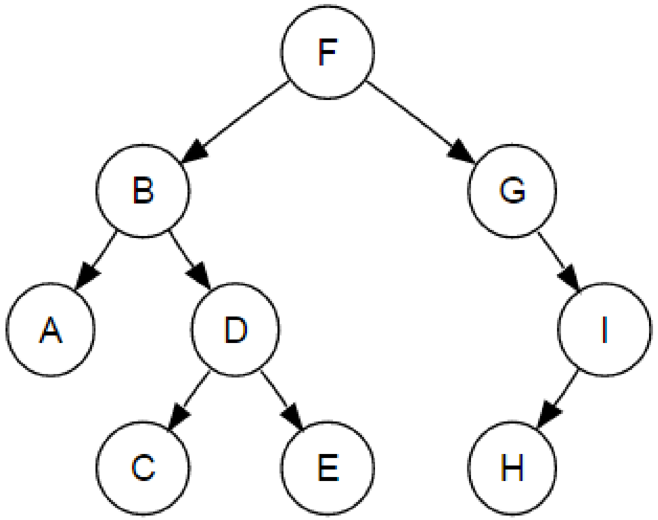
- A Tree where each node has at most 2 child nodes



Binary Search Tree

- A Binary Tree where at every single node, elements smaller than the node can be found in the left branch and elements larger than the node can be found in the right branch

- Future Questions (for BME 122):
  - Which element to pick as Root Node?
  - How to insert or remove a node while preserving the property?
  - How to balance the tree?



WA6

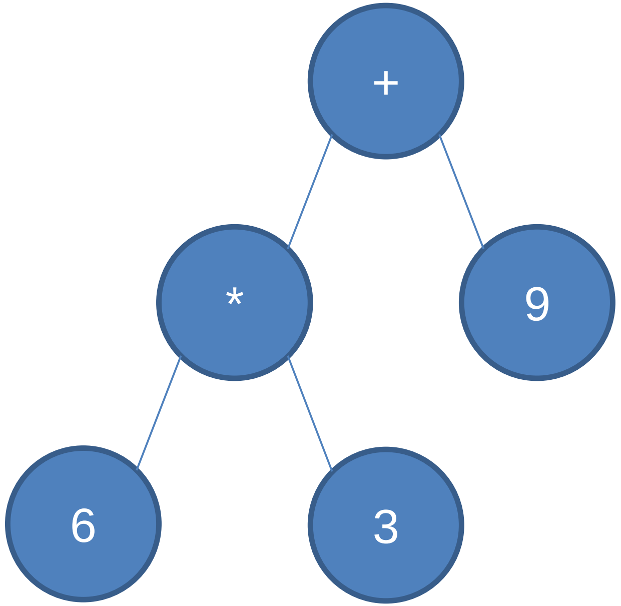
- C++ has a long history, with far more than 1 way of solving a problem. It can also compile C code and use C libraries.

- Converting string to double the C++ way (using streams):

```
#include <sstream>        // add this to top of program
double value;
stringstream convert( someString );
convert >> value;
// you can now use value. Note that conversions can fail just like C#'s Parse.
```

WA6

- The program constructs a binary computation tree for you
  - string data
  - Node * left
  - Node * right
- Main calls evaluate() on the root node
- Goal: calculate the final answer of the equation modeled by the tree
- Hint:
  - Recurrence: if the node is + then use recursion to add the final value of the left sub-tree with the final value of the right sub-tree, similarly for - * and /
  - Stop condition: if the node isn't one of the 4 operators, then convert the string to a double and return the double



WA6 Example Run:

```
5
6
7
8
9
+
-
*
/
=
expression: ( 5 / ( 6 * ( 7 - ( 8 + 9 ) ) ) ) = -0.0833333
```