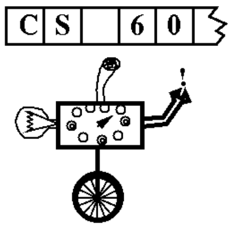
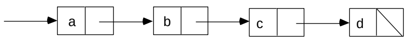


Information and Data Structures

September 7, 2001

Information Structures vs. Data Structures

- Data structures are built out of blocks of computer memory and references/pointers between them.
- Information structures are *abstractions* of data structures.
- Example: A “list”:
  - As a data structure, could be a linked list:



or it could be an array:

a	b	c	d

to give a few of the possibilities.

List Abstraction

- In an abstract sense, what matters most is the ***order*** of the elements in the list.
- We often don't even care how the list is represented in the machine!

List Structures

- We can just agree on some ***presentation*** or ***notation*** that shows this order, e.g.

`[a, b, c, d]`

- The notation resembles that used for sets

{2, 3, 5, 7}

- except that:
  - Order matters with lists; it doesn’t for sets.
  - Duplication matters in lists; it doesn’t for sets.

Representations Using Lists

- Pairs:
  `[1, 2]   [3, 4]   [5.0, 6.0]`
- Triples:
  `[1, 2, 3]    ["a", "b", "c"]`
- n-tuples:
  $[x_1, x_2, x_3, \ldots, x_n]$

Homogeneity

- In most of the examples we will look at, our lists will be homogeneous
  - All the elements have the same "type"
- We will also permit mixed lists

`[3, "Helium", 5.0]`

Lists of Lists

- We can even have lists with lists as elements

`[[1, 2], [3, 4], [5, 6]]`

`[[1, 2, 3], [4, 5, 6]]`

`[[1, 2, 3], [2, 3], [3], []]`

- Or lists of lists of lists, etc.

Length of a List

- The length, or number of elements, in a list is defined to be the number at the "top level"

`[ "a", "b", "c" ]`

`[ [ [1, 2, 3], [2, 3] ],  [ [3], [] ] ]`

`[[1, 2, 3, 4], [[1, 2], [3, 4]], [[[1, 2, 3,4]]]]`

Equality for Lists

- Two lists are defined to be *equal* when they have the same number of elements, and their elements occur in the same order.
- Examples:

  `[1,2,3,4]` is not equal to `[1,2,4,3]`

  `[1,2,3]` is not equal to `[1,1,2,3]`

Implementing Other Information Structures using Lists

Sets

- A set is not a list, but a finite set can be *implemented* as a list:
  - simply ignore the ordering of the list, and
  - either:
    - ignore duplicates, or
    - guarantee no duplicates

called a *representation invariant*

- Ignoring duplicates has advantages, such as in element removal (why?)

Association Lists

- An *association list* is a list of pairs.

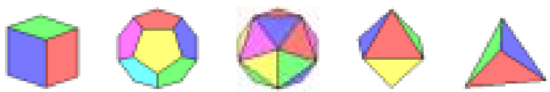
```
[[“January”, 31], [“February”, 28],
 [“March”, 31], [“April”, 30]]
```

- A *dictionary* associates a value with each member of a set (called the domain).

Example

- A dictionary of regular polyhedra represented with an association list:
  - With each name is associated *a pair* containing the number of faces and sides per face.

```
[[“cube”, [6, 4]],
 [“dodecahedron”, [12, 5]],
 [“icosahedron”, [20, 3]],
 [“octahedron”, [8, 3]],
 [“tetrahedron”, [4, 3]]
```



Binary Relations

- A *binary relation* on a set specifies which pairs of elements from this set are related, and which aren't.
  - The set is called the domain of the relation
  - Mathematicians like to represent a binary relation as the set of all pairs of related elements.
- A finite set can be represented as a list.
- An ordered pair can be represented as a list.
- Therefore, a binary relation with a finite domain can be represented as ...

Example

- Consider the binary relation "can be donor for" on the set of blood types: {A, B, AB, O}
- As a list, this could be represented:

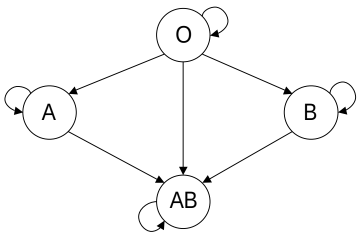
```
[[“A”, “A”],  [“A”, “AB”],  [“B”, “B”],
 [“B”, “AB”], [“AB”, “AB”], [“O”, “A”],
 [“O”, “AB”], [“O”, “B”],   [“O”, “O”]]
```

Directed Graphs

- A binary relation can be presented as a directed graph
  - The nodes of a directed graph correspond to the elements in the domain
  - The arcs (arrows) of a directed graph correspond to the pairs that are related

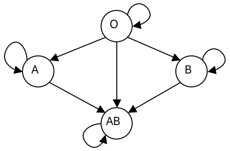
Directed Graph Example

- For the binary relation "can be donor for" the directed graph would be:



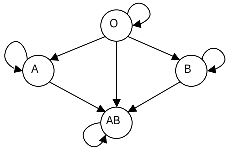
Properties of Binary Relations

- The previous relation example illustrates two common properties that a binary relation may have
  - *transitivity:* For every x, y, z in the domain (not necessarily distinct), if x is related to y and y is related to z, then x is related to z.
  - *reflexivity:* For every x in the domain x is related to x.



Properties of Binary Relations

- The previous example has one of the following two properties
  - *symmetry:* For every x, y in the domain if x is related to y then y is related to x.
  - *anti-symmetry:* For every x, y in the domain if x is related to y and y is related to x, then x = y.



Inferred Properties?

- Which of the following are true?
  - If the relation has the transitive and symmetric property, then it also has the reflexive property.
  - A relation cannot have both the symmetric and anti-symmetric property.

Additional Terminology

- An *equivalence relation* is a relation with the reflexive, symmetric, and transitive properties
  - If x is related to z and y is related to z, then x is related to y.
  - In other words, if each of a set of elements is related to a common thing, the elements in the set and the common thing are all related to each other.
- A relation with reflexive, anti-symmetric, and transitive properties is called a *partial order*.

Example

- Consider the relation “sounds the same as” on the set

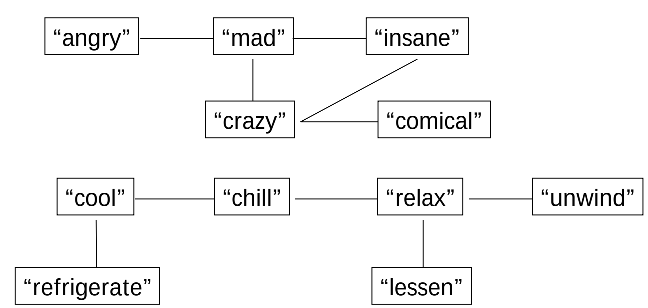
  {“air”, “ere”, “heir”, “buy”,
  “by”, “bye”, “dew”, “do”,
  “due”, “ewe”, “you”, “yew”}

- Is this an equivalence relation?

Undirected Graphs

- An undirected graph is a way of presenting a symmetric binary relation
  - Since whenever x is related to y also y is related to x, we don’t have to show direction with arcs.
  - Instead of calling them arcs then, it is common to call them edges.
- Representation?

Example: "is a synonym of"



More Information Structures

- There are a lot more information structures to talk about, but deferred until next time.
- Today's second topic is actually implementing the representations we have seen.

Programming with Lists in rex

Functional Programming

- Functional programming is one of the major fundamental programming paradigms.
  - It means programming only by evaluating expressions, not using assignment statements.
- It can be used in conjunction with other paradigms, such as object-oriented programming.

A Functional Programming Language

- We will use the language rex to exemplify functional programming.
- rex is interactive:
  - definitions are entered
  - expressions are evaluated to get results
- You may run rex on turing:

```
unix > rex
rex >
```

UNIX prompt

rex prompt

rex Usage Examples

```
rex > length([ [1, 2], [3, 4], [5, 6] ]);
3

rex > sort([3, 9, 1, 2, 8, 7, 5, 6, 4]);
[1, 2, 3, 4, 5, 6, 7, 8, 9]

rex > sort(["oats", "peas", "beans", "barley"]);
[barley, beans, oats, peas]

rex >
```

More rex Usage Examples

```
rex > x = [3, 9, 1, 2, 8, 7, 5, 6, 4];
1
```

This 1 means true, the definition was accepted.

```
rex > x;
[3, 9, 1, 2, 8, 7, 5, 6, 4]

rex > sort(x);
[1, 2, 3, 4, 5, 6, 7, 8, 9]

rex > x;
[3, 9, 1, 2, 8, 7, 5, 6, 4]
```

Still More rex Usage Examples

```
rex > length(x);
9

rex > reverse(x);
[4, 6, 5, 7, 8, 2, 1, 9, 3]

rex > append(x, x);
[3, 9, 1, 2, 8, 7, 5, 6, 4, 3, 9, 1, 2, 8, 7, 5, 6, 4]

rex >
```

Definitions in a File

contents of file `test.rex`, prepared with a text editor such as Emacs:

```
// This is a set of rex definitions, with comments
// x is a list of some small random numbers.
x = [3, 9, 1, 2, 8, 7, 5, 6, 4];

// y is a list of some grains.
y = sort(["oats", "peas", "beans", "barley"]);

// z is a list of pairs
z = [ [1, 2], [3, 4], [5, 6] ];

/*
   Above you see comments to end-of-line.
   You can also have multi-line comments such as this one,
       just like Java or C++.
*/
```

At least two ways to load a file:

Method 1: Include the file name on the UNIX command line:

```
unix > rex test.rex
test.rex loaded
rex > x;
[3, 9, 1, 2, 8, 7, 5, 6, 4]

rex > y;
[barley, beans, oats, peas]

rex > z;
[[1, 2], [3, 4], [5, 6]]
```

here

At least two ways to load a file:

Method 2: Include the file from a rex command line

```
unix > rex
rex > *i test.rex
read file test.rex
rex > x;
[3, 9, 1, 2, 8, 7, 5, 6, 4]

rex > y;
[barley, beans, oats, peas]

rex > z;
[[1, 2], [3, 4], [5, 6]]
```

here

Running Under Emacs

UNIX shell in emacs window

In Emacs:
`control-x 2` to split window
`escape-x shell` to get shell
Can cut/paste using only keystrokes

your rex file for editing

Abstraction Exercise

- Think up and describe an area outside of CS where you (or others) use abstraction.
  - Particularly multiple layers of abstraction
  - Examples:
    - Physics, Electronics, Logic Gates, FSMs, Pentium IV
    - Physics, Chemistry, Biology, Genetics