6. Data Types

1. Primitive data types
2. Character strings
3. Ordinals
   (1) Enumeration
   (2) Subrange
4. Arrays
5. Hashes
6. Records
7. Union
8. Pointers
9. Sets

[1] Primitive Data Types

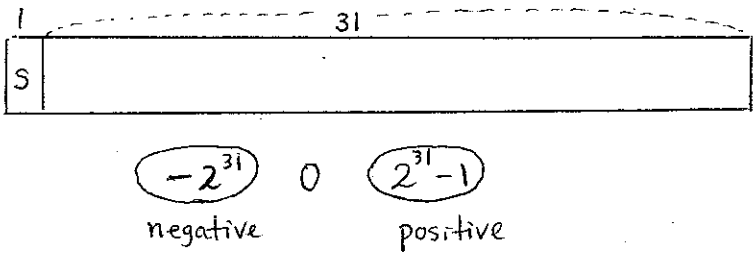
(1) Numeric

① Integer

- 1, 2, 4, 8 bytes
- Two's complement

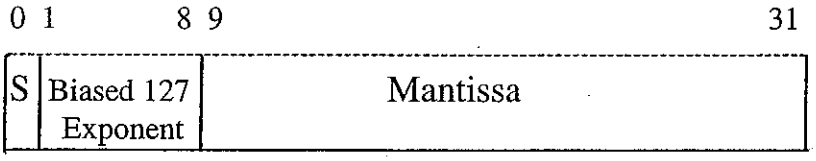
Advantages over Sign-magnitude method

1. Only one representation for zero
2. Subtraction can be treated as addition.



② Floating-Point Numbers (Real)

$$\pm S \times B^{\pm E}$$



-127 ~ +128

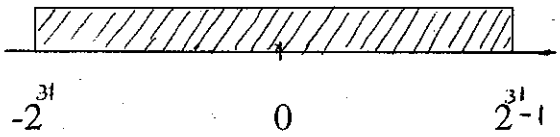
Ex. $1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010\ldots$

$1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010\ldots$

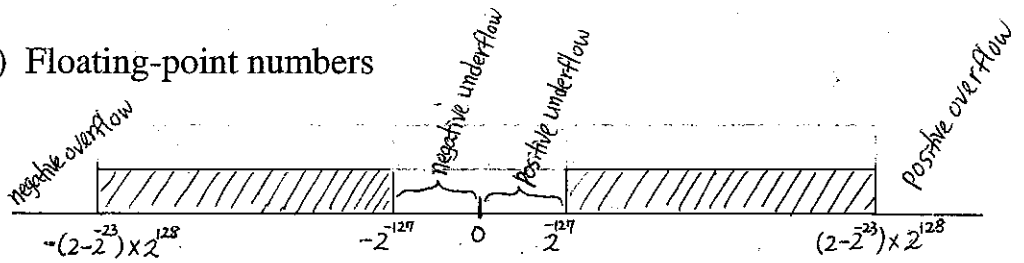
Note. Normalized: $+ 1.bbb..b * 2^{\pm E}$
Base: Either 2 or 16 (IBM S/370)

Expressible numbers in 32-bit word

(1) 2's complement integers

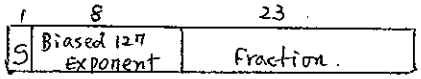


(2) Floating-point numbers



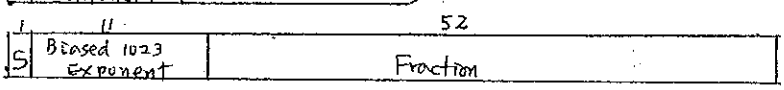
IEEE Standard

Single form



$10^{-38} - 10^{38}$

Double form

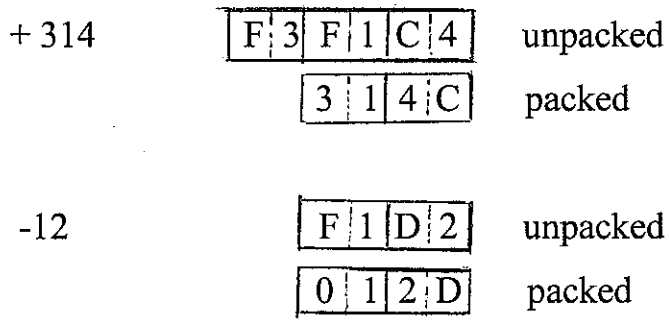


$10^{-308} - 10^{308}$

③ Decimal (Fixed-point)

- Business data processing
- Cobol, PL/I, C#

Ex. IBM S/370



(2) Boolean

- true(1) or false(0)
- readability
- Algol 60, Pascal, C

  C: false (0), true (others)

(3) Character

- o ASCII (8 bits)
- o Unicode (16 bits)

  Java, JavaScript, Python, Perl, C#

[2] Character String

- writability

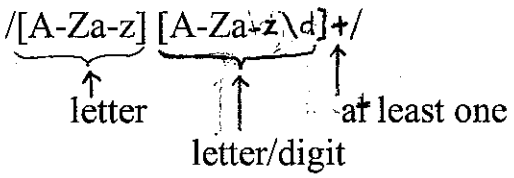
Operations on character strings: assignment, catenation, substring reference, comparison, pattern matching

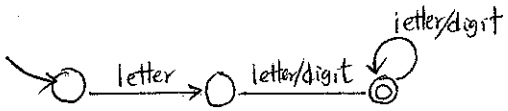
- Primitive type: Fortran 95, Java, C#, Ruby, Python
- Character string: C/C++
  Library (string.h) – unsafe
  strcpy, strcmp, strlen

Pattern matching

- based on regular expression
- Perl, JavaScript, Ruby, PHP

Ex. Variable name





o String length

- Static length strings:
  Python, Java, C++, Ruby, C#

compile time

length
address

- Dynamic length strings:
  JavaScript, Perl, Snobol
  C/C++ (limited)

run time

maximum length
current length
address

[3] Ordinal Data Types

- increase readability and reliability

(1) Enumeration types

Pascal, C/C++

Ex 1.
```
type   class = (fresh, sopho, junior, senior);
var  student_class:  class;
       :
if (student_class = junior) then ---
```

Implementation: 2 bits

00: fresh
01: sopho
10: junior
11: senior

Ex 2.
```
enum   colors {red, blue, green, yellow, black};
colors   myColor = blue, yourColor = red;

myColor++              // myColor = green //
```

(2) Subrange types

Pascal, Ada

Ex.
```
type  days = (mon, tue, wed, thu, fri, sat, sun);
      weekdays = mon..fri;
      weekends = sat..sun;

var   day1:  days;
      day2: weekdays;

day2: = day1;        // Error on sat or sun //
```

[4] Array Types

Ex. A: array [1..10, -5..5] of real;

Data type of A:	array
Dimension:	2
Row name:	1,2,…10
Column name:	-5,-4,…,5
Number of elements:	10 x 11
data type of each element:	real

(1) Syntax

- A(5): same as function call
- A[5]

(2) Range of subscripts (indices)

- A[1], A[2], A[3], A[4], A[5] : Fortran, Pascal, Ada
- a(0), a(1), a(2), a(3), a(4) : C/C++

(3) Range checking

- Yes: Ada, Java, ML, C#
- No: Fortran, C/C++, Perl

Note. Perl

Second element of the array @list ➔ $list[1]

↑ scalar

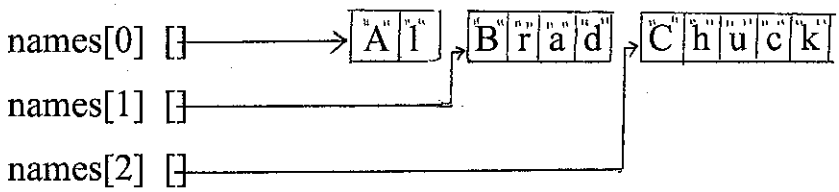
(4) Initialization

Fortran 95:

```
Integer, Dimension(3) :: List = (/0, 0, 0/)
```

C/C++, Java, C#

```
int list [ ] = {10, 20, 30, 40};
char name [ ] = "msu';      // 4 elements, including null character //
char *names [ ] = {"Al", "Brad", "Chuck"};
```



Java

```
String [ ]  names = {"Al", "Brad", "Chuck"};
```

Ada

```
List:   array(1..5) of Integer  :=  (1, 0, 5, 0, 0);
or List:   array(1..5) of integer  :=  (1 => 1, 3 => 5, others => 0);
```

(5) Rectangular array vs. Jagged array

Rectangular: Fortran, Ada, C#

myArray [3,7] // 2-D array //

Jagged: C/C++, Java, C#

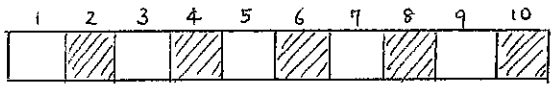
myArray [3][7] // array of array //

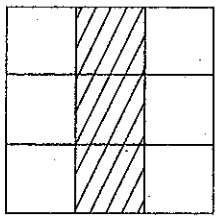
(6) Slices

- Fortran 90

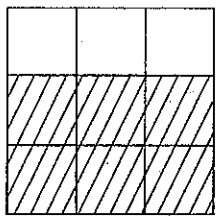
Ex. Dim (10) :: Vector
Dim (3,3) :: Mat
Dim (3,3,4) :: Cube

Vector (2:10:2)

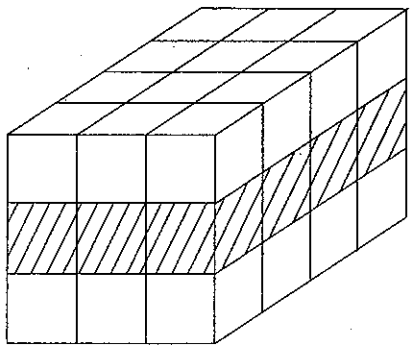




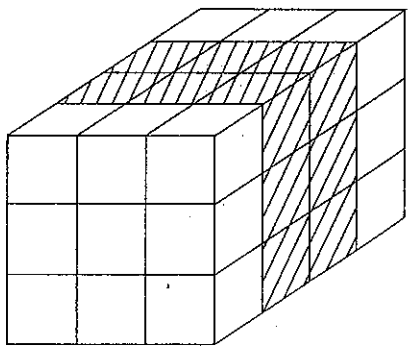
Mat (:, 2)



Mat (2:3, :)



Cube (2, :, :)

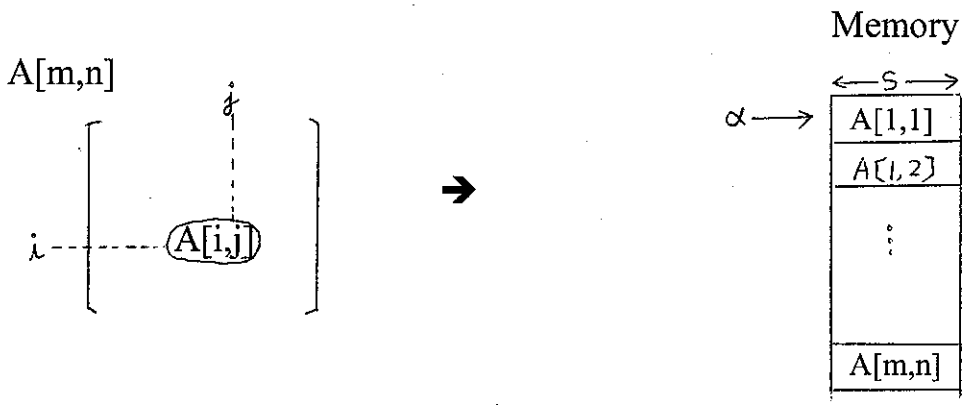


Cube (:, :, 2:3)

(7) Implementation

{ row-major order
{ column-major order: Fortran

Mapping function



Let (1) address of the first element be α.
(2) size of an element be s.

[Case 1] lower index is 1. (Fortran, Pascal, Ada)

row-major: Loc (A[i,j]) = α + [(i-1) n + (j-1)] s

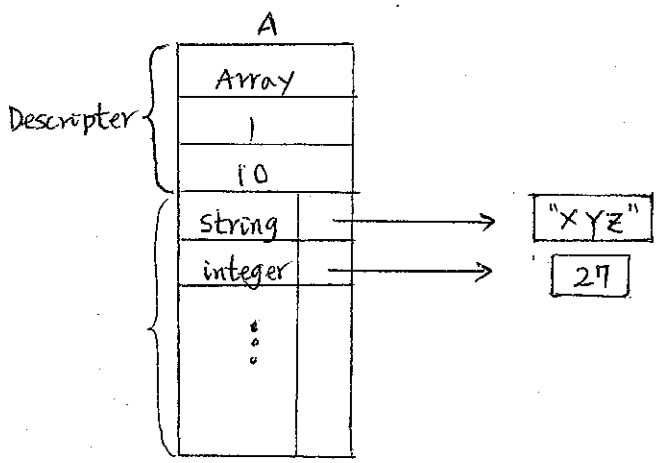
column-major : Loc (A[i,j]) =

[Case 2] lower index is 0. (C/C++, Java)

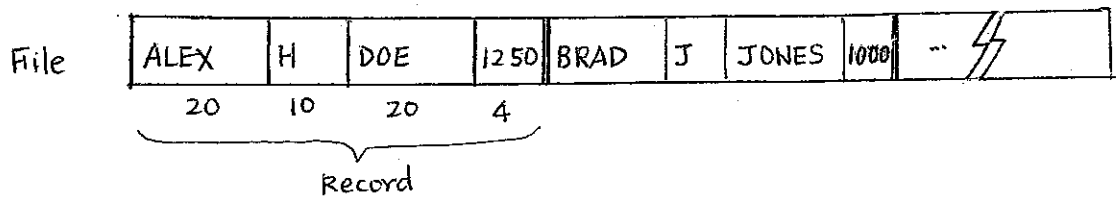
row-major: Loc (A[i,j]) =

Snobol 4.

```
A = ARRAY(10)
A<1> = "XYZ"
A<2> = 27
```



[6] Records



Cobol

```
01  EMPLOYEE-RECORD.
    02  EMPLOYEE-NAME.
        05  FIRST    PICTURE IS X(20).
        05  MIDDLE   PICTURE IS X(10).
        05  LAST     PICTURE IS X(20).
    02  HOURLY-RATE PICTURE IS 99V99.
```

Ada

```
type Employee_Name_Type is record
   First : String (1..20);
   Middle : String (1..10);
   Last : String (1..20);
end record;
type Employee_Record_Type is record
   Employee_Name: Employee_Name_Type;
   Hourly_Rate: Float;
end record;
Employee_Record: Employee_Record_Type;
```

Note. PL/I, Pascal, Java, C#

References to Record Fields

Cobol: MIDDLE OF EMPLOYEE-NAME OF EMPLOYEE-RECORD

Ada: Employee_Record.Employee_Name.Middle

Operations on Records

```
01  INPUT-RECORD.
    02  NAME.
        05  LAST           PICTURE IS X(20).
        05  MIDDLE         PICTURE IS X(15).
        05  FIRST          PICTURE IS X(20).
    02 EMPLOYEE-NUMBER     PICTURE IS 9(10).
    02 HOURS-WORKED        PICTURE IS 99.

01  OUTPUT-RECORD.
    02  NAME.
        05  FIRST          PICTURE IS X(20).
        05  MIDDLE         PICTURE IS X(15).
        05  LAST           PICTURE IS X(20).
    02  EMPLOYEE-NUMBER PICTURE IS 9(10).
    02  GROSS-PAY         PICTURE IS 999V99.
    02  NET-PAY           PICTURE IS 999V99.
```

```
MOVE CORRESPONDING INPUT-RECORD TO OUTPUT-RECORD.
```

[9] Set Type

- Pascal, Modula-2

(1) Set declaration

```
type   Digits = 0..9;
       DigitSet = set of Digits;
var    Numbers, Evens, Odds: DigitSet;
       S:  set of 1..10
```

(2) Set assignment

```
Evens: = [0, 2, 4, 6, 8];
Odds  := [1, 3, 5, 7, 9];
Numbers:= [ ];                  // empty set //
```

(3) Set operations

Union: Set1 + Set2

Ex. Numbers := Evens + Odds;

Intersection: Set1 * Set2

Difference: Set1 – Set2

Note. Size of a set is limited (system dependant).

If |w| = 32, maximum size of a set is 32.