Data Types

- finite set of representations
- assembly language : untyped

strongly typed - can't defeat typesystem
weakly typed - can get @ the machine
Dennis Ritchie : "C is strongly typed but weakly checked"
C++ reinterpret_cast<>

Do not confuse with
static - @ compile time
dynamic - @ run time

strong dynamic types: Perl, Smalltalk, Scheme, Python, Prolog, ...
- most scripting lang

But static catches errors earlier

Static types : - execution efficiency
- translation effic.
- writability - compile time errors
- security, reliability
- remove ambiguities
- design tool
- interface checking

Primitive Data Types (2)

- built into language
- char, byte, short, int, long, float double
- Java specifies exactly sizes
- C/C++ leaves it to compiler writer

specify operators +, −, * usual but division?

i/j → float or int

Pascal i/j → ~~real~~ real

i div j → int

zerodivide: int error

float → ±∞ or NaN?

int division

~~a/b divide => trunc(a/b)~~

~~a quot b =>~~

ML: i div j = floor(i/j)

i quot j = trunc(i/j) ← int / in C

i mod j = i − (i div j)*j

i rem j = i − (i quot j)*j ← int % in C

ceiling

round

Data Types (3)

Type Constructors

- define user types
  - typedef
  - struct
  - union
  - enumeration

} syntax varies by language

```
typedef  vector<int>  vecint;
using  vecint = vector<int>;
enum {RED, GREEN, BLUE};
```

What types are equivalent?

- by name
- by structure

Types: strong – can't defeat type system
weak – possible to break things
untyped – assembly language

don't confuse weak with dynamic

Cartesian Product Type

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

need projection function

~~struct foo {~~ – creates a constructor
~~int a;~~
~~int b;~~ – creates 3 selectors
~~int c;~~
~~}~~ (in C++ must use

```
struct foo{
  int a{};
  int b{};
  int c{};
};
```

creates a constructor fn
creates 3 selector fns

C++ allows uninitialized ctors
C all ctors uninit?
Java always init

Union

```
union{
    int a;
    double d;
};
```

- overlays fields some storage
- discriminated union has a flag to indicate which
- Pascal (pre-OO) lang had unions
- Java, C++ instead use OO.

Subset

- subtype small is integer range 0..99;
- still an int but restricts range
- Pascal & Ada

Newtypes

type dollars is new double;

- Ada
- dollars and doubles can't mix
- but dollars has all double opers.

~~Arrays & Functions~~ DataTypes (5)

Tags & Variants

```
type Foo = Int of int
         | Real of float
         | Error;
Foo s;
match s with
    | Int i  -> f(i)
    | Real r -> g(r)
    | Error  -> ~~throw~~ raise xxx;
```

Arrays & Functions

$f : U \to V$

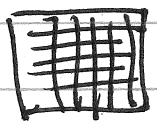
- can b array type or fn type — except Perl
- is a mapping
- arrays usually builtin; hashes not
- could cache fn values in array or table
- array index type 0..(n-1)?
  - arrays of arrays } not same
  - N dimensional arrays }
  - lower bound
  - slicing? a[i:j]
  - int a[n] — on stack? ← no C&O, no Java
  - runtime dimension — no FORTRAN IV
  - resize once alloc? ← no C/C++/Java, yes Perl

Data Types (6)

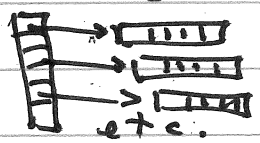
- C: can pass array address only
- C: not return array
- Java — all arrays on heap

Pascal: array bounds part of type

int x[10][20] in C



"new int[10][10] in Java



this is usual in C
e.g. argv.

ragged arrays

Matrix storage

column major order (Fortran)

row major order (C, C++, etc)

multi-D arrays difficult in C.

~~Pointers & Recursive types~~

~~Functions~~

Pointers & Recursive Types

int *

```
struct foo {
    struct foo* pj
}
```

implict in languages that do gc.

Data Types (7)

references

Java = like pointers
- but no ptr arithmetic
- not actual addresses
- updated w/ diff address by g c

C++ — like pointers
- may not be updated
- frozen on one object
- implicit dereference

Smalltalk, Perl — like Java

int& i = j;

recursive types

C/C++
```
struct node {int n; struct node* link;};
struct node *p;
```

Java similar but w/o unary *

Ocaml: type List = End | Node of int * List;

Parametric data Types

C++ template <typename T>

Java 5+ class<T> // generic

old Java - (none) but use Object as element

Perl, Smalltalk, Scheme - unnecessary (dynamic types)

Ocaml:
type 'a List = End | Node of 'a * List

End of list or failure

C++ ∅, nullptr

C: ∅ (NULL)

Java null

Ocaml type 'a option = None | Some of 'a

Scheme #f

Type Equivalence

- are two types the same?

```
struct A {          struct B {          struct C {          struct {
  int a;              char c;             int x;              int a;
  char c;             int a;              char y;             char c;
};                  };                  };                  } foo;
```

which structs are equivalent

with structural equivalence: A, C, foo

with name equivalence: none

C uses name equivalence for structs

C++/Java same.

(foo) - last one is an anonymous type.

- what about struct C *p;

where def of C not in file (private)

- use name equivalence

C/C++/Java use name equivalence for structs/classes

void foo(double (*f)(double));

decl of foo whose arg is a double (*)(double)
i.e ptr to fn with double arg & double result

double sqrt(double);
double sin(double); } proto types.

C++/C - uses structural equivalence for fn signatures

typedef int foo;

Java does not. has no functions, only methods
- implement an interface instead

ex: void foo(Mathfn f);

```
interface Mathfn {
    double eval(double);
}
```

Type Checking

compile time

run time

required in strongly typed languages
- static OR dynamic

C++/Java - type check is static
- class membership is dynamic

Interpretive - dynamic check

Type inference

```
int a; int b; int c = a + b;
```

type of result of + is int

no type conversion

```
C++11 auto c = a + b;
```

c is inferred as int

```
vector<int> v;
auto i = v.cbegin()
```

inferred as vector<int>::const_iterator

Ocaml:

```
let f n = match n with
  | 0 -> 1
  | n when n > 0 -> n * f (n-1)
  | raise error;
```

BAD! not tail recursive

static type inference

```
f : int -> int
n : int
* : int -> int -> int
- : int -> int -> int
> : int -> int -> bool
```

← uses unification

Type Compatibility

compat if one can assign to another

- lvalue = rvalue;
- same with fn arg & return

implicit conversion
  C++ numerics
  non explicit ctors

explicit - casts
  - constructors

static_cast
dynamic_cast
reinterpret_cast
const_cast
} avoid if at all possible.

Type conversion (coercion)

- implicit
- widening (int → long, float → double)
- narrowing (loses information)
- explicit (cast) (ctor)

Polymorphic Type checking

Hindley-Milner type inference

let max gt x y = if gt x y then x else y
assign types α β γ δ β γ δ γ δ

1. assign type variables to elements

max : $\beta \to \gamma \to \delta \to \varepsilon$

gt : $\gamma \to \delta \to \zeta$

but inside if :. gt : $\gamma \to \delta \to$ bool

then else must be same. :. $\gamma \equiv \delta$

so max: $(\gamma \to \gamma \to \text{bool}) \to \gamma \to \gamma \to \gamma$

became $\varepsilon \equiv \gamma \equiv \delta$

:. max : ('a → 'a → bool) → 'a → 'a → 'a

in C++

```
template <typename A>
A max (bool (*)(A,A), A, A);
```

Explicit Polymorphism