CSE 100: C++
TEMPLATES AND
ITERATORS
Announcements

• Gradesource and clickers:
  • We’ll be making one more pass for unregistered clickers tonight, but after that you’ll be on your own…
How is Assignment 1 going?

A. I haven’t looked at it.
B. I’ve read it, but I haven’t done anything
C. I’ve gotten the code and possibly started looking at it/playing around with it.
D. I’ve implemented some of the required functions, but I’m not done.
E. I’m done!
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:

```
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:

```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;
        }
};
```

B. How would you create a \texttt{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…
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 where its input argument, which is a pointer, points to
C. find cannot change the underlying set
Imagining ourselves as C++ STL class designers…

- set’s find function has this prototype:

```c++
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?
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.
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 \texttt{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
Iterator class template for BST

```cpp
template <typename T>
class BSTIterator {

private:
    Node<T>* curr;

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

Is this definition of the BSTIterator class complete?
A. Yes
B. No
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.
template<typename Data>
class BSTIterator : public std::iterator<std::input_iterator_tag, Data> {

private:
    BSTNode<Data>* curr;

public:
    /** Constructor. Use the argument to initialize the current BSTNode */
    /** in this BSTIterator. */ // TODO
    BSTIterator(BSTNode<Data>* curr) { // TODO }

    /** Dereference operator. */
    Data operator*() const {
        return curr->data;
    }

    /** Pre-increment operator. */
    BSTIterator<Data>& operator++() {
        curr = curr->successor();
        return *this;
    }
};
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