Concepts of Programming Languages

Lecture 09 - Advanced Types

Patrick Donnelly

Montana State University

Spring 2014

Administrivia

Reading:

Chapter 6

Types are the leaven of computer programming;
they make it digestible.

Robin Milner

Character String Types

Design issues:

- Is it a primitive type or just a special kind of array?
- Should the length of strings be static or dynamic?

Now fundamental and directly supported.

Libraries of string operations and functions.

Values are sequences of characters

Character String Types Operations

Typical operations:

- Assignment and copying
- Comparison (=, >, etc.)
- Catenation
- Substring reference
- Pattern matching

Character String Type in Certain Languages

C and C++

- Not primitive
- Use `char` arrays and a library of functions that provide operations

Character String Type in Certain Languages

C and C++

- Not primitive
- Use `char` arrays and a library of functions that provide operations

SNOBOL4 (a string manipulation language)

- Primitive
- Many operations, including elaborate pattern matching

Character String Type in Certain Languages

C and C++

- Not primitive
- Use `char` arrays and a library of functions that provide operations

SNOBOL4 (a string manipulation language)

- Primitive
- Many operations, including elaborate pattern matching

Fortran and Python

- Primitive type with assignment and several operations

Character String Type in Certain Languages

C and C++

- Not primitive
- Use `char` arrays and a library of functions that provide operations

SNOBOL4 (a string manipulation language)

- Primitive
- Many operations, including elaborate pattern matching

Fortran and Python

- Primitive type with assignment and several operations

Java

- Primitive via the String class

Character String Type in Certain Languages

C and C++

- Not primitive
- Use `char` arrays and a library of functions that provide operations

SNOBOL4 (a string manipulation language)

- Primitive
- Many operations, including elaborate pattern matching

Fortran and Python

- Primitive type with assignment and several operations

Java

- Primitive via the String class

Perl, JavaScript, Ruby, and PHP

- Provide built-in pattern matching, using regular expressions

Character String Length Options

Static: COBOL, Java's String class

Limited Dynamic Length: C and C++

- In these languages, a special character is used to indicate the end of a string's characters, rather than maintaining the length

Dynamic (no maximum): SNOBOL4, Perl, JavaScript

Ada supports all three string length options

Structures (Records)

Analogous to a tuple in mathematics

Collection of elements of different types

Used first in Cobol, PL/I

Absent from Fortran, Algol 60

Common to Pascal-like, C-like languages

Omitted from Java as redundant

Structure Example

```
struct employeeType {
      int id;
      char name[25];
      int age;
      float salary;
      char dept;
};

struct employeeType employee;
...
employee.age = 45;
```

Unions

C: union

Pascal: case-variant record

Logically: multiple views of same storage

Useful in some systems applications

Structure Example

```
type union =
     record
     case b :  boolean of
          true :  (i :  integer);
          false :  (r :  real);
     end;

var tagged :  union;
begin tagged := (b => false, r => 3.375);
     put(tagged.i); - error
```

Structure Example in C++

```
// simulated union type
class Value extends Expression {
      // Value = int intValue | boolean boolValue
      Type type;
      int intValue;
      boolean boolValue;
      Value(int i) {
            intValue = i;
            type = new Type(Type.INTEGER);
      }
      Value(boolean b) {
            boolValue = b;
            type = new Type(Type.BOOLEAN);
      }
}
```

Tuple Types

A tuple is a data type that is similar to a record, except that the elements are not named

Used in Python, ML, and F# to allow functions to return multiple values

Example: in Python:

- Closely related to its lists, but immutable
- Create with a tuple literal

```
myTuple = (3, 5.8, 'apple')
```

- Referenced with subscripts (begin at 1)
- Catenation with + and deleted with del

Tuple Types

ML:

```
val myTuple = (3, 5.8, 'apple');
```

- Access as follows:

`#1(myTuple)` is the first element

- A new tuple type can be defined

```
type intReal = int * real;
```

F#:

```
let tup = (3, 5, 7)

let a, b, c = tup
```

This assigns a tuple to a tuple pattern `(a, b, c)`

List Types

Lists in LISP and Scheme are delimited by parentheses and use no commas

`(A B C D)` and `(A (B C) D)`

Data and code have the same form

- As data, `(A B C)` is literally what it is
- As code, `(A B C)` is the function `A` applied to the parameters `B` and `C`

The interpreter needs to know which a list is, so if it is data, we quote it with an apostrophe

`'(A B C)` is data

Recursive Data Type

```
data Value = IntValue Integer | FloatValue Float |
     BoolValue Bool | CharValue Char
     deriving (Eq, Ord, Show)

data Expression = Var Variable | Lit Value |
     Binary Op Expression Expression |
     Unary Op Expression
     deriving (Eq, Ord, Show)

type Variable = String

type Op = String

type State = [(Variable, Value)]
```

Functions as Types

Pascal example:

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

Note that `f` returns a real value, but the args to `f` are unspecified.

```
public interface RootSolvable {
            double valueAt(double x);}

public double Newton(
            double a, double b, RootSolvable f);
```

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.

Type Equivalence

```
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?

Subtypes

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

In Ada subtypes can be directly specified.

```
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;
```

Subtypes

```
Integer i = new Integer(3);
...
Number v = i;
...
Integer x = (Integer) v;
```

Integer is a subclass of Number, and therefore a subtype

Polymorphism

Definition

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.

Word comes from Greek

Means ‘having many forms’

Examples of Polymorphism

Overloaded built-in operators and functions, such as '+':

Example

```
a + b
"hello " + "world"
```

Examples of Polymorphism

Java overloaded methods – number or type of parameters:

Example

class **`PrintStream`** defines `print`, `println` for
`boolean`, `char`, `int`, `long`, `float`,
`double`, `char[ ]`, `String`, `Object`

Examples of Polymorphism

Java overloads instance variable and methods:

Example

```
String name;

public String name() { ... }
```

Generics

Ada introduced the concept of generics.

Definition

A ***generic function*** or procedure is a template that can be instantiated at compile time with concrete types and operators.

Generic procedures are examples of parametric polymorphism.

Type binding delayed from code implementation to compile time.

Java sort

```
public static void sort(Comparable[ ] a){
    for (int i = 0; i < a.length; i++)
        for (int j = 0; j < a.length; j++)
            if (a[i].compareTo(a[j]) > 0){
                Comparable t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
}
```

```
public interface Comparable{
    public abstract int compareTo(Object o);
}
```

Programmer-defined Types

Recall the definition of a type:

Definition

A ***type*** is a collection of values and operations on those values.

Structures allow a definition of a representation; problems:

1. Representation is not hidden
2. Type operations cannot be defined

We will return to this concept in the context of Object-Oriented Programming.