### 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 = -2147483648;
  ```

**A data type is also a set of operations on the values**

---

### Why are data types important?

**Example:**

```java
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; z = 2;`
  - `double z; z = 2.5;`

---

### Java Types

**Type structure**

- `Primitive`: `boolean`, `Numeric`, `int`, etc.
- `Reference`: `class`, `interface`.

### C Types

**Type structure**

- `Basic`: `void`, `int`, `float`, `double`, `enum`.
- `Derived`: `struct`, `union`.

---

### 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:**
  ```c
  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**

```ada
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
```
C Example for Maintainability

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

Has to be changed to:
```c
if(i < 2) printf("this is a light color!\n");
```

Ada Example for Reliability

```ada
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:
  ```ada
type Digit_Type is range 0..9;  
```
- Not available in C,C++,Java. Need to use something like:
```c
byte digit; // -128..127
```
- defined over ordinal types:
  ```ada
  -- E.g., integer, enumerations, and subrange itself
  ```

Cartesian Product

- Ordered Pairs of elements from U and V
  ```c
  U x V = \{ (u, v) | u \in U and v \in V \}
  ```
- Operations:
  ```c
  \ - projection
  p_1: U x V \rightarrow U;  p_2: U x V \rightarrow V
  p_1(u,v)=u;  p_2(u,v)=v
  ```

Examples

- struct in C
  ```c
  struct IntCharReal
  { int i;
    char c;
    double r;
  }
  int x char x double
  ```
- record in Ada
  ```ada
  type IntCharReal is record
  i: integer;
  c: character;
  r: float;
  end record;
  ```
The same type?

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

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

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.,
    ```c
    union IntOrReal {
        int i;
        double r;
    }
    ```
- Different from records?

Undiscriminated Union in C

```c
union IntOrReal { 
    int i;
    double r;
} 
union IntOrReal x;
printf("%f\n", x.r); 
```
- Can be unsafe
Create Discriminated Union in C++

```cpp
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)

```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;
...
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 inconsistency.

Discriminated Union in ML

```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

```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**

- $U = \{ v \mid v \text{ satisfies certain conditions and } v \in V \}$
- 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
dcase which is
  when IsInt => i: integer;
  when IsReal => r: float;
end case;
end record;
subsubtype IRInt is IntOrReal(IsInt);
subsubtype IRReal is IntOrReal(IsReal);
x: IRReal := 2.3;

**Powerset**

- $P(U) = \{ U' \mid U' \subseteq U \}$
- 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 ...
- $x = 3; \text{ if } x \in S \text{ then } ...$
- If $S \subseteq T$ then ...

- $\cap(*)$, $\cup(\oplus)$, $-(-)$, $\in(\in)$

**Arrays and Functions**

```
f: U \rightarrow 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 \texttt{.length}

\begin{verbatim}
int n = ... //from user input
int[] x = new int[n];
System.out.println(x.length);
\end{verbatim}

Ada

size dynamically specified;
Set of subscripts

\begin{verbatim}
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;
\end{verbatim}

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[10,-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)