
CS11 – Introduction to C++

Winter 2011-2012

Lecture 5

Today's Topics

- Lab 4b: Sparse-vector arithmetic
- `gdb` – the GNU Debugger



This Week's Homework

- Complete the **SparseVector** functionality
 - Equality/inequality operations
 - Should be straightforward
 - Vector addition/subtraction
 - Should be... *possible?!*
-

Comparing Linked Lists

- Similar pattern to before
 - Must iterate over both lists at same time
 - The lists are the same if all nodes are the same
 - Two main criteria:
 - If two nodes being compared have different indexes, or different values, the lists are different.
 - If one list ends before the other, the lists are different.
-

Efficient Addition and Subtraction

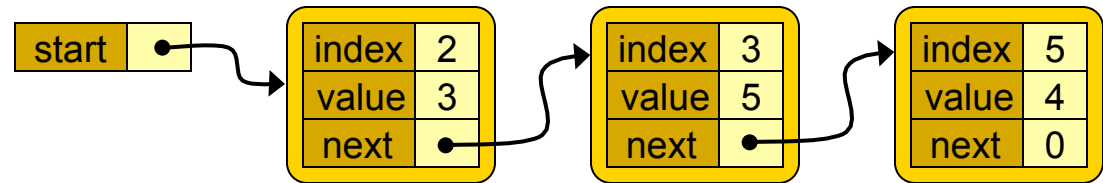
(Sparse Vector Style)

- Sparse vectors can be added and subtracted
 - Implementation should be efficient
 - Time proportional to number of non-zero values
- Concept:
 - Implement as $\mathbf{a} += \mathbf{b}$ (only one vector changes)
 - Traverse both vectors' node-lists simultaneously, calculating the result as you go
 - Only works if the lists are ordered by index!
 - This approach is *mostly* straightforward
 - There are other approaches, but all similar complexity

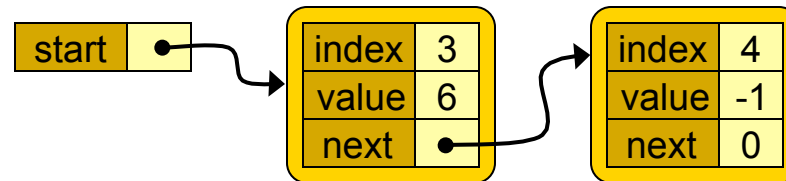
Efficient Adding

- Add these two vectors together

this:

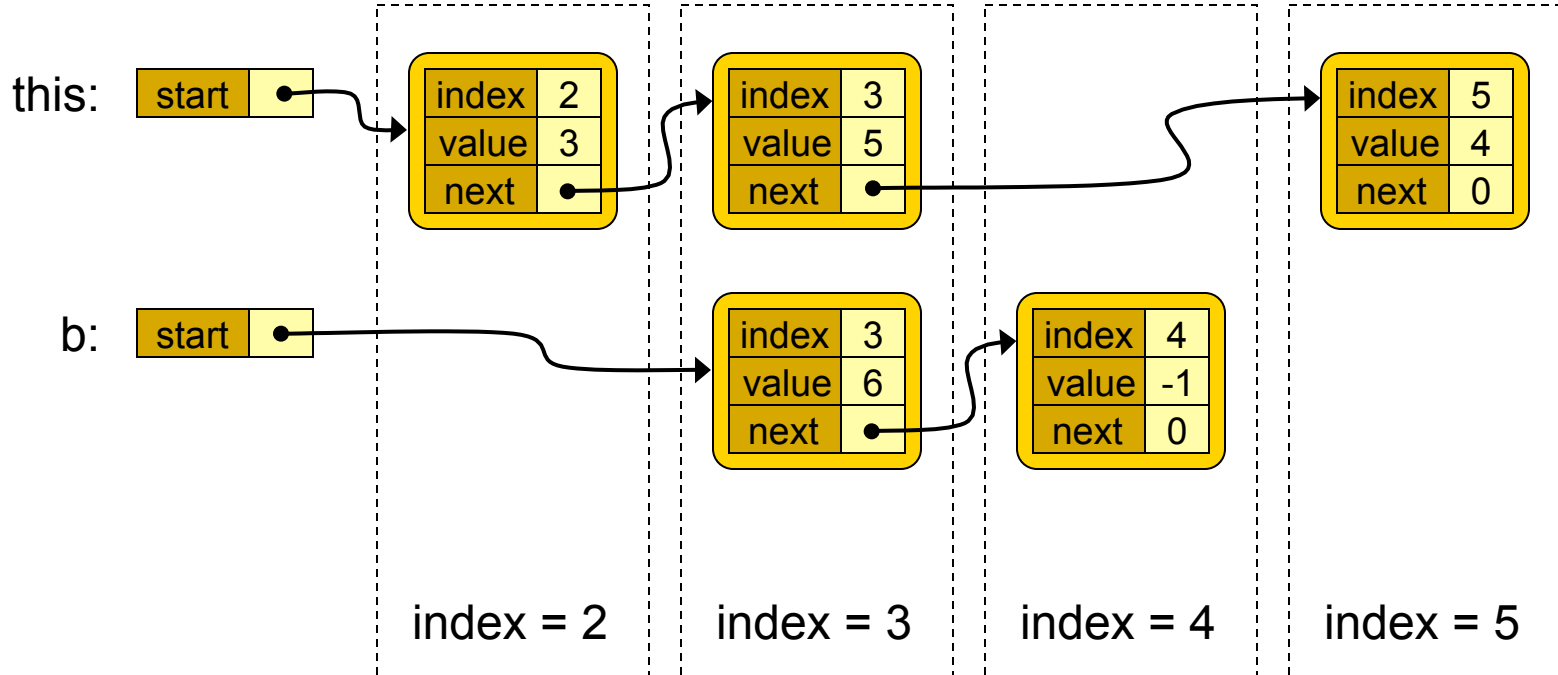


b:



Efficient Adding (2)

- Each loop iteration handles the next index-value separately



Vector Arithmetic

- Traverse both vectors in a single loop

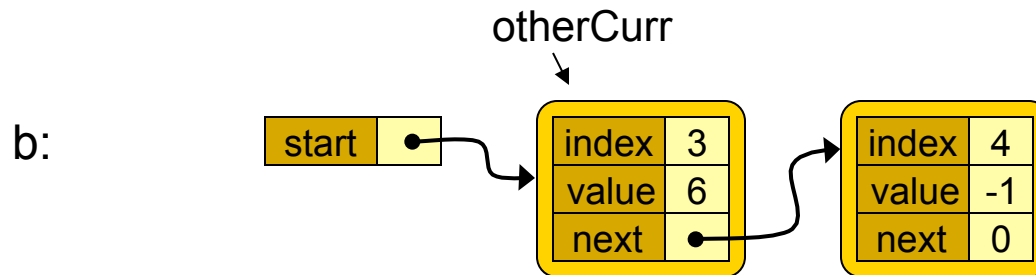
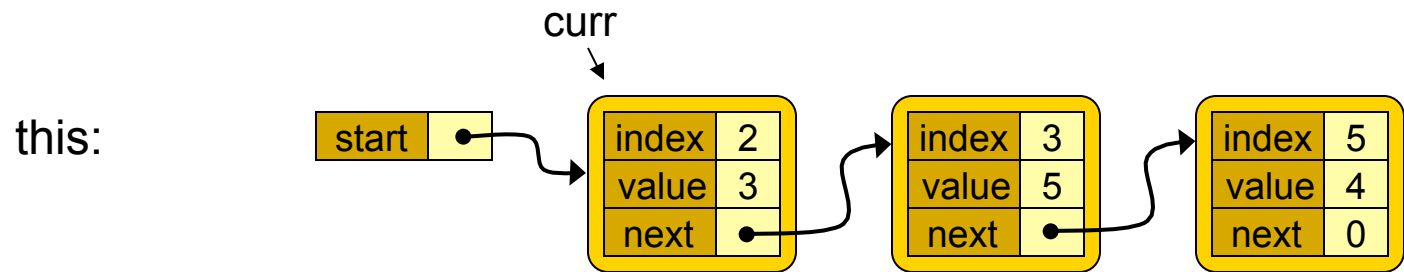
```
node *curr = start;
node *otherCurr = other.start;

while (curr != 0 && otherCurr != 0) {
    ...
}
...
```

- For each loop iteration:
 - Perform addition/subtraction of element at index i
 - $i = \min(\text{curr} \rightarrow \text{index}, \text{otherCurr} \rightarrow \text{index})$

Efficient Adding (3)

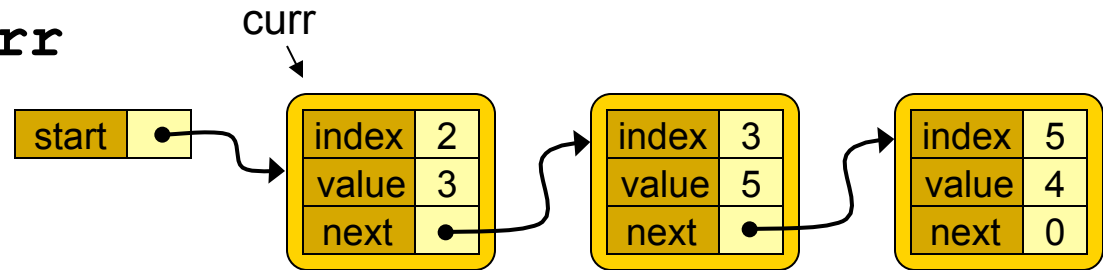
- Start iterating over the list.



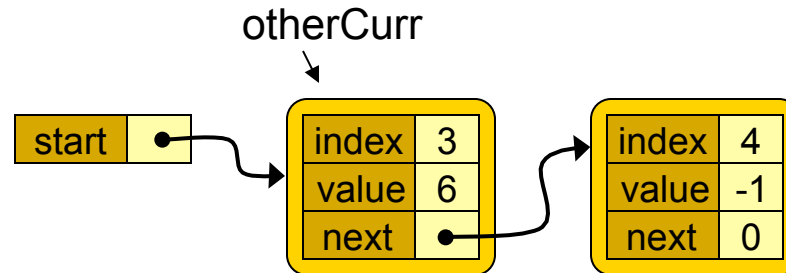
Efficient Adding (4)

- **b** doesn't have a value for index 2, so it's 0.
 - $3 + 0 = 3$
 - Advance **curr**

this:



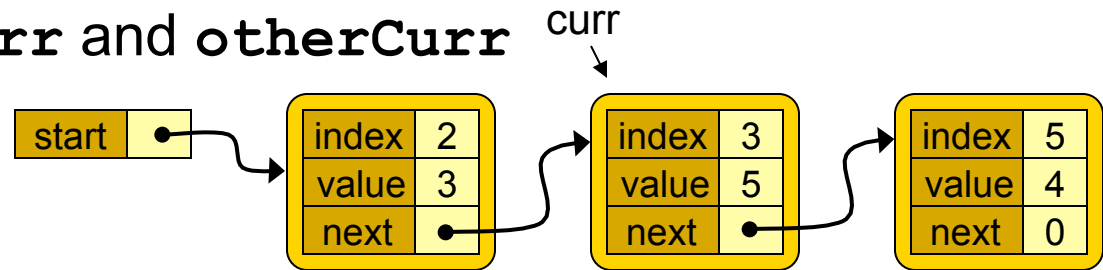
b:



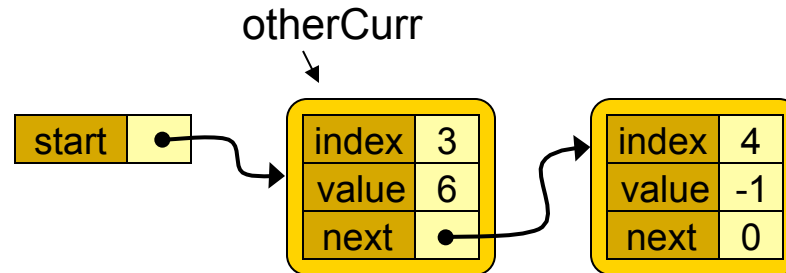
Efficient Adding (5)

- Both vectors have a value for index 3.
 - $5 + 6 = 11$
 - Advance **curr** and **otherCurr**

this:



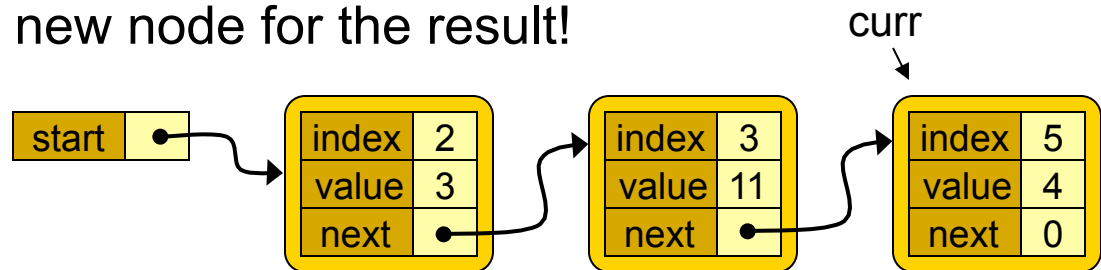
b:



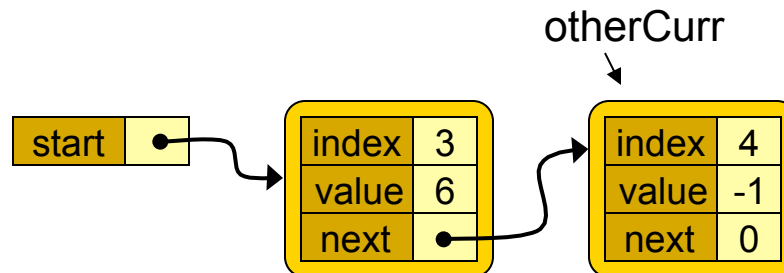
Efficient Adding (6)

- Only **b** has a value for index 4.
 - $0 + (-1) = -1$
 - Must create a new node for the result!

this:

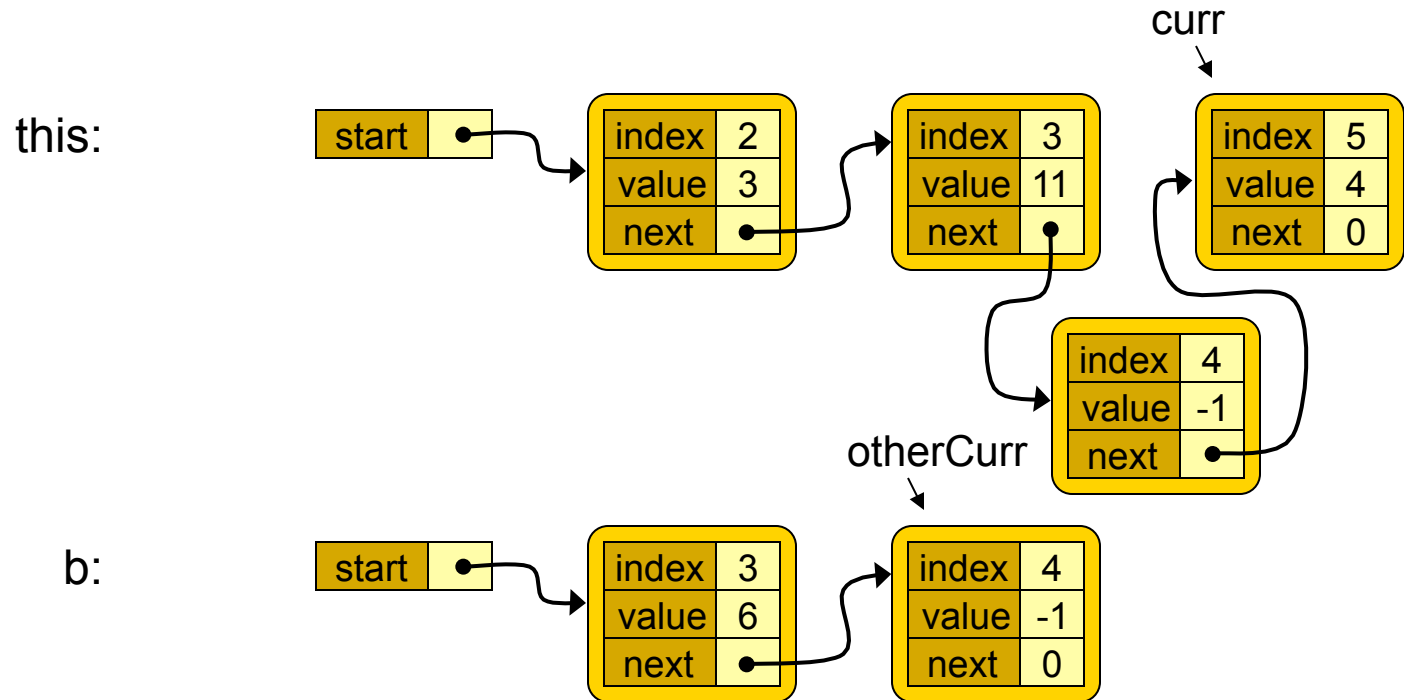


b:



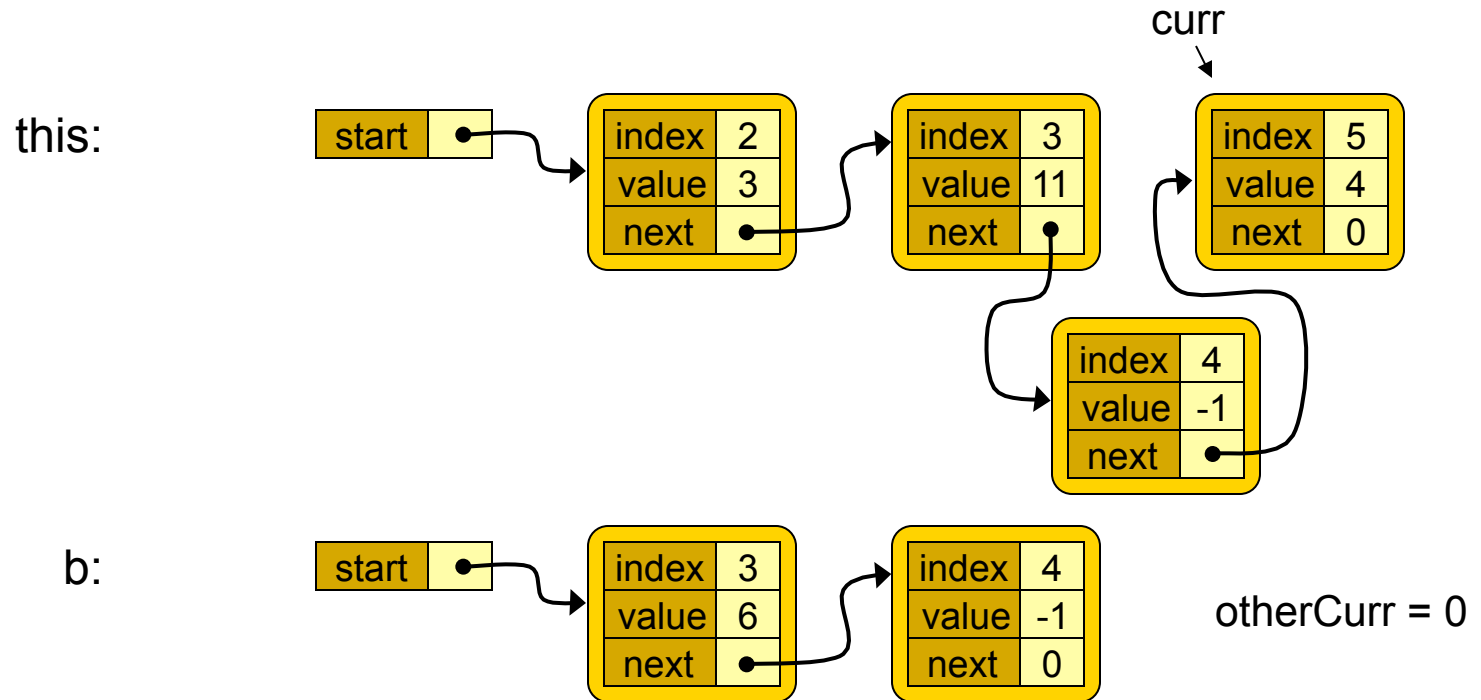
Efficient Adding (7)

- New node is *before* **curr**
- Now increment **otherCurr**



Efficient Adding (8)

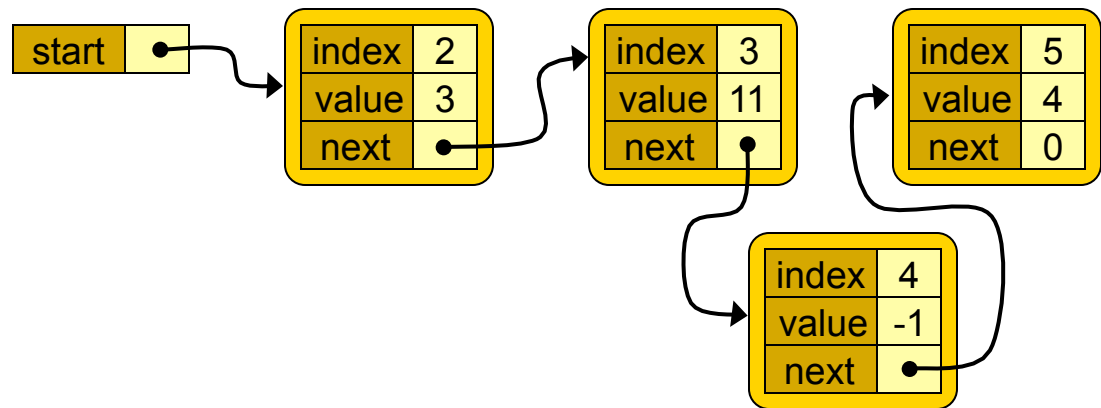
- Reached end of **b**, so we're done.



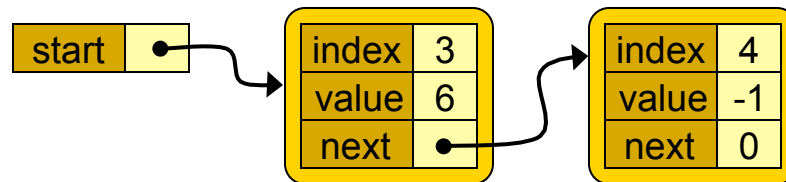
Efficient Adding

- Final result:

this:



b:



Vector Arithmetic ⁽¹⁾

- Our loop has three cases to handle:
 1. My node's index is less than other node's index
 - Can assume that corresponding value in other list is zero!
 - Nothing to add/subtract – my node's value doesn't change.
 - Move my node-pointer forward down the list
 2. My node's index is equal to other node's index
 - Perform addition/subtraction operation with the two values
 - Store result into my node
 - Move *both* node-pointers forward down the list
 3. My node's index is greater than other node's index
 - (My node's index = i , other node's index = j)
 - My value for index j is zero, but the result will be nonzero!
 - Add a *new* node to my list, for index j , containing result
 - Move other node-pointer forward down the list

Vector Arithmetic (2)

- Case 3 is the tricky one!
 - Add a new node to the list, right before **curr**
 - Need a **prev** node-pointer that chases **curr** through the list
 - Make sure to update **prev** properly, when a new node is added!
 - **curr** doesn't move...
 - ...but there's a new "previous node!"
 - Set **prev** to be the newly added node
-

Vector Arithmetic (3)

- Finally: What happens at end of loop?
 - Again, three cases
 1. Reached end of both lists at the same time
 - This is easy! Do nothing.
 2. Reached end of other list first
 - Also easy, since rest of elements in other list are zero.
 - Do nothing.
 3. Reached end of my list first
 - `curr == 0`, but `otherCurr != 0`
 - Copy remaining nodes in other list to end of my list
 - (Of course, make sure they reflect sum or difference...)
 - Again, `prev` comes in handy!

Other Notes about Efficient Adding

- Adding/subtracting vectors can produce nodes with value = 0
 - Vector 1: (0, 3, 5, 2, 0, 0)
 - Vector 2: (0, -3, 4, -2, 0, 5)
 - Sum is: (0, 0, 9, 0, 0, 5)
- An easy approach:
 - Don't worry about culling those out *during* the addition/subtraction itself
 - Can write a helper-fn that removes zero-value nodes, and call it at end of add/subtract operation

Zero-Value Nodes (2)

- Another approach:
 - Vector 1: (0, 3, 5, 2, 0, 0)
 - Vector 2: (0, -3, 4, -2, 0, 5)
 - Sum is: (0, 0, 9, 0, 0, 5)
- A zero-result is only generated when both inputs are nonzero
 - Exactly one of our cases can produce a zero-value node (second case, slide 16)
 - Already have a “previous node” pointer for adding in new nodes... could handle in main-loop too

Finding Bugs

- Several ways to find bugs
 - Use assertions to “catch” them
 - Print out info as your program runs (logging)
 - Use a debugger to watch your program execute
 - These approaches are complementary
 - Each has different strengths/weaknesses
 - Lab 4 is a great opportunity to practice your debugging skills. 😊
-

The Debugger: `gdb`

- `gdb` is the GNU Debugger
 - Plays very well with `gcc` and `g++`
 - *Very* sophisticated debugging features
 - Can step through your program line by line
 - Can set breakpoints in your program
 - Tell program to “break” (stop) when it hits a certain point
 - Can examine or modify your program’s data
 - *Many* more features too...
 - Interface is command-line only!
 - Several front-ends have been built onto `gdb`
-

Preparing to Debug

- **`gdb`** needs extra information about your code
 - Called “debug symbols”
- Compiler usually leaves this info out
 - Takes up extra space...
- Also want to turn off all optimizations!
 - Compiler may reorder your code to make it faster
- Compiler arguments for debugging:
 - `g++ -Wall -O0 -g source.cc -o execfile`
 - `-g` means “include debug symbols”
 - `-O0` means “don’t optimize anything”

Starting the Debugger

- You start your program in the debugger:
 - `gdb checksv`
 - Debugger loads your program
 - Reads the debug information out of the file
 - Debugger doesn't automatically run your program
 - Gives you a chance to set breakpoints, specify command-line arguments, etc.
 - To start your program:
 - `run` (no command-line args)
 - `run arg1 arg2 ...` (to specify command-line args)
-

Breakpoints

- Setting breakpoints is easy:
 - `break filename:line`
 - `break function`
 - `break classname::function`
 - Can use tab-completion with classes/functions!
- Can only set breakpoints when program is stopped
- Example 1: Debug the destructor

```
break 'SparseVector::~~SparseVector()'
```
- Example 2: Debug the copy-constructor

```
break 'SparseVector::SparseVector(SparseVector const&)'
```

 - Use tab-completion for this kind of thing

When the Breakpoint Is Hit

- Debugger stops when it hits any breakpoint
 - `list` - Lists the code
 - `where` - Shows the call-stack
 - `step` - Steps to next line of code
 - (Steps *into* function calls)
 - `next` - Steps to next line of code
 - (Steps *over* function calls)
 - `cont` - Continue running!
 - For help on commands:
 - `help` or `help command`
 - To end the madness:
 - `quit`
-

Examining Variables

- Use `print` to examine variables
 - `print` understands `*`, `->`, `&`, etc.
 - Example: debugging your list code
 - `break SparseVector::getElem`
 - `run`
 - `...`
 - `print curr` Shows pointer value
 - `print *curr` Shows node's contents
 - `print curr->next` Shows next pointer value
 - `print *(curr->next)` Shows next node
-

Debugging Crashes!

- If your program crashes when you run it:
 - Compile with debug symbols & no optimizations
 - Run it in the debugger
 - Debugger will tell you that it has crashed
 - Use **where** to find out where crash is occurring
 - Figure out where to set breakpoints, then restart
 - **run** command restarts your program at the beginning
-

Next Week!

- C++ templates
- Using exceptions to report errors

