Announcements

• Look out for the extra weekend section
  • More on git and C++ (iterators)
  • Live demo by your friendly tutors
  • Not mandatory but it will be fun
  • Bring your laptops to follow along
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:

```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?

- `BSTNode bst;` (without template)
- `BSTNode<int> bst;` (compiler error, no match with constructor)
- `BSTNode<int> bst(10);` (name of object)
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:

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?

BSTNode<int> *p = new BSTNode<int>(10);
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
  }
}
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);
The C++ Standard Template Library is a very handy set of built-in data structures (containers), including:

- `array`
- `vector` → elements contiguous in memory + dynamic resizing
- `deque`
- `forward_list` → singly linked list
- `list` → doubly linked list
- `stack`
- `queue`
- `priority_queue` → Heap: we will use this a lot!
- `set` → Good old BST (Balanced) for faster operations
- `multiset` (non unique keys)
- `unordered_set`
- `map`
- `unordered_map` → Hashmap
- `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:
BSTNode<T>* curr;

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

};
```
C++ STL Iterators

```cpp
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

---

A. The address of the root in the set container class
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
}
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