CSE 100: BSTS AND C++

Announcements

- PA1 coming soon
  - Checkpoint – implement BST
  - Final – implement methods in a KD Tree (we'll cover KD trees next week)
- Homework 1
  - Completeness vs. correctness
- Tutoring hours start this week. Please see the tutor calendar, as they change week to week.
- My office hours are set. Tuesdays 9:30-11:30am. Please see the Office Hours and/or tutor calendar for any updates and changes weekly

Goals for today

- Draw memory model diagrams for C++ pointers
- Explain C++ code for implementing binary search trees
- Explain memory management in C++

PA1, Part 1: Implementing BST operations in C++

- You need to implement at least these..
  - insert() – insert an element (note return type)
  - find () – find an element (note return type)
  - size() – returns total number of elements
  - clear() – deletes all the elements
  - empty() – checks if the BST is empty
  - height() – returns the height of the BST. Empty trees have height 0. Leaf nodes are height 1. So on.
  - successor() – returns the next element in an in-order traversal
  - And the iterator pattern (we will talk about it later this week)

Chapters 3.1 and 3.3 in the Stepik book will help, A LOT!

Today's topic: C++

C++'s main priority is getting correct programs to run as fast as it can; incorrect programs are on their own.

Java's main priority is not allowing incorrect programs to run; hopefully correct programs run reasonably fast, and the language makes it easier to generate correct programs by restricting some bad programming constructs.

-- Mark Allen Weiss, *C++ for Java Programmers*

Why C++ for data structures?

Let's design a BST!

- Design the class or classes you would need for a BST that holds integers. What fields would each class have? What methods?

An int-based BST in C++

The header file has the class declaration

In: `BSTNodeInt.h`

```
class BSTNodeInt {
public:
  BSTNodeInt* left;
  BSTNodeInt* right;
  BSTNodeInt* parent;
  int const data;

  BSTNodeInt( const int & d );
};
```

You may put the implementation in a separate file

In: `BSTNodeInt.cpp`

```
BSTNodeInt::BSTNodeInt(const int & d)
  : data(d) {
     left = right = parent = nullptr;
}
```

We'll spend all of today making sense of the pieces of this code

An int-based BST in C++

In: BSTNodeInt.h

```
class BSTNodeInt {
public:
  BSTNodeInt* left;
  BSTNodeInt* right;
  BSTNodeInt* parent;
  int const data;

  BSTNodeInt( const int & d );
};
```

In: bstTest.cpp

```
#include "BSTNodeInt.h"
#include <iostream>

using namespace std;

int main()
{
    BSTNodeInt n1(5);
    cout << "Created a BST node with data "
         << n1.data << endl;
}
```

Assume that BSTNodeInt is implemented correctly. Which of the following is true about the highlighted line of code?

A. It creates a BSTNodeInt object
B. It stores the address of a BSTNodeInt object in the variable n1
C. It causes a compile error
D. Both A and B

An int-based BST in C++

In: BSTNodeInt.h

```
class BSTNodeInt {
public:
  BSTNodeInt* left;
  BSTNodeInt* right;
  BSTNodeInt* parent;
  int const data;

  BSTNodeInt( const int & d );
};
```

In: bstTest.cpp

```
#include "BSTNodeInt.h"
#include <iostream>

using namespace std;

int main()
{
    BSTNodeInt n1(5);
    cout << "Created a BST node with data "
         << n1.data << endl;
}
```

Let's draw the memory diagram for the highlighted code

An int-based BST in C++

In: BSTNodeInt.h

```
class BSTNodeInt {
public:
  BSTNodeInt left;
  BSTNodeInt right;
  BSTNodeInt parent;
  int const data;

  BSTNodeInt( const int & d );
};
```

In: bstTest.cpp

```
#include "BSTNodeInt.h"
#include <iostream>

using namespace std;

int main()
{
    BSTNodeInt n1(5);
    cout << "Created a BST node with data "
         << n1.data << endl;
}
```

compile error is rule

left, right and parent were all variables that are pointers to BSTNodeInt type objects. What would happen if I made them BSTNodeInt type variables, as above?

A. The code would work as it did before
B. The code would have a memory leak
C. The code would cause a compile error
D. The code would cause a runtime error (but no compile error)

Pointers in C++

Which of the following statements is true about this code?

```
int a = 5;
int b = a;
int* pt1 = a;
```

A. Both pt1 and b can be used to change the value of a.
B. Only pt1 can be used to change the value of a.
C. This code causes a compile error.

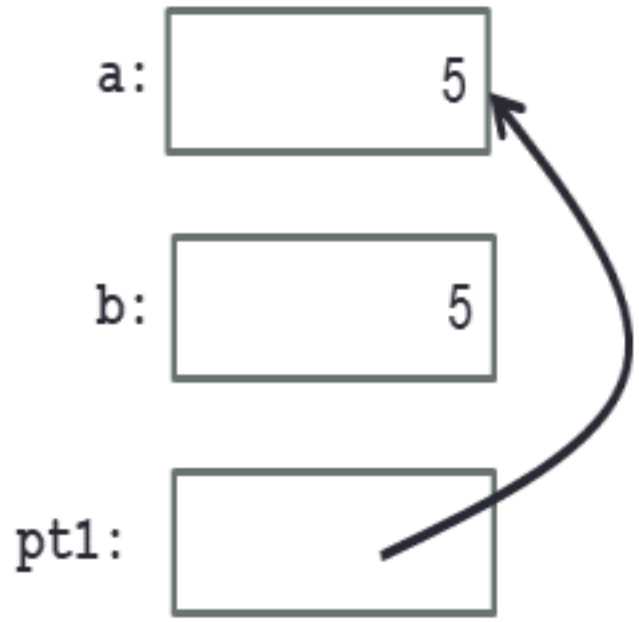
Pointers in C++

```
int a = 5;
int b = a;
int* pt1 = &a;
```

address	memory cell	identifier
512000	5	a
512004	5	b
512008	512000	pt1

Pointers in C++

```
int a = 5;
int b = a;
int* pt1 = &a;
```



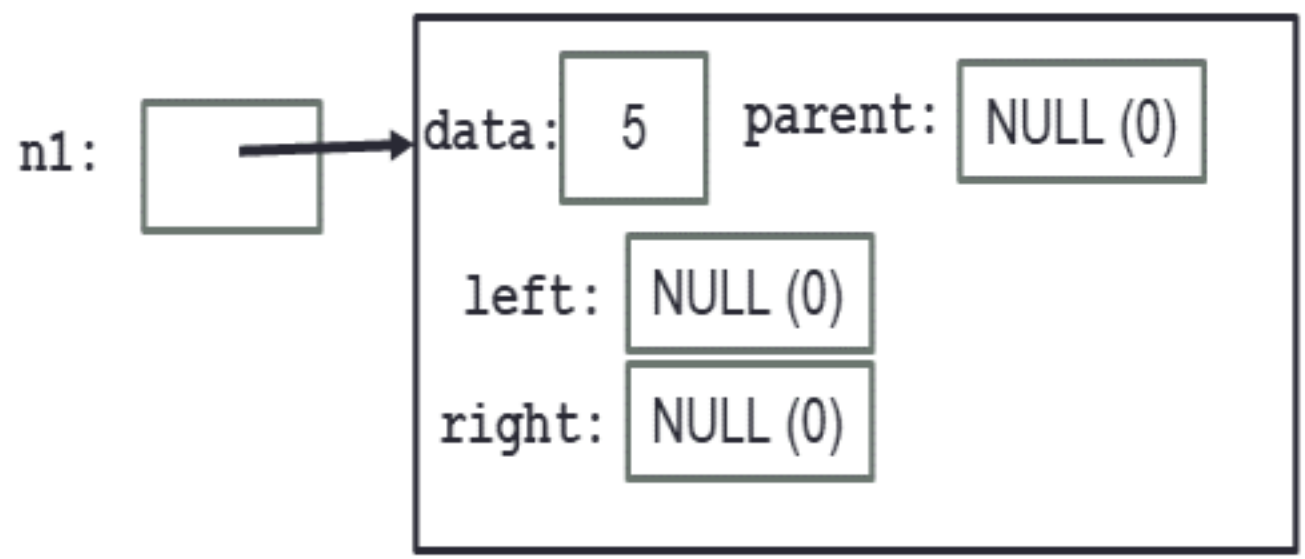
An int-based BST in C++

In: BSTNodeInt.h

```
class BSTNodeInt {
public:
  BSTNodeInt* left;
  BSTNodeInt* right;
  BSTNodeInt* parent;
  int const data;

  BSTNodeInt( const int & d );
};
```

In: bstTest.cpp

```
#include "BSTNodeInt.h"
#include <iostream>

using namespace std;

int main()
{
    BSTNodeInt* n1 = new BSTNodeInt(5);
    cout << "Created a BST node with data "
         << n1->data << endl;
}
```



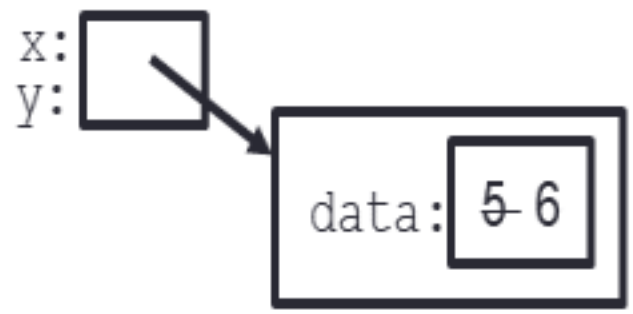
Pointers in C++

```
int main() {
  BSTNodeInt* x;          // declare x to be a pointer to a BSTNodeInt object
  x = new BSTNodeInt(5); // create a BSTNodeInt object, and make x point to it
  x->data = 6;            // dereference x, and access a member
                          // note: (*x).data = 6 is equivalent
  BSTNodeInt* y = x;
}
```

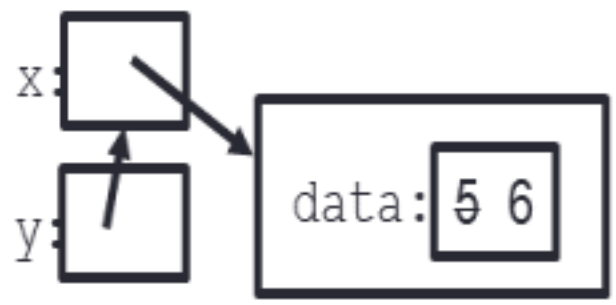
Which represents the diagram (`left`, `right` and `parent` not shown)?

A.

B.



C.



D. The line in red causes an error

An int-based BST in C++

In: BSTNodeInt.h

```
class BSTNodeInt {
public:
  BSTNodeInt* left;
  BSTNodeInt* right;
  BSTNodeInt* parent;
  int const data;

  BSTNodeInt( const int & d );
};
```

In: bstTest.cpp

```
#include "BSTNodeInt.h"
#include <iostream>

using namespace std;

int main()
{
    BSTNodeInt* n1 = new BSTNodeInt(5);
    cout << "Created a BST node with data "
         << n1->data << endl;
}
```

This code will run, but technically there is a problem with it. Can you find it?

An int-based BST in C++

In: BSTNodeInt.h

```
class BSTNodeInt {
public:
  BSTNodeInt* left;
  BSTNodeInt* right;
  BSTNodeInt* parent;
  int const data;

  BSTNodeInt( const int & d );
};
```

In: bstTest.cpp

```
#include "BSTNodeInt.h"
#include <iostream>

using namespace std;

int main()
{
    BSTNodeInt* n1 = new BSTNodeInt(5);
    cout << "Created a BST node with data "
         << n1->data << endl;
    delete n1;
}
```

Fixing the memory leak!

You must `delete` every piece of data you create with `new`. But usually there's no need to also set the pointer to NULL.
`delete` will call the object's destructor, if one is defined.

An int-based BST in C++

In: BSTNodeInt.h

```
class BSTNodeInt {
public:
  BSTNodeInt* left;
  BSTNodeInt* right;
  BSTNodeInt* parent;
  int const data;

  BSTNodeInt( const int & d );
};
```

In: bstTest.cpp

```
#include "BSTNodeInt.h"
#include <iostream>

using namespace std;

int main()
{
    BSTNodeInt n1(5);
    BSTNodeInt smaller(2);
    n1.left = &smaller;
}
```

need free memory

Draw the memory diagram for the code above and then state what is true about the code:

A. It has a memory leak
B. It will cause a compile error
C. Neither A nor B: It compiles and runs perfectly fine (though it doesn't really do much)

An int-based BST in C++

In: `BSTNodeInt.h`

```
class BSTNodeInt {
public:
  BSTNodeInt* left;
  BSTNodeInt* right;
  BSTNodeInt* parent;
  int const data;

  BSTNodeInt( const int & d );
};
```

In: `bstTest.cpp`

```
#include "BSTNodeInt.h"
#include <iostream>

using namespace std;

int main()
{
    BSTNodeInt n1(5);
    BSTNodeInt smaller(2);
    n1.left = &smaller;
}
```

So WHY use dynamic memory allocation??

Implementation and Compilation

An int-based BST in C++

In: `BSTNodeInt.h`

```
class BSTNodeInt {
public:
  BSTNodeInt* left;
  BSTNodeInt* right;
  BSTNodeInt* parent;
  int const data;

  BSTNodeInt( const int & d );
};
```

In: `BSTNodeInt.cc`

```
BSTNodeInt::BSTNodeInt(const int & d) : data(d) {
    left = right = parent = nullptr;
}
```