AMSC/CMSC 662
Computer Organization and Programming for Scientific Computing
Fall 2011
Data Structures in Scientific Computing
Dianne P. O'Leary

Data Structures in Scientific Computing

- You are probably familiar with several data types already:
  - integer variables
  - floating point variables
  - strings
  - arrays

- These take us a long way in scientific computing, but we can do better if we know more:
  - pointers
  - structures (cell arrays)
  - stacks
  - queues
  - linked lists
  - trees
  - hash tables
  - heaps
  - quadtrees
- We'll discuss what these are and why they are useful.

Organization ... or lack thereof

For each of these data structures, we need to know:

- What it is.
- How to insert, delete, and perform other operations on it.
- What it is good for.

I believe

- You should memorize what it is.
- You should be able to derive, from the definition, how to insert, delete, and perform other operations on it.
- You should be able, from the definitions, to choose a good structure for a given application.

So these notes are sketchy. Rather than reading, it is important to work through the operations and the applications, so you need to complete the notes yourself.

The Plan

- First: pointers, which make it all possible.
- Second: cell arrays, a MATLAB data structure, and why it is not enough.
- Third: other data structures, operations on them, and what they are good for.

In the process, get into the habit of routinely researching topics on the web and filling in the missing pieces yourself, and forever abandon the expectation that all the information you need will be provided to you.

Pointers

- A pointer contains a memory address.
- It points to a memory location in a computer.
- "Address" can be understood more or less literally.
- MATLAB pointers are initialized with `libpointer`, but pointers are much more useful in other programming languages.
- We'll use C notation for pointers: the address of a variable named `merlin` is `&merlin` and if `ptr` is a pointer to a memory location, then `*ptr` refers to the contents of that memory location.
- We'll see more examples when we discuss linked lists.

Three examples for which pointers are useful

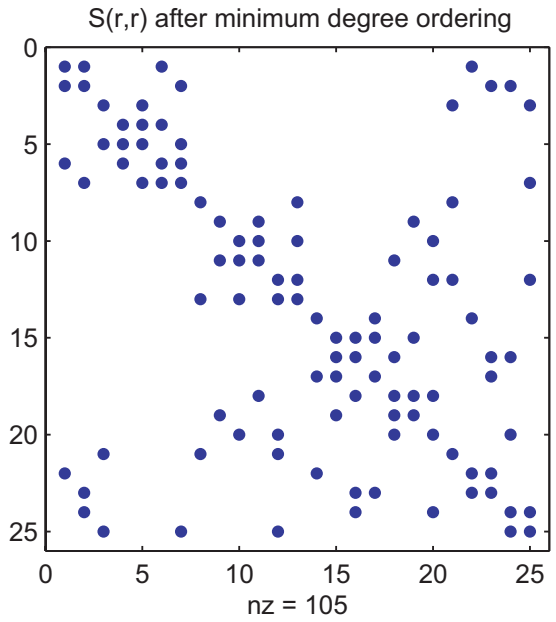
Example 1: Suppose I want to organize my address book:

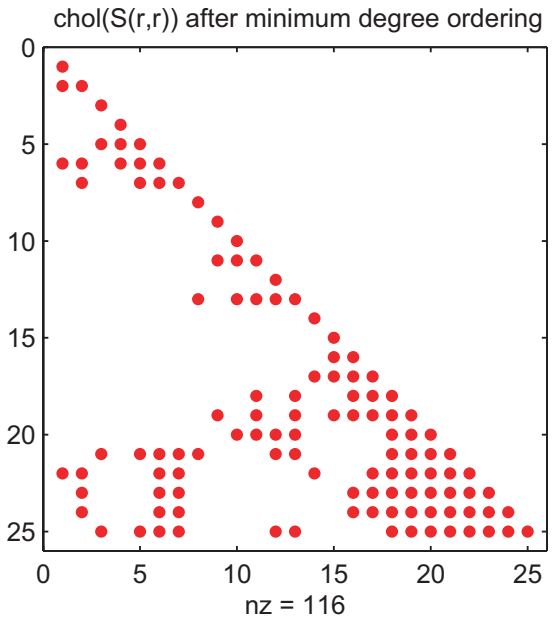
Day	Dorothy	(312) 555-9180	dorothy@hullhouse.net
Falk	Peter	(323) 555-9882	columbo@hollywood.com
Lincoln	Abraham	(217) 555-8923	honest-abe@verity.net
Wilder	Laura	(952) 555-8923	sis@LittleHouse.net

What functions do I need? Examples:

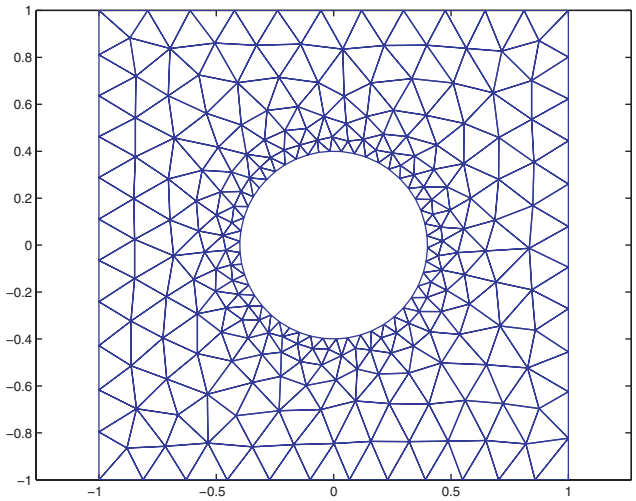
- Find Peter's email address or telephone number.
- Add an entry for Jim.
- Delete the entry for Lincoln.
- Update the address or telephone number for Dorothy.
- Add a url for each person.
- Sort the list by first name.

Example 2: How can I efficiently store a sparse matrix that I need to factor?





Example 3: How can I efficiently store a finite element grid that might need to be refined?



We'll work with Example 1 in these notes and Example 2 in a later class period.

Important to keep in mind: Part 1

MATLAB makes updating a data structure look easy.

For example, we can just define a 10×5 matrix this way:

```
for i=1:10,
    for j=1:5,
        A(i,j) = i+j;
    end
end
```

But this hides the fact that the array A is recopied into a larger space (at least) 10 times, every time the value of i changes.

We could have avoided this by inserting the statement `A = zeros(10,5)` before this code.

When we talk about efficient data structures for a particular application, we want to avoid hidden as well as obvious inefficiencies.

Important to keep in mind: Part 2

Memory hardware is arranged as a sequence of storage locations each having an address like 0000, 0001, 0002, ...

If we want a segment of memory to "behave" in a more complicated way, then this has to be built into the software.

This is how we handle the storage of A(1,1), A(2,1), ... A(10,1), A(1,2), etc.,

So when we ask MATLAB for the value of A(i,j), MATLAB calculates that we want the storage location that is (j-1)*10 + (i-1) double precision words after the memory location assigned to A(1,1).

If we add a row to A, MATLAB has to put A(11,1) right after A(10,1), so the array has to be rearranged and put in a space big enough for it.

So are arrays well suited to our address book?

No.

Cell arrays

Matlab provides cell arrays to overcome some of the difficulties.

```
A(1).lastname = 'Day';
A(1).firstname = 'Dorothy';
A(1).phone = 3125559180;
A(1).email = 'dorothy@hullhouse.net';
A(2).lastname = 'Falk';
```

etc.

How well does this serve our address book needs?

We need other alternatives.

Stacks

http://en.wikipedia.org/wiki/Stack_(data_structure)

Queues

http://en.wikipedia.org/wiki/Queue_(data_structure)

Linked lists

http://en.wikipedia.org/wiki/Linked_list

Trees

http://en.wikipedia.org/wiki/Tree_%28data_structure%29

Hash tables

http://en.wikipedia.org/wiki/Hash_table

Heaps

http://en.wikipedia.org/wiki/Heap_(data_structure)

Quadtrees

http://en.wikipedia.org/wiki/Quadtree