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:
    ```java
    int x;
    x ∈ Integers = [-2147483648, 2147483647]
    ```
- A data type is also a set of operations on the values
  Thus a data type is an algebra

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 \( z = 2; \)
    - 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 \( z; z=2.5; \)

Data Types in Sample Languages

- Different terminologies and hierarchies for similar things.
C Types

- Basic: void, numeric, pointer, function, struct, union
- Numeric: int, char, (signed), (unsigned), long int
- Floating: float, double, long double
- Enum: enum
- Struct: struct

Simple Data Types

- No internal structure: e.g., integer, double, character, and boolean.
- Often directly supported in hardware:
  - Machine dependency
  - Standardization efforts: e.g., IEEE standard 754 floating point
    - Single precision: 32 bit representation with 1 bit sign, 8 bit exponent, 23 bit mantissa
- 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;
  x++;
  printf("%d\n",x);
  printf("%d\n",Blue+1);
  return 0;
}

- What's the result?
  - Enum in C is simply int
  - Can customize the values
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
    - Even floating number is ordinal type in Ada
      - Unit_Interval is digits 8 range 0.0 .. 1.0;

Type constructors: Defining New Types

- Remember our view of data types as sets?
- 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)

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.

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
    $$p_1 : U \times V \rightarrow U; \quad p_2 : U \times V \rightarrow V$$
    $$p_1((u,v))=u; \quad p_2((u,v))=v$$
  - copy

Examples

- `struct` in C
  ```
  struct IntCharReal
  { int i;
    char c;
    double r;
  }
  int x char x 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;
}
```
The same type?

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

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

Record/structure are not exactly Cartesian products

- Component selector: projection by component names
  - ```c
  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

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

- `(2, "a", 3.14): string * int`
- `#3(2, "a", 3.14) = 3.14`

Union

- ```c
  \( U \cup V = \{ x \mid x \in U \text{ or } x \in V \} \)
  ```
  - 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.,
    ```c
    union IntOrReal
    {
        int i;
        double r;
    }
    ```

- Different from records?

Undiscriminated Union in C

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

```c
union IntOrReal x;
```

```c
x.i = 1;
printf("%d\n", x.i);
```

- Can be unsafe

Create Discriminated Union in C++

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

```c
IntOrReal x;
```

```c
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)

```ada
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;
```

```ada
x: IntOrReal := (IsReal, 2.3);
```

- 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

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

```ml
val x = IsReal(2.3);
```

How about Java?

- Is there record or union in java? Why?

“Union” in Java

```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
```