INSTITUTE OF SONOLOGY

Royal Conservatoire The Hague

Data Structures

Programming and Music
http://www.bjarni-gunnarsson.net

Data Structures

A data structure is a particular way of organizing data in a computer so that it can be used ***efficiently***.

Can be seen as a schema for organizing related pieces of information.

Well known data structures include items such as ***queue, stack, linked list, heap, dictionary***, and ***tree***, or conceptual unity, such as the name and address of a person.

A data structure may include redundant information, such as length of the list or number of nodes in a subtree.

Collections

Various types of collections exist in SuperCollider to best represent required data. The three main types are:

1. ***Indexed by number*** (Array, List, Interval, Range, Array2D, Signal, Wavetable, String)

2. ***Indexed by symbol*** or another object(Dictionary, IdentityDictionary, MultiLevelIdentityDictionary, Environment, Event, Library) these use lookup based on associations, pairs that associate a key and a value and are optimized for retreival using the association key.

3. Collections that are ***accessed by searching*** (Set and Bag)

Arrays

An ***array*** is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key.

Arrays are among the oldest and most important data structures, and are used by almost every program.

Arrays are useful because the element ***indices*** can be calculated when a program runs giving access to its elements with direct indexing.

Arrays

Arrays have a ***fixed*** maximum ***size*** beyond which they cannot grow.

Other data structures are appropriate for expanding lists.

Literal Arrays can be created at compile time. These are faster then those created during execution.

Arrays are ***enclosed in brackets*** and each item is separated by a comma.

An example: [1, 2, 3]

Dictionary

A ***Dictionary*** is an associative collection mapping keys to values. Two keys match if they are equal.

Each possible key appears ***at most once*** in a dictionary.

The contents of a Dictionary are ***unordered*** and one should not depend on the order of items in a Dictionary.

It is possible to iterate the ***keys***, the ***values*** or the ***pairs*** these create.

Dictionaries and other associative arrays allow easy ***lookup*** of data entries.

Set

A set is an ***unordered collection*** of objects where no two objects are the same.

Sets are one of the most fundamental concepts in mathematics.

Sets A and B are equal <u>if and only if</u> they have precisely the same elements.

Special operations can be performed on Sets such as:
Union and ***Intersection***,

Event

In SuperCollider, IdentityDictionary and its subclasses ***Environment*** and ***Event*** ***override the doesNotUnderstand method*** to forward message name and its arguments to be included in the dictionary. This allows for dynamic objects and object modelling.

Using events, one can simulate object by attaching to them both variables and methods.

Using a ***event-based approach*** is more quick and simple instead of writing a class and allows us to model more fluently.

Custom

Using classes in SuperCollider it is easy to implement ***custom data structures*** that reflect the needs of a specific problem.

This can be done both by using ***inheritance*** and by ***composing an object*** that consists of other objects.

An example of a custom data structure could be a class containg information for ***students***. This class could contain:

* A dictionary containing all the students
* Instance methods such as retrieving students in alphabetic order
* State variables indicating for example if students are available
* Class methods such as formatting the name for a student to print

```
(
    NF(\iop, {|freq=78, mul=1.0, add=0.0|
        var noise = LFNoise1.ar(0.001).range(freq, freq + (freq * 0.1));
        var osc = SinOsc.ar([noise, noise * 1.04, noise * 1.02, noise * 1.08],0,0.2);
        var out = DFM1.ar(osc,freq*4,SinOsc.kr(0.01).range(0.92,1.05),1,0,0.005,0.7);
        HPF.ar(out, 40)
    }).play;
)

(
    NF(\dsc, {|freq = 1080|
        HPF.ar(
            BBandStop.ar(Saw.ar(LFNoise1.ar([19,12]).range(freq,freq*2), 0.2).excess(
            SinOsc.ar( [freq + 6, freq + 4, freq + 2, freq + 8])),
            LFNoise1.ar([12,14,10]).range(100,900),
            SinOsc.ar(20).range(9,11)
        ), 80)
        ;
    }).play;
)
```

```
var <>pindex, <>cindex;

initialize {
    if(pindex.isNil, { pindex = 1000 });
    if(cindex.isNil, { cindex = 2000 });
}

clearProcessSlots {
    pindex = 1000;
    (this.pindex - 1000).do{|i| this[this.pindex+i] = nil; }
}

clearOrInit {|clear=true|
    if(clear == true, { this.clearProcessSlots() }, { this.initialize() });
}

transform {|process, index|
    if(index.isNil && pindex.isNil, {
        this.initialize();
    });

    pindex = pindex + 1;
    this[pindex] = \filter -> process;
}

control {|process, index|
    var i = index;

    if(i.isNil, {
        this.initialize();
        cindex = cindex + 1;
        i = cindex;
    });

    this[i] = \pset -> process;
}
```

Exercises

```
(
    NF(\depfm, {|freqMin=5, freqMax=20, mul=20, add=80, rate=0.5, modFreq=2100, index=0.3, amp=0.2|
        var trig, seq, freq;
        trig = Dust.kr(rate);
        seq = Diwhite(freqMin, freqMax, inf).midicps;
        freq = Demand.kr(trig, 0, seq);
        HPF.ar(PMOsc.ar(LFCub.kr([freq, freq/2, freq/3, freq/4], 0, mul, add),
        LFNoise1.ar(0.3).range(modFreq,modFreq*2), index) * amp, 50)
    }).play;
)
```

Exercises

1. Create an array that contains random numbers. Then iterate the array and compare it item by item and finally print the largest number found and the index it was found at.

2. Implement a function that generates an array that consists of 5 frequency numbers between 100 and 1000 Hz and an octave down and octave up transposed version of those numbers.

3. Implement a function that generates an array filled with a variable amount of odd numbers only.

4. A function that generates variable amounts of arrays containing random numbers.

5. Implement a function that fills a dictionary with a list of MIDI numbers. Write another one that iterates the dictionary and retrieves only odd number pitches.

6. Implement a function phoneBook that adds a person and her number to a phone book, but only if they have not been added before.