

The University of North Carolina at Chapel Hill

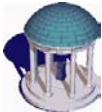
COMP 144 Programming Language Concepts
Spring 2003

Introduction to Data Types

Stotts, Hernandez-Campos

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

1

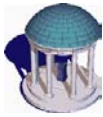


Data Types

- Computers manipulate sequences of bits
- But most programs manipulate more general data
 - Numbers
 - String
 - Lists
 - ...
- Programming languages provide data types that raise the level of abstraction from bits to data
 - But computer hardware only knows about bits!

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

2



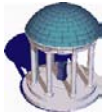
Data Types

The Purpose of Types

- Types provide **implicit context**
 - Compilers can infer more information, so programmers write less code
 - *E.g.* the expression `a+b` in Java may be adding two integer, two floats or two strings depending on the context
- Types provides a set of semantically **valid operations**
 - Compilers can detect semantic mistakes
 - *E.g.* Python's lists support `append()` and `pop()`, but complex numbers do not.

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

3

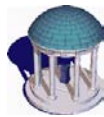


Type Systems

- High-level languages have ***type systems***
 - All objects and expressions have a type
 - *E.g.* `int (*) (const void *)` is the type of a C++ function
- A type system consists of
 - A mechanism for defining types and associating them with certain language constructs
 - A set of rules for ***type checking***
 - » *Type equivalence*
 - » *Type compatibility*
 - » *Type inference*

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

4



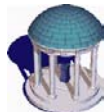
Type Systems

Type Checking

- Type checking is the process of ensuring that a program obeys the language's **type compatibility rules**
- *Strongly typed languages* always detect types errors
 - *Weakly typed languages* do not
 - All expressions and objects must have a type
 - All operations must be applied in appropriate type contexts
- *Statically typed languages* are strongly typed language in which all type checking occurs at compile time
 - *Dynamically typed languages*: some checking at run-time
 - E.g. Haskell vs. Java

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

5

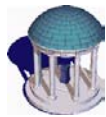


What is a type?

- Three points of view:
 - Denotational: *a set of values*
 - Constructive: *a type is built-in type or a composite type*
 - » Composite types are created using type constructors
 - » E.g. In Java, `boolean` is a built-in type, while `boolean[]` is a composite type
 - Abstraction-based: *a type is an interface that defines a set of consistent operation*
- These points of view complement each other

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

6



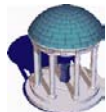
Classification of Types

Built-in Types

- Built-in/Primitive/Elementary types
 - Mimic hardware units
 - *E.g.* boolean, character, integer, real (float)
- Their implementation varies across languages
- **Characters** are traditionally one-byte quantities using the ASCII character set
 - Early computers had a different byte sizes
 - » Byte = 8 bits standardized by Fred Brooks *et al.* thanks to the IBM System/360
 - Other character sets have also been used

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

7



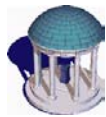
Built-in Types

Unicode Characters

- Newer languages have built-in characters that support the Unicode character set
 - <http://www.unicode.org/unicode/standard/WhatIsUnicode.html>
 - Unicode is implemented using two-byte quantities
 - *E.g.* Java
 - » <http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html>
 - *E.g.* Python
 - » `u"¡Hola!"`
 - » <http://www.python.org/doc/current/tut/node5.html#SECTION00513000000000000000>

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

8



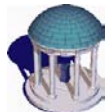
Built-in Types

Numeric Types

- Most languages support integers and floats
 - The range of value is implementation dependent
- Some languages support other numeric types
 - Complex numbers (*e.g.* Fortran, Python)
 - Rational numbers (*e.g.* Scheme, Common Lisp)
 - Signed and unsigned integers (*e.g.* C, Modula-2)
 - Fixed point numbers (*e.g.* Ada)
 - Bignums (int Python, int vs. Integer in Haskell)
- Some languages distinguish numeric types depending on their precision
 - Single vs. double precision numbers
 - » C's `int` (4 bytes) and `long` (8 bytes)

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

9



Built-in Types

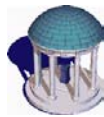
Bignum

- Python example

```
i = 0
k = 2
while i < 10:
    print k*k
    print "    \n"
    k = k*k
    i = i+1
```

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

10



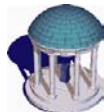
Built-in Types

Bignum?

- Try the same in Perl

```
$i = 0;  
$k = 2;  
  
while ($i < 10) {  
    print $k*$k, " \n";  
    $i++;  
    $k *= $k;  
}
```

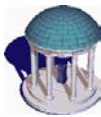
Need a Perl module for this...



Classification of Types

Enumerations

- Enumerations improve program readability and error checking
- They were first introduced in Pascal
 - `type weekday = (sun, mon, tue, wed, thu, fri, sat);`
 - They define an order, so they can be used in enumeration-controlled loops
 - The same feature is available in C
 - » `enum weekday {sun, mon, tue, wed, thu, fri, sat};`
 - Other languages use constants to define enumeration
 - Pascal's approach is more complete: integers and enumerations are not compatible



Classification of Types

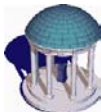
Subranges

- Subranges improve program readability and error checking
- They were first introduced in Pascal
 - E.g.

```
type test_score = 0..100;  
    workday = mon..fri;
```
 - They define an order, so they can be used in enumeration-controlled loops

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

13



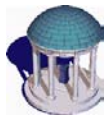
Classification of Types

Composite Types

- Composite/Constructed types are created applying a constructor to one or more simpler types
- Examples
 - Records
 - Variant Records
 - Arrays
 - Sets
 - Pointers
 - Lists
 - Files

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

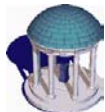
14



Classification of Types

Orthogonality

- Orthogonality is an important property in the design of type systems
- More orthogonal languages support have more flexible composite types



Reading Assignment

- Scott's chapter 7
 - Intro
 - Section 7.1