ECE 220

Lecture x001B - Final exam review

About last time

About last time

- Review tree traversals

About last time

- Review tree traversals

- Review BST code

Tree traversal

Tree traversal

- Last time, we printed a tree's boundary:

Tree traversal

- Last time, we printed a tree's boundary:
  - root + left boundary + leaves + right boundary

Tree traversal

- Last time, we printed a tree's boundary:
  - root + left boundary + leaves + right boundary
  - Did this in counter-clockwise fashion

Tree traversal

- Last time, we printed a tree's boundary:
  - root + left boundary + leaves + right boundary
  - Did this in counter-clockwise fashion
  - A remark was made: the type of traversal does not matter for the sequence in which leaf nodes are visited.

Tree traversal

- Last time, we printed a tree's boundary:
  - root + left boundary + leaves + right boundary
  - Did this in counter-clockwise fashion
  - A remark was made: the type of traversal does not matter for the sequence in which leaf nodes are visited.
    - True … ***BUT surely printing the leaf nodes in a clockwise fashion should be possible?***

Tree traversal

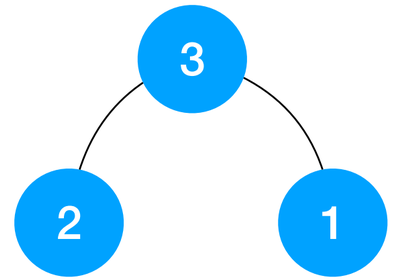
- Standard traversals are:
  - Preorder: N-L-R
  - Inorder: L-N-R
  - Postorder: L-R-N

Will do CCW

- *Reversed* traversals are:
  - Preorder: N-R-L
  - Inorder: R-N-L
  - Postorder: R-L-N

Will do CW

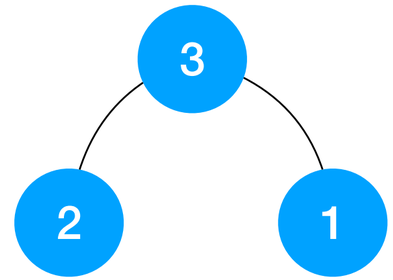
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}
```

```
node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

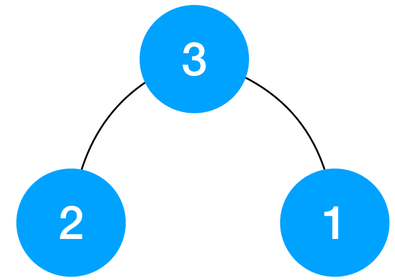
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}
```

```
node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

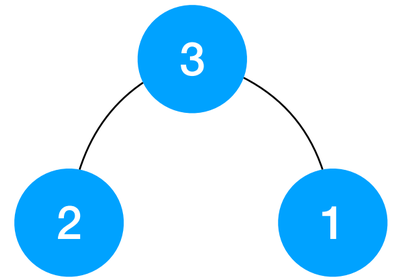
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right

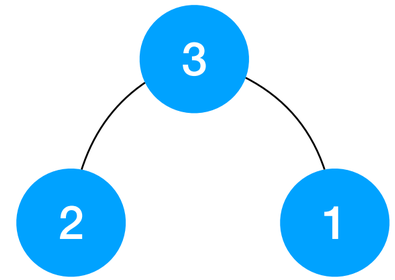
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		

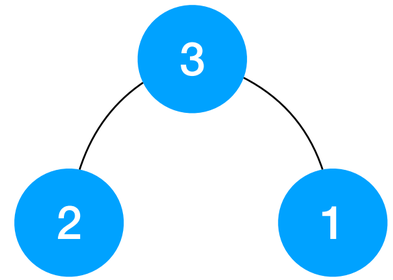
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		

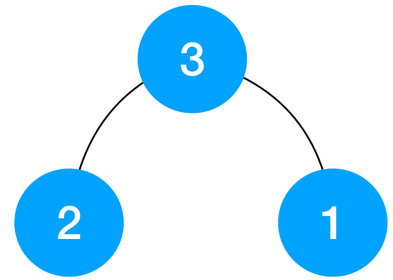
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		

Is this a BST?

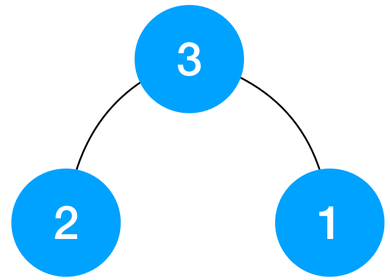



```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}
```

cursor	prev	left	right
3	NULL		
2	NULL		

```
node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

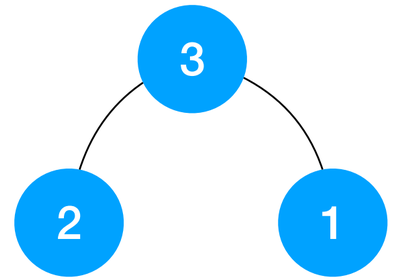
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		

Is this a BST?

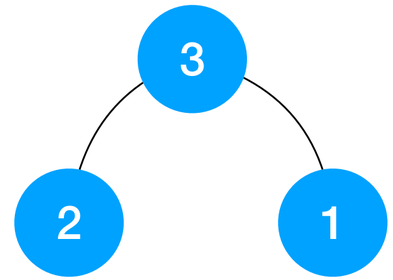



```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}
```

cursor	prev	left	right
3	NULL		
2	NULL		

```
node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

Is this a BST?

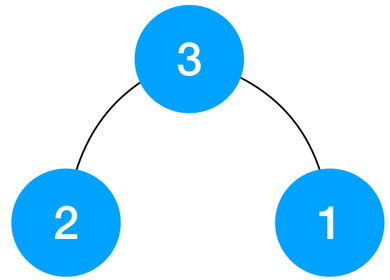



```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}
```

```
node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		
NULL			

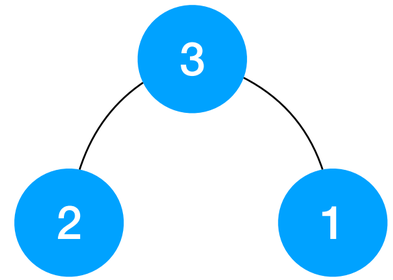
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		
NULL			

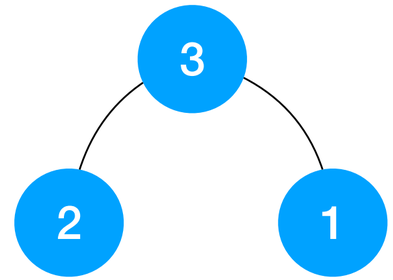
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		
NULL			

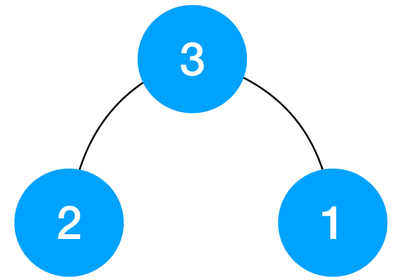
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		
NULL			

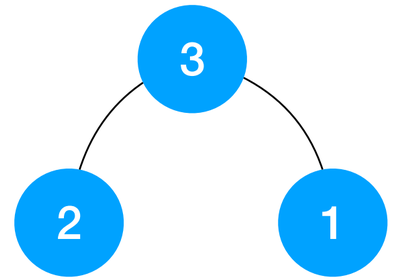
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL	true	
NULL			

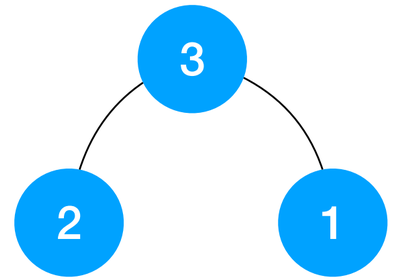
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL	true	
NULL			

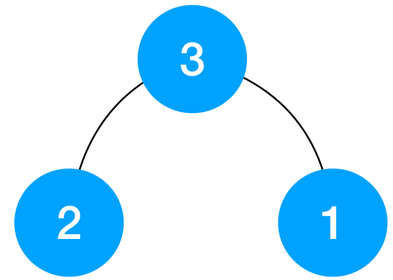
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
      return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL	true	
NULL			

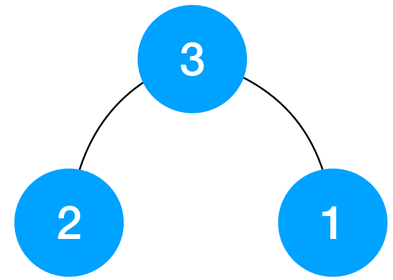
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
      return false;

→   prev = cursor;
    bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			

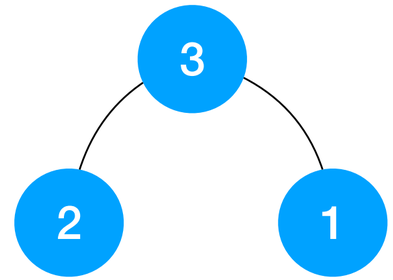
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
      return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			

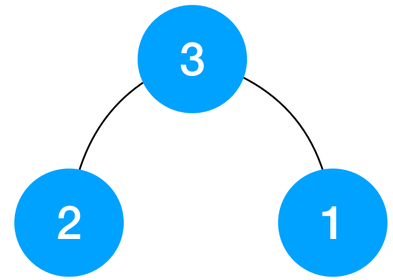
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			

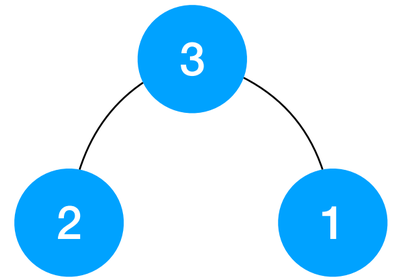
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			
NULL	2		

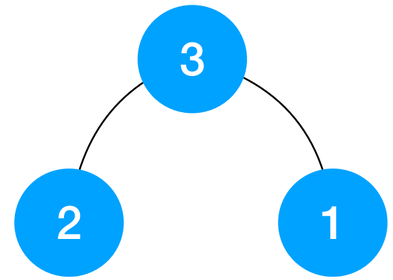
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			
NULL	2		

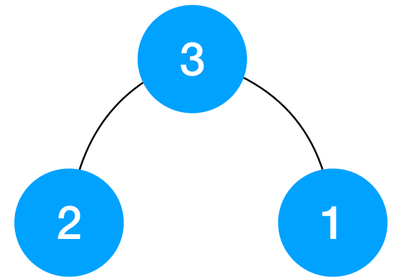
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
→  return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
     return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			
NULL	2		

Is this a BST?

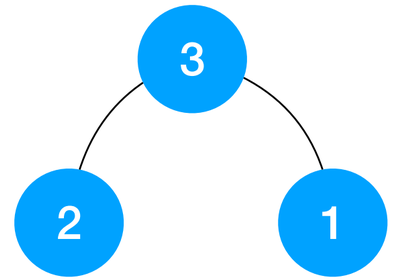



```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}
```

```
node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			
NULL	2		

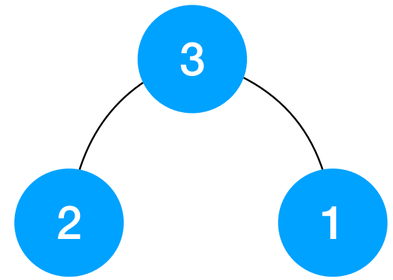
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
      return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	true
NULL			
NULL	2		

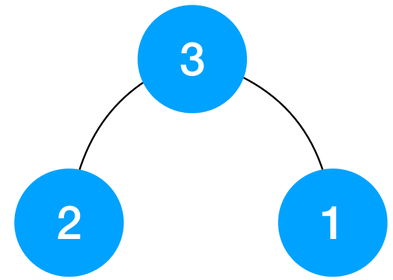
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
      return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

→   return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	true
NULL			
NULL	2		

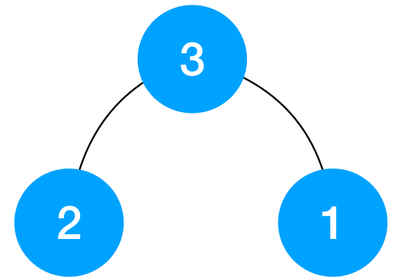
Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	true
NULL			
NULL	2		

Is this a BST?

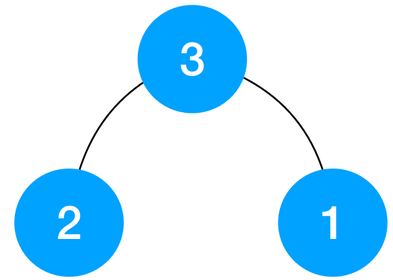



```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	true
NULL			
NULL	2		

Keep going … try it online: see Gitlab

Is this a BST?

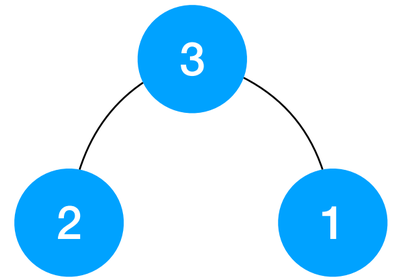



```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2	true	
2	2	true	true
NULL			
NULL	2		

Keep going … try it online: see Gitlab

Is this a BST?



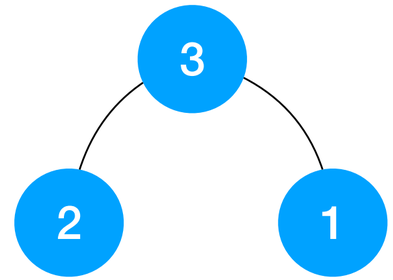

```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}
```

```
node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2	true	
2	2	true	true
NULL			
NULL	2		

Keep going … try it online: see Gitlab

Is this a BST?

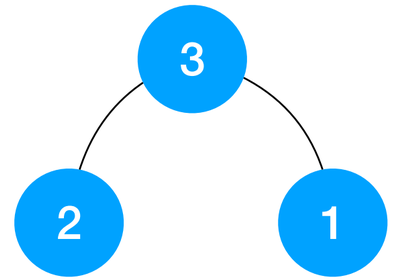



```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
      return false;

→   prev = cursor;
    bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2	true	
2	2	true	true
NULL			
NULL	2		

Keep going … try it online: see Gitlab

Is this a BST?

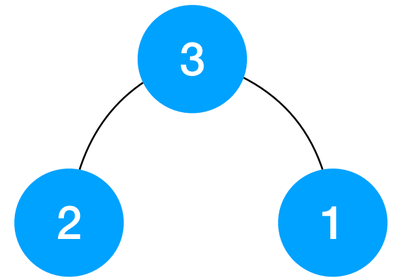



```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
      return false;

→   prev = cursor;
    bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	3	true	
2	3	true	true
NULL			
NULL	3		

Keep going … try it online: see Gitlab

Is this a BST?

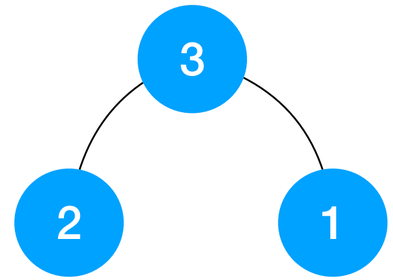



```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
      return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}


node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	3	true	
2	3	true	true
NULL			
NULL	3		

Keep going … try it online: see Gitlab

Is this a BST?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);
}
```

```
node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	3	true	
2	3	true	true
NULL			
NULL	3		

Keep going … try it online: see Gitlab

Other announcements

- Conflict exam policy (recap e-mails)
- HKN review session (1500 - 1730 hrs, 05/04 in ECEB 2013)
- Programming competition tomorrow
- Additional study material is on the course website
  - Practice, practice, practice!
- Exam format

Practice material

Let us do some posted practice material/questions …

We did CPP, FareySequence, BST to Doubly Linked List

https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24/-/tree/main/lec0430

Topics to review …

Part 1: LC-3

- Assembly language programming & process
- Memory-mapped I/O: input from keyboard, output to monitor
- TRAPs & Subroutines, Interrupts & Exceptions
- Stacks

Part 2: C

- Built-in data types, operators, scope
- Functions & run-time stack
- Pointers & arrays
- Recursion: searching, sorting, backtracking
- I/O: streams and buffers, read from / write to file
- User-defined data types: enum, struct, union
- Dynamic memory allocation
- Linked data structures: linked list (stack, queue) & trees

Part 3: C++

- Class (encapsulation, inheritance, abstraction)
- Pass by value /(const) reference / address
- Virtual function, operator overload, template (polymorphism)

Part 1 - LC3

- Address space: 216 locations, addressability: 16 bits
- General-purpose registers: R0, R1, ... R7
- Special-purpose register: PC, IR
- Input from keyboard: KBDR/KBSR
- Output to monitor: DDR/DSR
- Operate instructions: ADD, AND, NOT
- Data movement instructions: LD, LDI, LDR, LEA, ST, STR, STI
- Control instructions: BR, JSR/JSRR, JMP, RET, TRAP, RTI
- Condition codes: N (negative), Z (zero), P (positive)
- TRAPs: In, GETC, OUT, PUTS (uses R0; R7 is modified after call)
- Subroutines: callee-save vs. caller-save, nested subroutine needs to save R7
- Interrupts: external event, supervisor vs. user stack, RTI instruction
- Exceptions: internal event for handling errors
- Stack: FILO, overflow, underflow, R6 – stack ptr, R5 – frame ptr

Part 2 - C language

- Scope: local vs. global variables (determined by location of declaration)
- Storage class: static (retains value, global data area) vs. automatic (stack)
- Control structures: conditionals (if, if-else, switch); loops (for, while, do-while)
- Functions & run-time stack (C to LC-3)
- Pointer: address of a variable in memory
- Array: a list of values arranged sequentially in memory
- Pass by value vs. pass by reference (pointer)
- Pointer Array Duality (int array[10] = {1,2,3,4,5,6,7,8,9}; int *ptr = array;)
- Recursion: base case(s) & recursive case(s)
- File I/O: fopen, fclose, fscanf, fprintf
- Linked lists & trees (pointer, struct, dynamic memory allocation)

Part 3 - C++ language

- Class vs. struct: 4 features of OOP (polymorphism, inheritance, encapsulation, abstraction)
- Dynamic memory allocation: new & delete
- Basic I/O: std, cin, cout
- Pass by value vs. pass by address vs. pass by (const) reference
- Operator and function overloading
- Base class & derived Class: access identifier (public, protected, private)
- Virtual function & virtual function table: static vs. dynamic binding
- Function and class templates: separate type with container
- Big three: copy constructor (deep vs. shallow copy), destructor, copy assignment operator
- Implicit ‘this’ pointer: a pointer to the invoking object
- Vectors: dynamic arrays, elements are stored in consecutive locations
- Lists: doubly linked lists, elements are allocated individually
- Iterators: the mechanism used to minimize an algorithm’s dependency on the data structure on which it operates