

CSE 100: BST OPERATIONS

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!

Q1: Which of the following is NOT a type of depth-first traversal of a BST?

- A. In-order LVR
- B. Pre-order VLR
- C. Post-order LRV
- ☒ D. Parallel-order

Q2: Which abstract data type is naturally used to support a depth-first traversal of a BST?

- A. Stack
- B. Queue
- C. Dictionary
- D. Graph

- ① Recursion (using runtime stack)
- ② Stack.
- ③ PA1: Successor & iterators

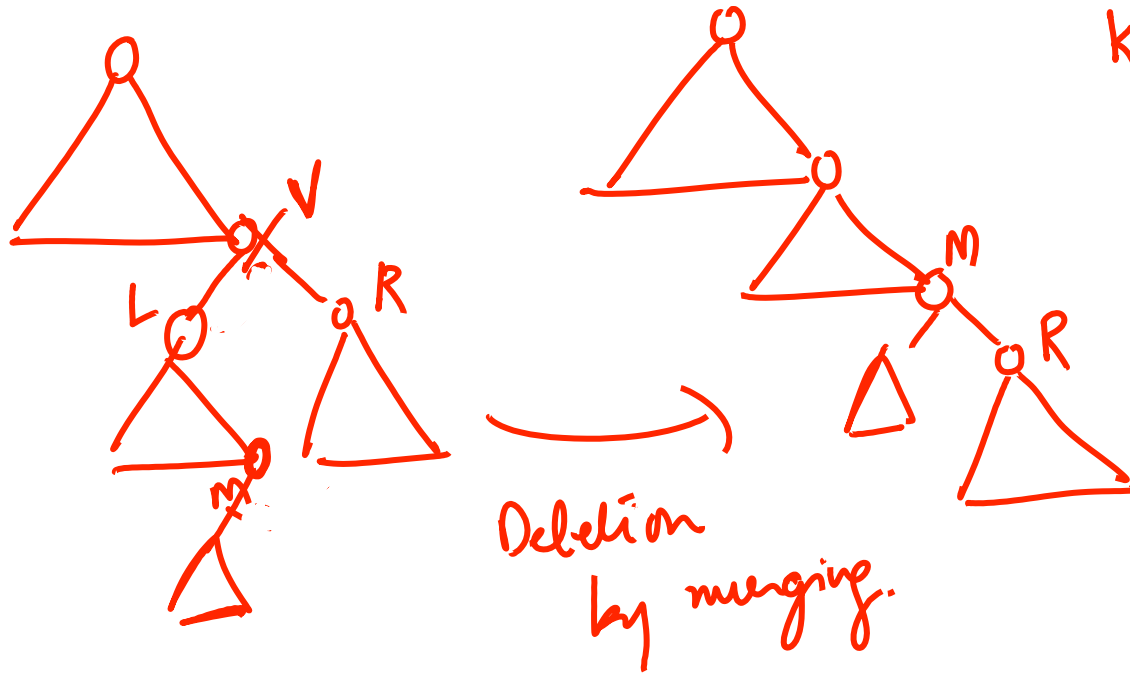
Q3: True or False: An element inserted into a BST is always inserted as a leaf. (Recall that a leaf is simply a node with no children, which includes the root).

- ☒ A. True
- ☐ B. False

Q4: When is 'Deletion by Merging' used while deleting a node from a BST?

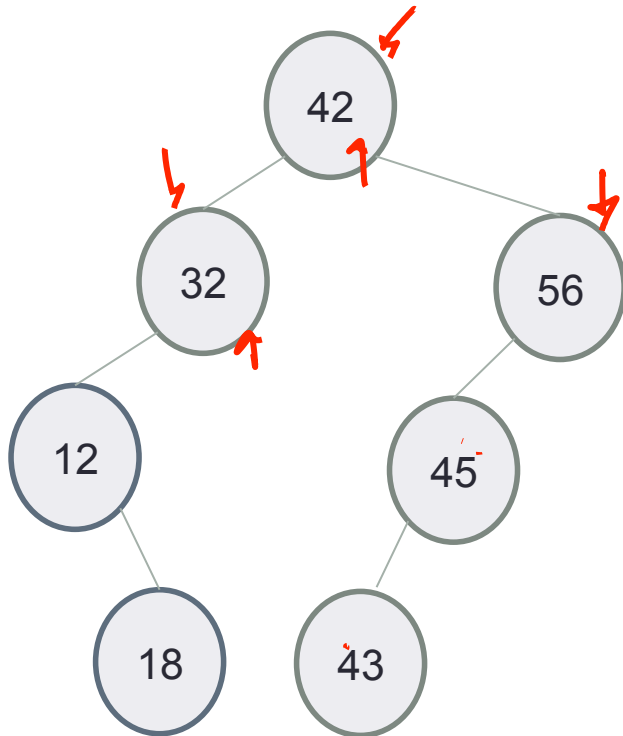
- A. When the node being deleted is a leaf node
- B. When the node being deleted has one child
- ☒ C. When the node being deleted has two children
- D. When the node being deleted has one subtree

Deleting a node from a BST



$$\begin{aligned} \text{key}(\bar{T}_R) &> \text{key}(\bar{T}_L) \\ &> \max(\text{key } \bar{T}_L) \end{aligned}$$

In-order traversal of BST

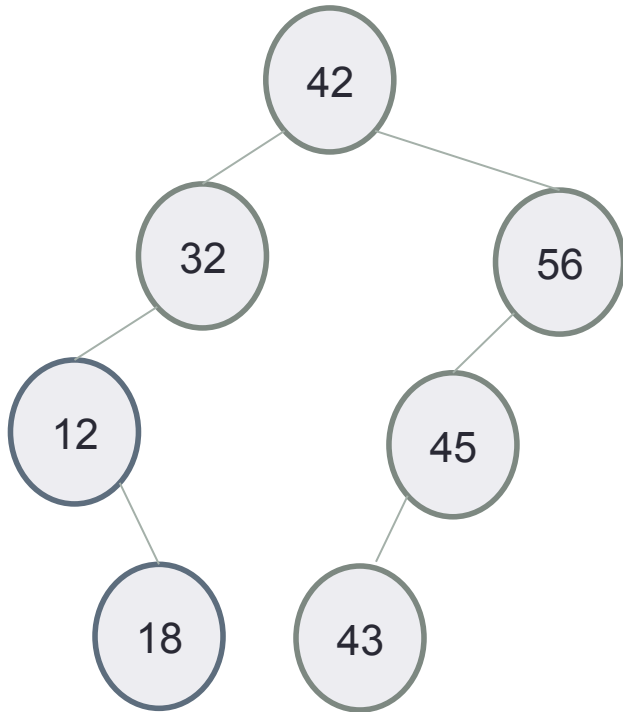


12, 18, 32, 42, 43, 45, 56 ←

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

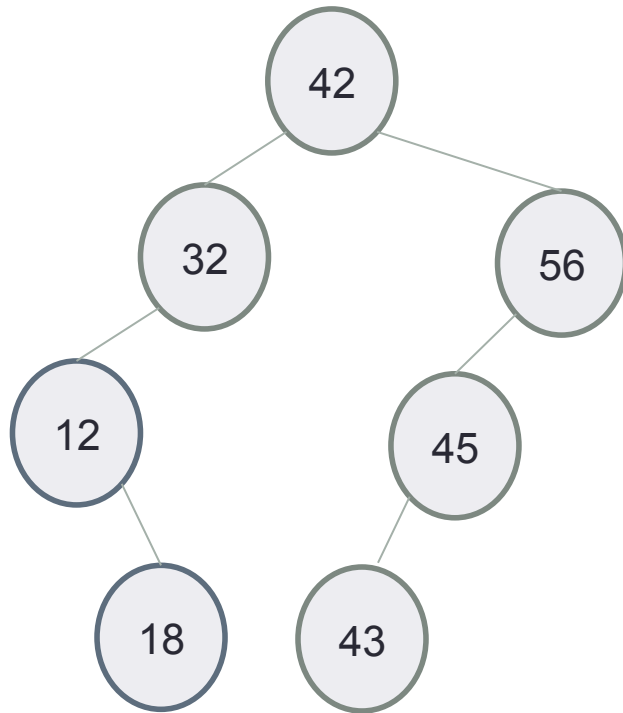
The successor method performs an in-order tree traversal, one step at a time.



- ☒ A. True
- ☐ B. False

but the successor method should start with the node that has the smallest key

The successor method performs an in-order tree traversal, one step at a time.



Find several different cases to consider in the tree below; describe how to find the successor in each case.

Case 1: Node has a right subtree T_R
Successor is the node with min key in T_R

Case 2: Node has no right subtree
Find a parent (a node higher up in the tree) that contains the node in its left subtree

Prove: If a node has a right subtree, its successor is contained in the right subtree.

Prove: If a node has a right subtree, its successor resides in its right subtree

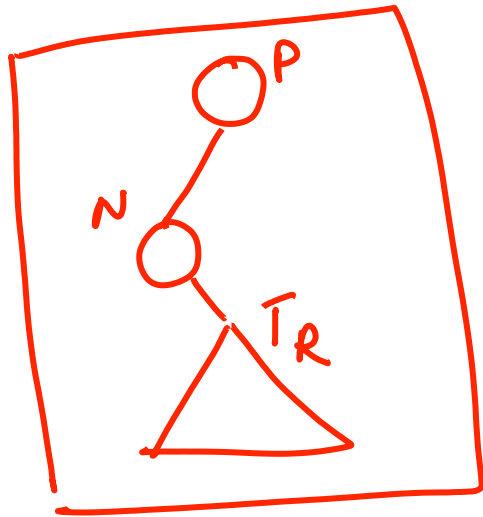
Case 1. Consider a node that is in the left subtree of its parent (P)

By the property of BSTs:

$$\text{key}(N) < \min\{\text{key}(T_R)\} \quad \text{--- (1)}$$

Since T_R is in the left subtree of P

$$\min\{\text{key}(T_R)\} < \text{key}(P)$$

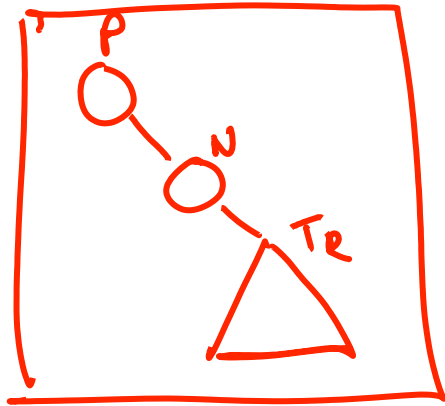


Case 1

From (1), (2)

$$\text{key}(N) < \min\{\text{key}(T_R)\} < \text{key}(P)$$

Case 2: The node(N) is in the right subtree of its parent(P)



Case 2

T_R : Right subtree of N

By the property of BSTs:

$$\text{key}(P) < \text{key}(N) < \text{key}(T_R)$$

In this case again the successor of N should be in T_R , specifically it is the $\min \{ \text{key}(T_R) \}$ i.e. the node with the min key in T_R

In 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:

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

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

Handwritten red notes:
: data(d)
An arrow points from the underlined data = d; to the data(d) in the handwritten note.

Which of the following is a problem with the C++ implementation above?

- ☒ A. Because data is a constant variable, the constructor will cause an error.
- ☐ B. You cannot pass an integer by reference into a function. Integers must be passed by value.
- ☐ C. Since d is passed by reference, you cannot assign its value to data, which is an int. You need to dereference it first.
- ☐ D. The constructor needs a semi-colon at the end of its definition.

In 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:

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

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

Why make data const?

Which of the following is a problem with the C++ implementation above?

A. Because data is a constant variable, the constructor will cause an error.

In 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:

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

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

};
```

What is the problem with how we have declared left, right and parent above?

- A. They should be `BSTNode*` (pointers to `BSTNodes`) and not `BSTNode` type.
- B. They should be declared to be `const`
- C. They should be declared as `BSTNode&` (reference variables).

In 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:

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

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

};
```

And now, a little practice with pointers...

Which of the following statements is true about this code?

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

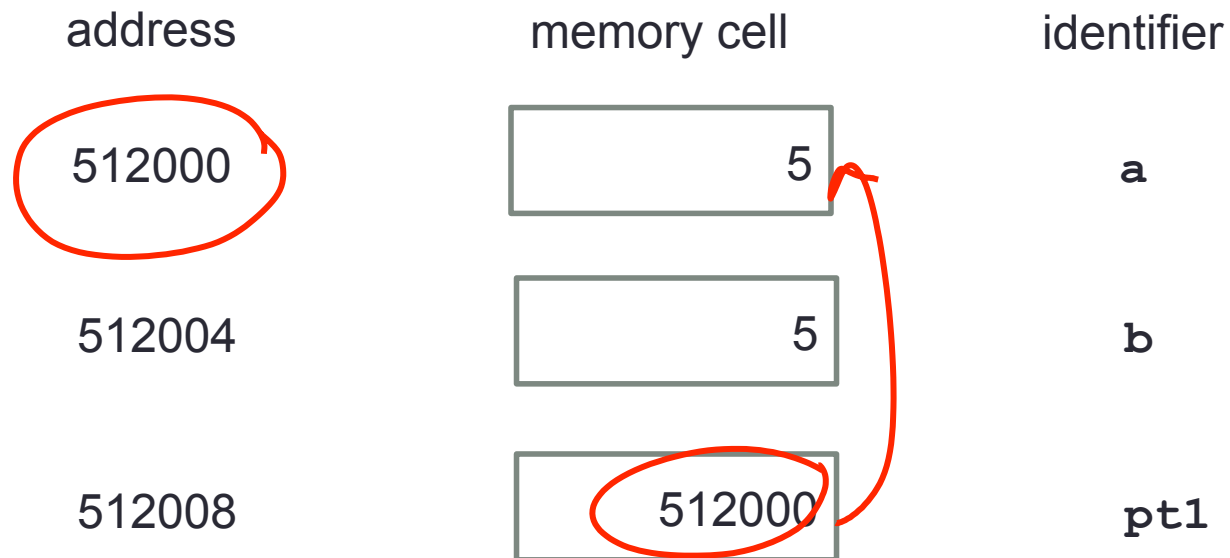
A. Both ptr and b can be used to change the value of a.

B. Only pts can be used to chance the value of a.

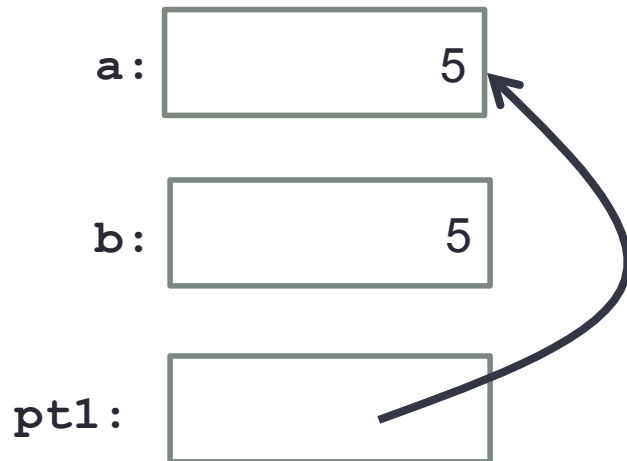
☒ C. This code causes a compile error.

*int *pt1 = &a; // Need to assign address of a to pt1*

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

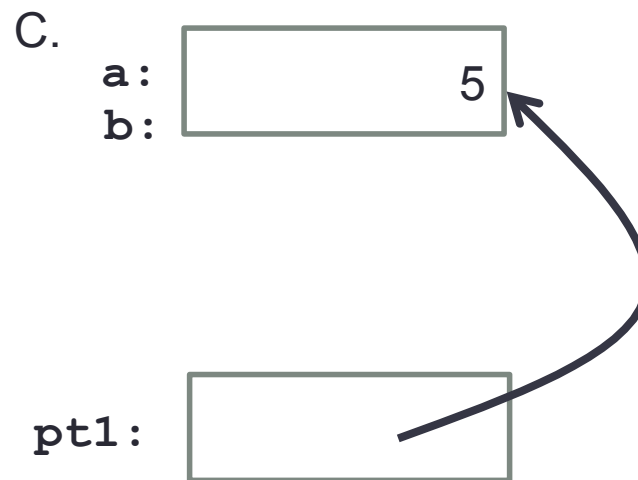
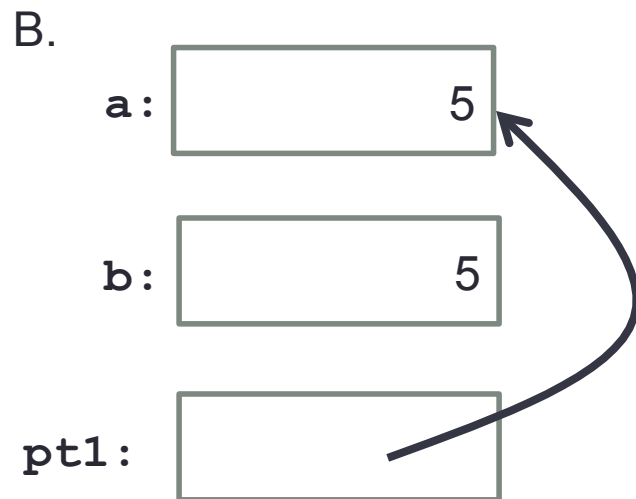
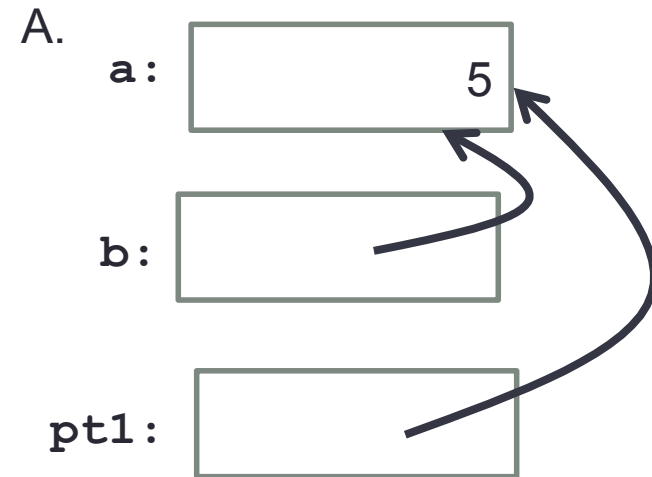


```
int a = 5;  
int b = a;  
int* pt1 = &a;
```



How does the diagram change if we change the code as follows. Which is the correct picture now?

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



In 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:

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

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

};
```

Are there any remaining problems with this C++ implementation?

A. Yes

B. No

In 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:

```
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;
```

```
    }
```

```
};
```