Welcome back!

Data structures lab – week 3
Wake-up quiz

- Based on your intuition (or knowledge), which of the following statements is true about Linked Lists (LL) and binary search trees (BST):
  a) LLs have faster search time than BSTs
  b) BSTs have faster search time than LLs
  c) They have the same search time
- Correct answer is c.
  - After this class, you will know why.
Outline

- Last week
- Winter warmup comments
  - Hints for future success
- Pseudo-code to implementation-code
- Trees in the forest
- Assignment 2

By the way, did you know that C++ was originally invented by a Danish guy?
Week 2 recap

- How a lab lecture works
- Linked Lists
  - Revisited today
- Coding guidelines
  - Also revisited today
- Assignment questions
  - May be revisited today
    - But hopefully, you all did assignment 1 by now.
Week 2 class evaluation

- 100% increase in responses!
  - Up from 7 to 14
- Selected comments (slightly edited)
  - "Cover more material"
  - "Spend more time on projects"
  - "Eclipse is not everything"
  - "go ducks!!!"
- Full survey results online
Hints for success

• Hint number 1: Read the assignment
  • “You should conform **exactly** to the input and output specification.”
    – ”Let me say that again: conform **exactly** to the input and output specification”
      • This is from the website.
  • Many had extra stuff in there.
    – ”Please input a number”
    – ”Please input a name”
Hints for success

• Hint number 2: Look at your code

```cpp
#include <warmup.h> ← Not included
#include <iostream>
using namespace std;

int main() {
    std::cout << "I like to do more than necessary";
    return 0;
}
```

• What's wrong here?
Hints for succes

- Hint number 3: Comply with standards.
  - And knowledgeable people.
- Quote from the C++ FAQ Lite:
  - "main() must return int. Not void, not bool, not float. int. Just int, nothing but int, only int."
- With g++, "void main()" will not compile, "main()" will.
  - But that does not make it correct
Hints for success

• Hint number 4: Use large test cases
  `char[400][100]`
  – Stores 400 names of length 100
  – What's wrong with that?
• It is easier to catch errors like the above.
• It is easier to get a feel for running time.
Hints for success + prosperity

- Hint number 5: Use the terminal.
- Easier for testing large test cases
  - ./myProgram < largeTestCase > outputTestCase
- Eclipse can still be used as development environment, if you prefer.
- The terminal is powerful beyond C++!
Wake-up quiz – hints for success

• What was hint number 1?
  a) Use the terminal
  b) Use large test cases
  c) Comply with standards
  d) Look at your code
  e) Read the assignment

• e is correct but they are all important!
From pseudo to implementation

- Find an algorithm in pseudocode
- (Understand the algorithm)
- Implement the algorithm
- Wait, how do we do this again?
  - I don't know what pseudocode is
  - I don't know where to find the pseudocode
  - What about data structures?
- Well, listen closely
A pseudo stack

- I want to implement a stack and the elementary stack operations
- I look at chapter 10 in Cormen
  - This is where the pseudocode is.
- I see something that has line numbers and a different font than everything else
  - This is the pseudocode.
A pseudo stack

- Now, I have the pseudocode for
  - StackEmpty(S), checks if S is empty
  - Push(S,x), add element x on to S
  - Pop(S), remove the top element from S
- But what is S?
  - It depends on the situation.
  - It depends on what I need.
  - But I need to know at least the top element
A stack implementation

```c
struct element {  
  string name;  
  element * below_me;  
};

struct stack {  
  element * top;  
};
```
A stack implementation

```c
bool stackEmpty(stack& S) {
    if (S.top == NULL)
        return true;
    return false;
}

void push(stack& S, element& x) {
    x.below_me = S.top;
    S.top = &x;
}

element * pop(stack& S) {
    element * x;
    x = S.top;
    S.top = x->below_me;
    return x;
}
```
Wake-up quiz – stack 'em up

• A Linked List can represent both stacks and queues. Can stacks by themselves be used to represent a queue?
  a) yes, we need one stack to do it.
  b) yes, we need two stacks to do it.
  c) yes, we need three stacks to do it.
  d) no

• Correct answer is b.
A queue – stack style

- I want to implement a queue using two stacks.
- I already have a stack data structure.
- I need the pseudocode for the queue operations.
  - I don't have this.
    - Yay, I get to solve a problem.
A queue – stack style

- **EnQueue(Q,x)**
  - Just push x onto *head*.

- **DeQueue(Q)**
  - If *tail* is empty
    - Loop until *head* is empty
      - Pop element x from *head*
      - Push element x to *tail*
  - Return Pop of *tail*
A queue – stack style

• How does the data structure look?

```c
struct queue {
    stack head;
    stack tail;
};
```
A queue – stack style

```c
void enQueue(queue& Q, element& x) {
    push(Q.head, x);
}

element * deQueue(queue& Q) {
    if (stackEmpty(Q.tail)) {
        while (!stackEmpty(Q.head)) {
            element * x;
            x = pop(Q.head);
            push(Q.tail, *x);
        }
    }
    return pop(Q.tail);
}
```
From pseudo to implementation

- Look in the book
- Identify the important features of a data structure or choose existing one
- Try to map the pseudocode to the programming language
  - Pseudocode does not know the difference between a pointer and a reference
    - (Do you?)
Trees in the forest

This is a binary search tree
Binary search trees

- Every node has at most 2 children.
- Every node consists of:
  - A key
  - A pointer to the left child, *left*
  - A pointer to the right child, *right*
  - A pointer to the parent, *p*
Binary search trees

- Every node \( x \) satisfies the *binary-search-tree-property (bstp)*:
  - For every node \( y \) in the left subtree of \( x \):
    - \( y.key \leq x.key \)
  - For every node \( y \) in the right subtree of \( x \):
    - \( y.key \geq x.key \)
Binary search tree

• Often used for search
• Because of the bstp, searching for a value $k$ is pretty straightforward
  – Start at the root $r$
  – If $k < r.key$
    • Go left
  – Else
    • Go right
• Insertion is similar
Tree Search
Wake-up quiz – BSTs

- A BST $T$ has $n$ nodes and height $h$
- What is the running time of tree-search?
  a) $O(\lg n)$
  b) $O(h)$
  c) $O(n)$
  d) $O(n^2)$
- Correct answer is b
  - For a complete binary tree, $h = \lg n$
Assignment 2

- Implement a binary search tree data structure.
  - Support insert and search
  - Do not bother about deletion
- Expand your linked list from A1 to include searching
- Compare running time for search with BST and LL.
Assignment 2

- Back to the first warm-up question
- What happens if the BST is just one big line of nodes?
  - LL and BST running time is the same!
  - $O(n)$
  - How can we deal with this?
Assignment 2

- We want the BST to have height $\lg n$
- We can balance the tree
  - coming up later in the term
- We can randomize the insertion
  - You will do this in the assignment.
  - Hopefully, this will lead to a performance boost.
    - The book says it does.
    - You should measure it.
Programming help

- Every Monday, 5pm-6pm
- Deschutes 100.
- Things you should do:
  - Read the assignment beforehand
  - Have specific questions
  - Try on your own before asking
Grades

• You have each been assigned a "secret" number, avoiding the use of student ids.
• As an extra bonus, you will have to figure out the secret number yourself. Here's how:
  - Create a C++ program and:
    • Use your student id as the "seed" for the standard random number generator
    • Read the random number
• Grades will be posted later today
Thank you

Questions?