22c:111 Programming Language Concepts

Fall 2008

Types II

Copyright 2007-08, The McGraw-Hill Company and Cesare Tinelli. These notes were originally developed by Allen Tucker, Robert Noonan and modified by Cesare Tinelli. They are copyrighted materials and may not be used in other course settings outside of the University of Iowa in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holders.
Contents

5.1 Type Errors
5.2 Static and Dynamic Typing
5.3 Basic Types
5.4 NonBasic Types
5.5 Recursive Data Types
5.6 Functions as Types
5.7 Type Equivalence
5.8 Subtypes
5.9 Polymorphism and Generics
5.10 Programmer-Defined Types
### 5.5 Recursive Data Type (in OCaml)

```ocaml
type variable = string (* just a synonym *)
type op = string

type value = IntValue of int | FloatValue of float |
             BoolValue of bool | CharValue of char

type expression = Var of variable | Lit of value |
                  Binary of (op * expression *expression) |
                  Unary of (op * expression)
```
5.6 Functions as Types

Pascal example:

```pascal
function newton(a, b: real; function f: real): real;
```

Know that \( f \) returns a real value, but the arguments to \( f \) are unspecified.
public interface RootSolvable {
    double valueAt(double x);
}

public double Newton(double a, double b, RootSolvable f);
5.7 Type Equivalence

Pascal Report:

*The assignment statement serves to replace the current value of a variable with a new value specified as an expression. ... The variable (or the function) and the expression must be of identical type.*

Nowhere does it define *identical type.*
struct complex {
    float re, im;
};
struct polar {
    float x, y;
};
struct {
    float re, im;
} a, b;
struct complex c, d;
struct polar e;
int f[5], g[10];
// which are equivalent types?
5.8 Subtypes

A subtype is a type that has certain constraints placed on its values or operations.

In Ada subtypes can be directly specified.
Ada

subtype one_to_ten is Integer range 1 .. 10;
type Day is (Monday, Tuesday, Wednesday,
    Thursday, Friday, Saturday, Sunday);
subtype Weekend is Day range Saturday .. Sunday;
type Salary is delta 0.01 digits 9
    range 0.00 .. 9_999_999.99;
subtype Author_Salary is Salary digits 5
    range 0.0 .. 999.99;
Java

// Integer is a subclass of Number, 
// and therefore a subtype
...
Integer i = new Integer(3);
...
Number v = i;
...
Integer x = (Integer) v;
Polymorphism and Generics

A function or operation is *polymorphic* if it can be applied to any one of several related types and achieve the same result.

An advantage of polymorphism is that it enables code reuse.

This sort of polymorphism is more correctly referred to as *parametric* polymorphism
Polymorphism vs. Overloading

Polymorphic means: having many forms

Overloading fits this description:

Example: overloaded built-in operators and functions $+$ $-$ $*$ $/$ $==$ $!=$ $...$ in C, C++, Java

However, overloaded symbols are not polymorphic in the parametric sense
Java: instance variable, method

- name, name()
- *ad-hoc polymorphism* (ie., overloading)

Ada generics: *generics*

- A form of parametric polymorphism
- type binding delayed from code implementation to compile time

Ocaml/Haskell: *parametric types and functions*

- Full blown parametric polymorphism
generic
    type element is private;
    type list is array(natural range <> ) of element;
    with function ">(a, b : element) return boolean;

package sort_pck is
    procedure sort (in out a : list);
end sort_pck;
package sort_pck is
procedure sort (in out a : list) is
begin
   for i in a'first .. a'last - 1 loop
      for j in i+1 .. a'last loop
         if a(i) > a(j) then
            declare t : element;
            begin
               t := a(i);
               a(i) := a(j);
               a(j) := t;
            end;
         end if;
      end for;
   end for;
end sort;
Instantiation

package integer_sort is

    new generic_sort( Integer, ">");
Programmer-defined Types

Recall the definition of a type:

A set of values and a set of operations on those values.

Structures allow a definition of a representation; problems:

• Representation is not hidden
• Type operations cannot be defined

Defer further until Chapter 12.