CSE 3302
Programming Languages

Data Types

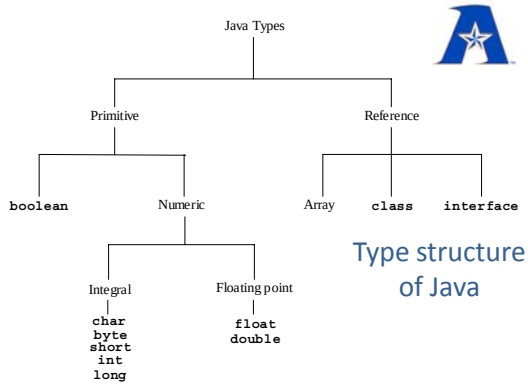
Chengkai Li, Weimin He
Spring 2008

Data Types

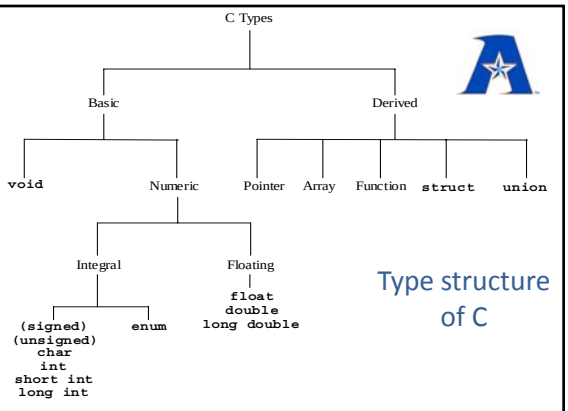
- **What is a data type?**
  A name with certain attributes:
  – The values that can be stored, the internal representation, the operations, …
- **A data type is a set of values**
  - e.g., `int` in Java:
    `int x;`
    x∈ Integers= [-2147483648, 2147483647]
- **A data type is also a set of operations on the values**

Why are data types important?

- **Example: z = x / y; (Java)**
  – int x, y; x=5; y=2;
    - Integer division, x/y results in 2.
    - int z: z = 2;
    - double z: z=2.0;
  – double x, y; x=5; y=2;
    - floating-point division, x/y results in 2.5
    - int z: wrong!
    - double z: z=2.5;



Type structure of Java



Type structure of C

Simple Data Types

- **No internal structure:**
  e.g., `integer`, `double`, `character`, and `boolean`.
- **Often directly supported in hardware.**
  – machine dependency
- **Most predefined types are simple types.**
  – Exceptions: *String* in Java.
- **Some simple types are not predefined**
  - Enumerated types
  - Subrange types

Enumerated Types

Ordered set, whose elements are ***named*** and ***listed*** explicitly.

- Examples:

```
enum Color_Type {Red, Green, Blue};        ( C )
type Color_Type is (Red, Green, Blue);     (Ada)
datatype Color_Type = Red | Green | Blue;  (ML)
```

- Operations: ?
  Successor and predecessor

Ada Example

```
type Color_Type is (Red, Green, Blue);

x : Color_Type := Green;
x : Color_Type'Succ(x);
x : Color_Type'Pred(x);
put(x);        -- prints GREEN
```

- No assumptions about the internal representation of values
- Print the value name itself

Pascal Example

```
type
     cardsuit = (club, diamond, heart, spade);
     card = record
               suit: cardsuit;
               value: 1 .. 13;
            end;
var
     hand: array [ 1 .. 13 ] of card;
```

- `Succ(diamond) = heart; Pred(spade) = heart;`
- `club < heart;` is true.
- `for acard := club to heart do`

C Example

```
#include <stdio.h>
enum Color {Red, Green, Blue};
enum Courses {CSE1111=1, CSE3302=3, CSE3310=3, CSE5555=4};
main() {
    enum Color x = Green;
    enum Courses c = CSE3302;
    x++;
    printf("%d\n",x);
    printf("%d\n",Blue+1);
    printf("%d\n",c);
   return 0;
}
```

- `Enum` in C is simply `int`
- Can customize the values

Java Example

```
public enum Planet { MERCURY (2.4397e6), EARTH (6.37814e6);
  private final double radius; // in meters
  Planet(double radius) {this.radius = radius; }
  private double radius() { return radius; }

  public static void main(String[] args) {
    for (Planet p : Planet.values())
      System.out.printf("The radius of  %s is %f\n", p, p.radius());
  }
}
```

`java.util.Enumeration has different meaning`

```
for (Enumeration e = v.elements() ; e.hasMoreElements() ;)
    System.out.println(e.nextElement());
```


Evaluation of Enumeration Types

- **Efficiency** – e.g., compiler can select and use a compact efficient representation (e.g., small integers)
- **Readability** -- e.g. no need to code a color as a number
- **Maintainability** – e.g., adding a new color doesn't require updating hard-coded constants.
- **Reliability** -- e.g. compiler can check operations and ranges of value.

Courtesy of Charles Nicholas at UMBC

C Example for Maintainability

```
enum Color {White, Green, Blue, Black};
enum Color {White, Yellow, Green, Blue, Black};
main(){
    enum Color x = Black;
    int i = x;
    while (i >= White){
        if (i < Green)
            printf("this is a light color!\n");
        i--;
    }
}
```

What if no enumeration?

```
if (i < 1) printf("this is a light color!\n");
```

Has to be changed to:

```
if (i < 2) printf("this is a light color!\n");
```


Ada Example for Reliability

```
type DAY is (MON, TUE, WED, THU, FRI, SAT, SUN);
type DIRECTION is (NORTH, EAST, SOUTH, WEST);

GOAL : DIRECTION;
TODAY : DAY;
START : DAY;

TODAY := MON;
GOAL := WEST;
START := TODAY;

TODAY := WEST; -- Illegal: WEST is not a DAY value
TODAY := 5; -- Illegal: 5 is not a DAY value
TODAY := TODAY + START; -- Illegal: "+" is not defined for DAYS
```


Subrange Types

Contiguous subsets of simple types, with a ***least*** and **greatest** element.

- Example:

```
type Digit_Type is range 0..9;              ( Ada)
```

- Not available in C,C++,Java. Need to use something like:

```
byte digit;     //-128..127
...
if (digit>9 || digit <0) throw new DigitException();
```

- defined over **ordinal types:**
  - ordered, every value has a next/previous element
    - E.g., integer, enumerations, and subrange itself

Type constructors: Defining New Types

- **Type constructors as set operations:**
  - Cartesian product
  - Union
  - Subset
  - Functions (Arrays)
- **Some type constructors do not correspond to set operations** (e.g., pointers)
- **Some set operators don't have corresponding type constructors (e.g., intersection)**

Cartesian Product

- **Ordered Pairs of elements from U and V**

  $U \times V = \{(u, v) \mid u \in U \text{ and } v \in V\}$

- **Operations:**
  - **projection**

  $p_1: U \times V \rightarrow U$; $p_2 : U \times V \rightarrow V$

  $p_1((u,v))=u$; $p_2((u,v))=v$

Examples

- `struct` in C

```
struct IntCharReal
{    int i;
     char c;
     double r;
}
```

`int × char × double`

- `record` in Ada

```
type IntCharReal is record
     i: integer;
     c: character;
     r: float;
end record;
```

The same type?

```
struct IntCharReal
{     int i;
      char c;
      double r;
}

struct IntCharReal
{     char c;
      int i;
      double r;
}
```


The same type?

```
struct IntCharReal
{     int i;
      char c;
      double r;
}

struct IntCharReal
{     int j;
      char ch;
      double d;
}
```


Record/structure are not exactly Cartesian products

- Component selector: projection by component names

```
struct IntCharReal x;
x.i;
```

- Most languages consider component names to be part of the type.
- Thus the previous two types can be considered different, even though they represent the same Cartesian product.

ML: Pure Cartesian Product

```
type IntCharReal = int * char * real;
```

- `(2, #"a", 3.14)`
- `#3(2, #"a", 3.14) = 3.14`

Union

- $U \cup V = \{ x \mid x \in U \text{ or } x \in V \}$
  - data items with different types are stored in overlapping region, reduce memory allocation.
  - Only one type of value is valid at one time.
  - E.g.,

```
union IntOrReal {
      int i;
      double r;
}
```

- Different from records?

Undiscriminated Union in C

```
union IntOrReal {
          int i;
          double r;
  }
  union IntOrReal x;
  x.i = 1;
  printf("%f\n", x.r);
```

- Can be unsafe

Create Discriminated Union in C++

```
struct IntOrReal {
    bool isInt;
    union {
          int i;
          double r;
    };
};

IntOrReal x;
x.isInt = true;
x.i = 1;
...
if (x.isInt) printf("%d\n", x.i);
else printf("%f\n", x.r);
```

- Safe now
- or not?

Discriminated Union in Ada

- Variant record (with tag or discriminator)

```
type Disc is (IsInt, IsReal);
type IntOrReal (which: Disc) is
record
   case which is
        when IsInt =>  i: integer;
        when IsReal =>  r: float;
   end case;
end record;
...
x: IntOrReal := (IsReal, 2.3);
put (x.i);  -- generates ERROR
```

- Safe: programmers won't be able to create inconsistent data

Discriminated Union in Pascal

- Variant record
- Can be unsafe:
  - First, the tag is optional
  - Second, the tag can be set inconsistently.

Discriminated Union in ML

```
datatype IntOrReal =
     IsInt of int | IsReal of real;
```

- `val x = IsReal(2.3);`

How about Java?

- Is there record or union in java? Why?

"Union" in Java

- `public abstract class A {…};`
  `public class B extends A {…};`
  `public class C extends A {…};`

  Abstract class `A`: union of `B` and `C`.

- Discriminated union: `instanceof`

Subset

- U = { *v* | *v* satisfies certain conditions and *v*∈V}
- Ada subtype
- Example 1
  - `type Digit_Type is range 0..9;`
  - `subtype IntDigit_Type is integer range 0..9;`

`subtype` in Ada

- Example 2

```
type Disc is (IsInt, IsReal);
type IntOrReal (which: Disc) is
record
   case which is
        when IsInt =>  i: integer;
        when IsReal =>  r: float;
   end case;
end record;

subtype IRInt is IntOrReal(IsInt);
subtype IRReal is IntOrReal(IsReal);

x: IRReal := 2.3;
```


Powerset

- P(U) = { U' | U' ⊆ U }
- Example: Pascal
  `set of` <ordinal type>
  - **`var`** `S: set of 1.. 10;`
  - **`var`** `S: 1.. 10;`
  - `What's the difference?`
- The order isn't significant though
  unordered, values are distinct

`set of` in Pascal

```
var S, T: set of 1 .. 10;
S := [1, 2, 3, 5, 7];
T := [1 .. 6];
```

`Set operations can be performed on the variables.`
What are these?

- `T := S*T;`
- `If T = [1, 3, 5] then …;`
- `x = 3;  if x in S then …;`
- `if S<=T then …;`

- ∩ (*) ∪ (+) − (-) = (=) ≠ (<>)
- ⊃ (>) ⊇ (>=) ⊂ (<) ⊆ (<=) ∈ (in)

Arrays and Functions

f: U → V

index type component type

- [0, …] (C/C++/Java)
- Ordinal type (Ada/Pascal)

C/C++

Allocated on stack, size statically specified.

```
typedef int TenIntArray [10];
typedef int IntArray [];

TenIntArray x;
int y[5];
int z[]={1,2,3,4};
IntArray w={1,2};
IntArray w;   //illegal
int n = ...   //from user input
int a[n];     //illegal
```

Java

Allocated on heap, size dynamically specified;
Size can be obtained by .length

```
int n = ...   //from user input
int [] x = new int [n];
System.out.println(x.length);
```


Ada

size dynamically specified;
Set of subscripts

```
type IntToInt is array(integer range <>) of integer;

get(n);   //from user input
x: IntToInt(1..n);
for i in x'range loop
  put(x(i));
end loop;
```


Multi-dimensional arrays

- C/C++

```
int x[10][20];
```

- Java

```
int [][] x = new int [10][20];
```

- Ada

These two are different

```
type Matric_Type is array(1..10, -10..10) of integer;
x(i,j);

type Matric_Type is array(1..10) of array (-10..10) of integer;
x(i)(j);
```


Storage

- row-major form

```
x[1,-10],x[1,-9],…,x[1,10],x[2,-10],…,x[2,10],x[3,-10],…
```

- column-major form

```
x[1,-10],x[2,-10],…,x[10,-10],x[1,-9],…,x[10,-9],x[1,-8],…
```

- C/C++

```
int array_max(int a[][20], int size)
```

- Java

```
int array_max(int [][][]a)
```