Maintaining order

Sorting and searching

CS112 Scientific Computation
Department of Computer Science
Wellesley College

A small student database

ID#	First	Last	Year	Major	GPA
1	Jane	Williams	2007	BISC	3.3
2	Ann	Smith	2009	CS	2.8
3	Susan	Jones	2008	ARTS	3.1
4	Claire	French	2007	ENG	2.7
5	Amy	Clark	2009	CHIN	3.5
6	Brenda	Thomas	2007	ECON	2.5
...	...	...	...	...	...

Some questions:

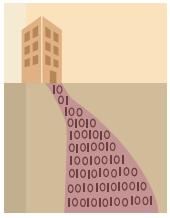
1. How many ECON majors are in the class of 2009?
2. What are the names of all the ARTS majors?
3. What is the average GPA of CHEM majors in the class of 2007?

Managing databases

- Database management focuses on developing ways to organize and maintain large amounts of data, ...
- ... and ways to extract information from this data

Common database operations:

- Select a subset of the data that satisfies certain criteria
- Summarize the data, or some portion of it



Representing table of student data

- The table of student data has the following organization:

```
{{1 'Jane' 'Williams' 2006 'BISC' 3.3}
 {2 'Ann' 'Smith' 2008 'CS' 2.8}
 ... }
```

Selecting data from studentData table

Select students from the class of 2010:

```
function newTable = selectFromTable1 (table)
% constructs a new table that only contains data for
% students from the class of 2010
newTable = { };
for index = 1:length(table)
   % preserve items whose year is 2010
   if (table{index}{4} == 2010)
      newTable{end+1,1} = table{index};
   end
end
```

Time-out exercises

```
function newTable = selectFromTable1 (table)
% constructs a new table that only contains data for
% students from the class of 2010
newTable = { };
for index = 1:length(table)
   % preserve items whose year is 2010
   if (table{index}{4} == 2010)
      newTable{end+1,1} = table{index};
   end
end
```

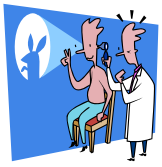
(1) Modify selectFromTable1 to return a table that only contains data for students who are CS majors

(2) Modify selectFromTable1 to return a table that only contains data for students who who have a GPA that is at least 3.0

Projecting fields of data

Another common database operation, called projection, selects one or more fields (columns) of data

```
>> majors = projectfromTable (studentData, fields, 'major')
majors =
  'BISC'
  'CS'
  'ARTS'
  'ENG'
  ...
```



Code for projectFromTable

```
function projection = projectFromTable (table, fields, fieldName)
% extracts a single field of data from the input table, given a cell
% array of all field names and desired single field name as input

% determine column index from field name
fieldIndex = find(strcmp(fields, fieldName));
numItems = length(table);          % determine number of items
projection = cell(numItems, 1);    % create empty cell array
for index = 1:numItems             % store items in projection
   projection{index} = table{index}{fieldIndex};
end
```

How do you search a phonebook?

- If you expect to search for values in certain field a lot, it pays to order the data, either alphabetically or numerically
- But, how can we sort the items in a table in the first place?
- And, once the items are sorted, how can we search for a particular value, in an efficient way?

MATLAB sort function

```
>> nums = [7 2 9 7 8 3 6 1 3 4];
>> sortNums = sort(nums)
sortNums =
   1  2  3  3  4  6  7  7  8  9
>> sortNums = sort(nums, 'descend')
sortNums =
   9  8  7  7  6  4  3  3  2  1
>> [sortNums sortIndices] = sort(nums, 'ascend')
sortNums =
   1  2  3  3  4  6  7  7  8  9
sortIndices =
   8  2  6  9  10  7  1  4  5  3
```

Exercise: What does the expression nums(sortIndices) return?

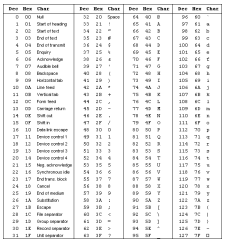
Now let's sort a cell array of strings

```
>> words = {'early' 'cloud' 'heights' 'a' 'black' 'great' 'from' 'descended'};
>> sortWords = sort(words)
sortWords =
  'a'  'black'  'cloud'  'descended'  'early'  'from'  'great'  'heights'
>> words = {'early' 'Cloud' 'heights' 'A' 'black' 'Great' 'from' 'Descended'};
>> [sortWords sortIndices] = sort(words)
sortWords =
  'A'  'Cloud'  'Descended'  'Great'  'black'  'early'  'from'  'heights'
sortIndices =
   4  2  8  6  5  1  7  3
```

Hmmmm…. What's going on here???

ASCII code (again)

- When comparing the order of two strings MATLAB uses the order of characters in the ASCII code in which *all capital letters appear before all lowercase letters*
- Exercise: Write a function that sorts a cell array of words alphabetically, independent of capitalization



Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Sorting the table of student data

- Recall that the table of student data has the following organization:

```
{{1  'Jane'  'Williams'  2006  'BISC'  3.3}
 {2  'Ann'  'Smith'  2008  'CS'  2.8}
 ...  }
```

- How can we create a new table in which the students' last names are ordered *alphabetically?*

```
{{9  'Joanne'  'Benton'  2009  'ND'  3.3}
 {11  'Sally'  'Blake'  2009  'ND'  3.1}
 {10  'Kate'  'Bower'  2006  'NEUR'  3.4}
 {5  'Amy'  'Clark'  2008  'CHIN'  3.5}
 ...  }
```


First consider a simpler problem

- girls =
  { {5 'Sally'}
    {3 'Jane'}
    {6 'Mary'} }

- sortedGirls =
  { {3 'Jane'}
    {6 'Mary'}
    {5 'Sally'} }

Step 1. Write two statements that produce the following cell array from girls:

```
names =
    {'sally'
     'jane'
     'mary'}
```

Step 2. Write two statements that produce sortedGirls from names and girls

sortByField

```
function newData = sortByField(data, fields, fieldName, isString)
fieldValues = projectFromTable(data, fields, fieldName);
if isString
   values = lower(fieldValues);
else
   values = zeros(length(table), 1);
   for i = 1:length(table)
      values(i) = fieldValues{i};
   end
end
[sortValues sortIndices] = sort(values);
newData = data(sortIndices);
```

get cell array of values with given field name

convert strings to lower case

move numbers from cell array to vector

sort values, save indices

put original values in sorted order

The search is on ...

- Now that we have a field of the table in sorted order, how can we search this field for a particular value, in an efficient way?

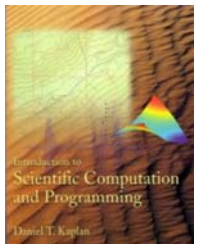
Let's play a game . . .

I'm thinking of a number between 1 and 100

What is it?

An auxiliary function

- To search a cell array for a particular string, we need a way to compare two strings to see which comes first
- Kaplan provides a function alphabetcompare for this purpose



searchTableBinary

```
function newTable = searchTableBinary (table, fields, fieldName, …
                                                    fieldValue, isString)
% given input table of sorted values in fieldName, searches for all items with
% a value in this field given by fieldValue. isString should be 1 for string values
% and 0 for numbers. uses the binary search strategy

newTable = {};

fieldIndex = find(strcmp(fields, fieldName));

if isString
   compareFunction = 'stringCompare';
else
   compareFunction = 'numCompare';
end

…
```


Auxiliary functions

```
function compare = stringCompare (item, index, value)
% returns 1 if value is equal to item{index}, 2 if value is later
% in the alphabet, and 3 if value is earlier in the alphabet
compare = alphabetcompare(lower(item{index}), lower(value));

function compare = numCompare (item, index, value)
% returns 1 if value is equal to item{index}, 2 if value is larger
% than item{index} and 3 if value is smaller
if (item{index} == value)
   compare = 1;
elseif (item{index} < value)
   compare = 2;
else
   compare = 3;
end
```


searchTableBinary – continued…

```
function newTable = searchTableBinary (table, fields, fieldName, …
                                                    fieldValue, isString)
  …
first = 1;                    % set initial region of table to search
last = length(table);

while (last-first) > 1
   middle = floor((first+last)/2)      % calculate middle index
   val = feval(compareFunction, table{middle}, fieldIndex, fieldValue)
   if (val == 1 )              % fieldValue is found, so add item to table
         newTable{end+1,1} = table{middle};
         break;
   elseif (val == 2)           % fieldValue is at an index larger than middle
         first = middle;
   else                        % fieldValue is at an index smaller than middle
         last = middle;
   end
end
```

searchTableBinary – continued…

```
function newTable = searchTableBinary (table, fields, fieldName, …
                                                    fieldValue, isString)
% if one occurrence of fieldValue was found, search above and below
% this item for other items with the same fieldValue

if (length(newTable) >= 1)
   index = middle - 1;       % check smaller indices
   while (index > 0) & …
         (feval(compareFunction, table{index}, fieldIndex, fieldValue) == 1)
      newTable{end+1, 1} = table{index};
      index = index - 1;
   end
   index = middle + 1;       % check larger indices
   while (index <= length(table)) & …
         (feval(compareFunction, table{index}, fieldIndex, fieldValue) == 1)
      newTable{end+1, 1} = table{index};
      index = index + 1;
   end
end
```