CSE 100: BSTS AND C++
Announcements

• PA 1 released due next Thursday (10/8) at 8pm
  • Start early!!
  • First grading 10/6, final grading 10/8

• Discussion section tomorrow
  • Section B, 8:00a-8:50a, WLH-2005
  • Topic: Getting started with PA1 and C++

• Reading Quizzes
  • Due every Tuesday at 8am offered through TED.
Goals for today

• Draw memory model diagrams for C++ pointers and references
• Explain C++ code for implementing binary search trees
• Explain pass-by-reference and constants in C++
Integrity Guidelines

Basic rules
- Do not look at or copy other people’s code and do not share your code with others (other than your partner). Period.
- “Other people” includes what you can find/share on the internet.
- Read the Integrity Statement carefully. Ask if you have questions.

Integrity
- You will be tested on your ability to understand and write code for data structures in this class (and invariably during interviews)
- Cheaters will likely get “caught” during the exam because exams, for the most part, make your grade in this class.
- Why else shouldn’t you cheat?
  - Its unethical
  - Its unfair to students who do the work legitimately
  - Hurts the reputation of the UCSD CSE degree
BST Operations

• Your first Programming Assignment will ask you to implement:
  • Find
  • Insert
  • Clear
  • Size
  • An Iterator
  • (A few other methods)

• We will assume that you have already seen these operations and/or can learn them from the reading. We will not explicitly cover (most of) them in class.
Under the hood: Searching an element in the BST

To search for element with key $k$
1. Start at the root
2. If $k = \text{key(root)}$, found key, stop.
3. Else if $k < \text{key(root)}$, recursively search the left subtree: $T_L$
   Else recursively search the right subtree: $T_R$

Search for 41.
Now search for 43.
Under the hood: Finding the successor of an element in the BST

Which node is the successor of 56?
How would you find it?
Under the hood: Finding the successor of an element in the BST

Which node is the successor of 18?
How would you find it?
Traversing the BST in sorted order

Different methods of tree traversal:
- Pre order traversal
- Post order traversal
- In order traversal

successor() – returns the next element in an in-order traversal
In-order traversal of BST

Which of the following results from an in-order traversal of a BST?

A. Nodes are visited in the order in which they were inserted into the BST
B. Nodes are visited in order of the number of children that they have
C. Nodes are visited in sorted order of their keys
D. None of the above
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?
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;
    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 1:

```cpp
class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;
    public BSTNode( const int & d ) {
        data = d;
    }
};
```

Which of the following is a problem with the C++ implementation above?
A. You should not declare the types of your variables in C++
B. The class BSTNode should be declared public
C. The semicolon at the end of the class will cause a compile error
D. In C++ you specify public and private in regions, not on each variable or function
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 2:

```cpp
class BSTNode {
    public:
        BSTNode left;
        BSTNode right;
        BSTNode parent;
        int data;

    public BSTNode( const int & d ) {
        data = d;
    }
};
```

- The other problem is with how we have declared left, right and parent above.
- They should be `BSTNode*` (pointers to `BSTNode`es) and not `BSTNode` type.
Which of the following statements is true about this code?

```cpp
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++

```cpp
type a = 5;
type b = a;
type* pt1 = &a;
```

<table>
<thead>
<tr>
<th>address</th>
<th>memory cell</th>
<th>identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>512000</td>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td>512004</td>
<td>5</td>
<td>b</td>
</tr>
<tr>
<td>512008</td>
<td>512000</td>
<td>pt1</td>
</tr>
</tbody>
</table>
Pointers in C++

```cpp
int a = 5;
int b = a;
int* pt1 = &a;
```
class MyClass {
    private:
        int a;
    public:
        void setA(int a) { this->a = a; }
        int getA() { return a; }
};

What will the line
MyClass c;
do?
A. Declare a variable of type MyClass, but not create an object
B. Declare a variable of type MyClass and create an object of type MyClass
C. Declare a variable of type pointer to MyClass, but not create an object
D. Declare a variable of type pointer to MyClass, and create an object of type MyClass
Pointers in C++

class C {
  private:
    int a;
  public:
    void setA(int a) { this->a = a; }
    int getA() { return a; }
};

What will the line
C* c;
do?
A. Declare a variable of type C, but not create an object
B. Declare a variable of type C and create an object of type C
C. Declare a variable of type pointer to C, but not create an object
D. Declare a variable of type pointer to C, and create an object of type C
Pointers in C++

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

```
x:
     a:
         5
```
Pointers in C++

```c++
int main() {
    C* x; // declare x to be a pointer to a C object
    x = new C(); // create a C object, and make x point to it
    x->setA(5); // dereference x, and access a member
    // note: (*x).setA(5) is equivalent

    C* y = x;
}
```

Which represents the new diagram?

A. ![Diagram A]
B. ![Diagram B]
C. ![Diagram C]
D. The line in red causes an error
References in C++

```cpp
int main() {
    int d = 5;
    int & e = d;
}
```

The diagram that represents the code above is C

A. d: 5
   e: 5

B. d: 5
   e: 

C. d: 5
   e: 5

D. This code causes an error
int main() {
    int d = 5;
    int & e = d;
    int f = 10;
    e = f;
}

How does the diagram change with this code?
Pointers and references. Draw the picture for this code is the one on the left.

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

What are three ways to change the value in the box to 42?
Pointers and references. Draw the picture for this code is the one on the left.

```c
int a = 5;
int & b = a;
int* pt1 = &a;
```
References in C++

```cpp
int main() {
    int const d = 5;
    int & e = d;
}
```

Does this code have an error? If so, why?

A. No, there is no error
B. Yes, there is an error because ints cannot be constant in C++
C. Yes, there is an error because a reference to a constant must also be declared constant
References in C++

```cpp
int main() {
    int const d = 5;
    const int & e = d;
}
```
C++, attempt 3:

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

    BSTNode( const int & d ) {
        // body here
    }
};
```

Imagine this code works (it doesn’t yet). If it did, consider creating a new BSTNode as follows:

```cpp
int myInt = 42;
BSTNode* myNode = new BSTNode( myInt );
```

The code above in red specifies that d is passed by constant reference. Which of the following diagrams best represents what that means?

A. myInt: 42  
   d: 42  
   d is not allowed to change what’s in its box

B. myInt: 42  
   d:  
   The address in d’s box can’t be changed

C. myInt: 42  
   d: 42  
   d can’t change myInt because there’s no connection

D. This code has an error
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 3:

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

    BSTNode( const int & d ) {
        data = d;
    }
};
```

Another problem with the C++ implementation above is that:

Because data is a constant variable, the constructor will cause an error.
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 4:

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

        BSTNode( const int & d ) :
        data(d) { }
};
```

One more potential issue....
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 5:

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

    BSTNode( const int & d ) :
    data(d) {
        left = right = parent = 0;
    }
};
```

ALWAYS initialize in C++. C++ won’t do it for you. Why not?
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 5:

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

        BSTNode( const int & d ) :
        data(d) {
            left = right = parent = 0;
        }

};
```

ALWAYS initialize in C++. C++ won’t do it for you. Why not?

What if we don’t want to be stuck with ints?
BST, with templates:

```cpp
template<typename Data>

class BSTNode {
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
        data(d) {
            left = right = parent = 0;
        }
};
```
BST, with templates:

```cpp
template<typename Data>

class BSTNode {
 public:
   BSTNode<Data>* left;
   BSTNode<Data>* right;
   BSTNode<Data>* parent;
   Data const data;

   BSTNode( const Data & d ) :
   data(d) {
      left = right = parent = 0;
   }

};
```

A. How would you create a **BSTNode** object on the runtime stack?
BST, with templates:

```
template<typename Data>

class BSTNode { 
    public:
        BSTNode<Data>* left;
        BSTNode<Data>* right;
        BSTNode<Data>* parent;
        Data const data;

        BSTNode( const Data & d ) :
            data(d) {
                left = right = parent = 0;
            }

};
```

B. How would you create a pointer to BSTNode with integer data?
BST, with templates:

```cpp
template<typename Data>

class BSTNode { 
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
      data(d) {
        left = right = parent = 0;
      }

};
```

C. How would you create an **BSTNode** object on the heap?
BST, with templates:

```cpp
template<typename Data>

class BSTNode {
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
        data(d) {
            left = right = parent = 0;
        }
};
```

BSTNodes will be used in a BST, and with a BSTIterator…
Working with a BST

template<typename Data>
class BST {

private:

    /** Pointer to the root of this BST, or 0 if the BST is empty */
    BSTNode<Data>* root;

public:

    /** Default constructor. Initialize an empty BST. */
    BST() : root(nullptr){
    }

    void insertAsLeftChild(BSTNode<Data>* parent, const Data & item)
    {
        // Your code here
    }
Working with a BST: Insert

```cpp
void insertAsLeftChild(BSTNode<Data>* parent, const Data & item) {
    // Your code here
}
```

Which line of code correctly inserts the data item into the BST as the left child of the parent parameter.
A. `parent.left = item;`
B. `parent->left = item;`
C. `parent->left = BSTNode(item);`
D. `parent->left = new BSTNode<Data>(item);`
E. `parent->left = new Data(item);`
CHANGING GEARS: C++STL and BSTs

- The C++ Standard Template Library is a very handy set of built-in data structures (containers), including:
  - array
  - vector
  - deque
  - forward_list
  - list
  - stack
  - queue
  - priority_queue
  - set
  - multiset (non unique keys)
  - unordered_set
  - map
  - unordered_map
  - multimap
  - bitset

Of these, `set` is one that is implemented using a balanced binary search tree (typically a red-black tree)
Imagining ourselves as C++ STL class designers…

- set’s find function has this prototype:

```cpp
template <typename T>

class set {

public:
    iterator find ( T const & x ) const;

};
```

What does the final const in the function header above mean?
A. find cannot change its input argument
B. find cannot change the value of its input argument (x)
C. find cannot change the underlying set
Imagining ourselves as C++ STL class designers...

- set’s find function has this prototype:

```cpp
template <typename T>

class set {

public:

    iterator find ( T const & x ) const;

```

The documentation for set’s find function says:

*Searches the container for an element with a value of x and returns an iterator to it if found, otherwise it returns an iterator to the element past the end of the container.*
C++ STL Iterators

What is an iterator?

• In the iterator pattern of OO design, a container has a way to supply to a client an iterator object which is to be used by the client to access the data in the container sequentially, without exposing the container’s underlying representation.
Iterator class template for BST

template <typename T>
class BSTIterator {

private:
    Node<T>* curr;

public:
    /** Constructor */
    BSTIterator(Node<T>* n) : curr(n) {}
C++ STL Iterators

set<int> c;
...
// get an iterator pointing to container’s first element
set<int>::iterator itr = c.begin();

What do you think `begin()` returns?
A. The address of the root in the set container class
B. The address of the node with the smallest data key
C. The address of the smallest data key
D. None of the above
C++ STL Iterators

```cpp
set<int> c;
...
// get an iterator pointing to container’s first element
set<int>::iterator itr = c.begin();
// get an iterator pointing past container’s last element
set<int>::iterator end = c.end();
// loop while itr is not past the last element
while(itr != end) {
    cout << *itr << endl; // dereference the itr to get data
    ++itr; // increment itr to point to next element
}
```

Circle/list the functions that the iterator has to implement.
In which class is successor() defined?
A. BST
B. BSTNode
C. BSTIterator
D. Data
C++ STL Iterators

set<int> c;
...
// get an iterator pointing to container’s first element
set<int>::iterator itr = c.begin();
// get an iterator pointing past container’s last element
set<int>::iterator end = c.end();
// loop while itr is not past the last element
while(itr != end) {
    cout << *itr << endl; // dereference the itr to get data
    ++itr; // increment itr to point to next element
}

What kind of traversal is the above code doing?
A. In order
B. Pre order
C. Post order
D. None of the above