Data types

March 8, Lecture 13

Strings

- Some languages offer them as arrays of chars
- Other languages (most notably scripting) provide more extended support

- Usually specified as a sequence of characters, like ‘aba’ or “aba”
- C and descendants distinguish between single characters ‘a’ and strings “aa”

- **Escape sequences**: Non-printing characters and double quotations
- For example Pascal: ‘ab’’cde’ is six-character, where the 3rd one is quote mark
- C99 and C++ have many escape sequences :
  - (for example special escape for unicode)

Strings

- Several languages that don't allow dynamic arrays still allow dynamic strings
  - Fundamental to many things and easier to implement

- Some languages require that the string size is known at elaboration time so that they can be implement in stack: C, Pascal, Ada
- Pascal and Ada support string operators including assignment and comparison for lexicographical ordering
- C provides allows only to create a pointer to a string literal
  - char s* ---------- s="abc" (makes a point to static storage)
  - If it is declared as char s[6], that won't work

- In C++,Java,C# strings are objects

Sets

- An unordered collection of an arbitrary number of values of the same type

```
var A, B, C : set of char;
    D, E : set of weekday;
...
A := B + C;    (* union; A := {x | x is in B or x is in C} *)
A := B * C;    (* intersection; A := {x | x is in B and x is in C} *)
A := B - C;    (* difference; A := {x | x is in B and x is not in C} *)
```

- Most times sets are implemented as bit vectors of length enough to accommodate the maximum size of the set. For example a set of ASCII characters would be 128 bits (equal to their number).
- Union and intersection are easy bit-wise ORs and ANDs.

Pointers and recursive types

- Recursive type is one whose objects contain a reference to an other object of the same type.
- Most recursive types are records
- Used to build data structures, trees, lists, etc

- Languages like LISP, ML that use the reference model of variables it is easy to include recursion. Variables are references.
- Languages like C, Pascal, require the notion of a pointer, another variable whose value is a reference to an object.

Pointers and recursive types

- In some languages pointers are restricted to point only to the heap.
- In other languages (including C) pointers can refer to non-heap objects

- Issues arise with reclaiming space allocated in the heap, after its end of life
  - Some languages want the user to do it.
  - In other languages it's part of the definition and happens automatically

Pointers and recursive types

- In imperative languages variables can use a value model, or a reference model, or a combination of the two.
- In value model, A:=B, when A,B are pointers makes A point to the same object that B does
- This happens to all variables (not only pointers) when language uses reference model.
- This happens conceptually and semantically, but internally the language can implement **mutable** objects (such as trees) with addresses, but actual values for **immutable** objects. This helps with efficiency.

Pointers and recursive types

- Operations on pointers:

1. **Allocation and de-allocation** of objects in the heap
2. **Dereferencing**, to access the objects
3. **Assignment** of one pointer to another

- The behavior of these operators depend on whether the language is functional or imperative.

- Function languages employ a reference model to names
  - (there are no variables or assignments)
- Objects are occupied automatically with a structure determined by the language
  - Example is lists in LISP

Pointers and recursive types

- A tree implementation in ML (datatypes)

```
datatype chr_tree = empty | node of char * chr_tree * chr_tree;
```