Data Types

Data Type:

- Defines a collection of data values and a set of predefined operations on those data values
- Contemporary concepts of data types have evolved over the past 60 years.

In earliest language,

only a few data structures were supported. For instance, in pre-Fortran linked lists and binary trees were implemented with arrays.

- ALGOL68, is the first language that provided a few flexible structure-defining operators with which a programmer can design the data structure for each need.

In the imperative programming languages

the two most common structured data types are arrays and records → Non scalar

structured data types are generally defined with type operators, or constructors

For example, C use brackets and asterisks as type operators to specify arrays and pointers.

Concepts of Descriptors

- A descriptor is a collection of attributes of a variable.

  implementation-wise

  Descriptor is an area of memory that stores the attributes of a variable

- If attributes are all static, descriptors are required only at the compile time.

  Descriptors are built by the compiler, usually as a part of the symbol table.

- For dynamic attributes,

  part or all of the descriptor must be maintained during execution

Use of Descriptors :

- Descriptors are used for type checking and building the code for allocation and deallocation.

Static String
Length
Address

Compile time descriptor

Limited dynamic String
Maximum length
Current length
Address

Run-time descriptor

⊞ Primitive Data types:

- Data types that are not defined in terms of other types are called primitive data types
- Nearly all programming languages provid a set of primitive data types.
  - Some primitive data types are merely reflections of the hardware.
    ↳ Integers
  - Other requires only a little non-hardware support for their implementation.
- Structured types are defined using the primitive data types with one or more type constructors.

Integers:

- It is the most common primitive numeric data type
- Hardware of computers supports several sizes of integers.
  - byte, short, int, long : Java
  - C/C++ : Unsigned integers
    ↳ integers without signs

Floating Point:

- Model real numbers, but only as approximations.
  - Fundamental numbers such as π, e can not be correctly represented

    $\pi = 3.14159265358793 \ldots$

    $e = 2.71828182845904$

- Floating points are stored as binary, which exacerbates the problem.
  - Finite memory problem
  - Accuracy issues in arithmetic operations.

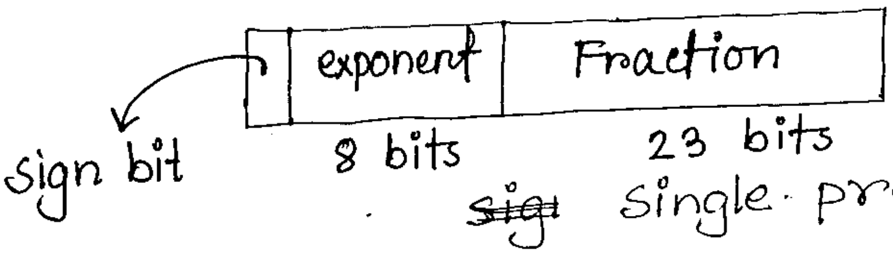
- Most languages include two floating-types
  - float 4 bytes
  - double 8 bytes

Additionally, we also have "long double"

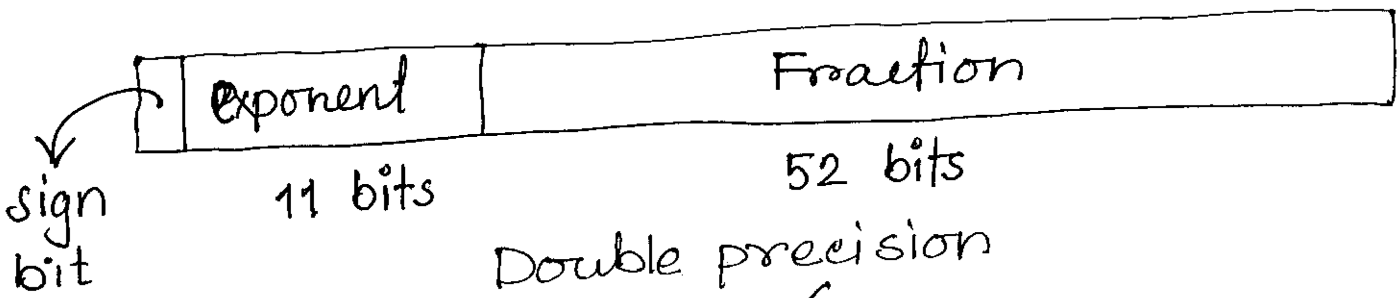
As in c-program:

Type	storage	Precision
float	4 byte	6 decimal
double	8 byte	15 decimal
long double	10 byte	19 decimal

- IEEE Floating-Point standard 754 are generally used



~~sig~~ single precision



Double precision
↳ Accuracy of the fraction part

Complex Number:

Some programming languages also support a complex data type.

For instance, Python and Fortran

In Fortran: let's say $x = 1 + i \equiv a + ib$

so, $a = 1$, $b = 1$

Then, the syntax goes as follows:

```
complex :: x
x = (1, 1)
```

In Python

```
import cmath
x = 10
y = 20
z = complex(10,20)
```

```
import numpy
complex (4,5)
```

$$\text{complex}(4,5) \equiv 4 + i5$$

Complex ~~a~~ number in C

```
typedef struct complex {
        float real;
        float imag;
                        } complex;
int main ( ) {
  complex test;
  complex  A, B;
  test. real  =  A.real + B.real
  test. imag  =  A.imag + B.imag
```

Decimal data type :

- Computers that are designed to support business systems application have hardware support for decimal data types.
  - Essential to COBOL

- Convenient and accurate for some decimal numbers. For instance,

BCD format
↳ Binary Coded Decimal

0.1 in decimal type be exactly represented, whereas, this could not be exactly represented as floating type.

- Advantage : Accuracy; as explained above
- Disadvantage : Limited range, wasteful
  ↳ memory

Boolean Types:

- Simplest of all the data types
- Range of values : two elements

$$\begin{tikzcd} & \text{two elements} \arrow[dl] \arrow[dr] & \\ \text{True} & & \text{False} \end{tikzcd}$$

- Often used to represent switches or flags
- Represented by a single bit, but as a single bit of memory can not be accessed, they are often stored as byte.
- Advantage: Readablity

Character:

characters are stored in computer as numeric codings.

↳ most commonly used coding: ASCII

↓

Use values from 0 to 127 to represent 128 characters

Alternative:

- Unicode character code that uses 16-bit character set.
- Unicode ~~uses~~ includes characters from most of the natural language
- Originally used in Java; later, C# and JavaScript also supported it.

⊞ Character String Types:

A character string type is one in which the values consists of sequence of characters.

↳ Mostly, input and output of all kind of data are often done in terms of strings.

Design Issues:

One of the most intriguing design questions for charcter string is its length.

↳ should the length be static or dynamic ?

⊞ Character string types operations

The most common string operations are

- Assignment and copying
- Comparison
- Concatenation
- Substring reference
- Pattern matching

Design issue:

Should strings be a primitive type or character array.?

↳ if strings are not defined as a primitive type, strings are stored as array of single characters. Example: C/C^{++} use char arrays to store character strings

char str[]= "apples"

As defined, $\boxed{\text{str}}$ is an array of $\boxed{\text{char}}$ elements
Precisely, here, the character elements
appleso
$\hookrightarrow$ Null character

String operations

Some of the most commonly used library functions in C/C++ are:

strcpy :	moves string
strcat :	catenates one string with other
strcmp :	Compares two given strings lexicographically.
strlen :	Returns the number of characters in the given string

by the order of their character codes.

- In C/C++, strings are simply arrays, whereas the Java string is treated as an object of the class java.lang.String.
- In Java, "+" operator is overloaded to facillitate string concatenation.

1. Assignment name = "Hello World"
2. Length name.length( )
= 10

~~Q. Consider the code segment~~

Character String Type in Certain Languages

C and C++
- Not primitive
- Use char arrays and a library of functions that provide operations.

Fortran and Python :
- Primitive type with assignment and several operations

Java
- Primitive via the string class

Perl, JavaScript, Ruby, PHP
- Provide built-in pattern matching using regular expressions.

Some "string" operations in PYTHON

```
str1 = 'Hello'     |  str3 =     str1 + str2
str2 = 'World'     |        = HelloWorld
                   |  str4 =    str * 3
                   |        = HelloHelloHello
```

That is : "+" ≡ catenation
'*' ≡ Repeatition

Iteration :
```
count = 0
for i in 'Hello World':
    if (i==    ):
        count += 1
print (count, 'letters found')
```
3 letters found will be printed

Membership :
```
> 'o' in 'Hello'     |  > 'el' not in 'Hello'
True                 |  False
```

Built-in functions :
```
str = 'northsouth'
list_enumerate = list (enumerate (str))
     = [(0, 'n'), (1,'o'), (2,'r') ...
        (3, 't'), (4, 'h') ... ... ]
print ('len(str) = ', len (str))
     len(str) = 4
```

format ( ) method for Formatting strings in Python

Consider a string object

North, south, Dhaka

Basic_order = "{ }, { } and { }". format ~~[illegible]~~ ('North', 'South', 'Dhaka')

$\downarrow$

North, South and Dhaka

Position of the strings —

position_order = "{1}, {0} and {2}". format ('North', 'South', 'Dhaka')

$\downarrow$

South, North. and Dhaka

A few other string operations and processings are allowed in Python

- format ( )
- lower ( )
- upper ( )
- join ( )
- split ( )
- find ( )
- replace ( )

String Length Options

Static length : String length is static and set when the string is created.

As in Python, ~~b~~ Java, Ruby etc. the strings are static.

Limited dynamic length:

Allows string to have varying length up to a declared and fixed minimum set by the variable's definition.

For example, in c, it uses a special character to indicate the end of the strings.

Dynamic length strings:

- Allows strings to have varying length with no maximum length.
- Example: JavaScript, Perl, and the standard C++ library.

Important —

- requires the overhead of dynamic storage allocation and deallocation

But ↳ it provides the maximum flexibility.

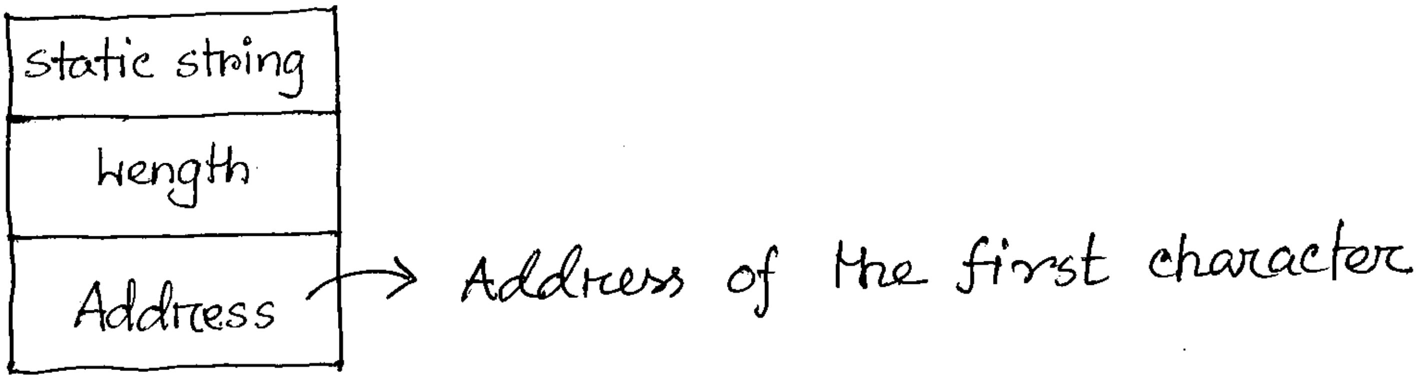
Evaluation

- String types are important to the [illegible] writability.

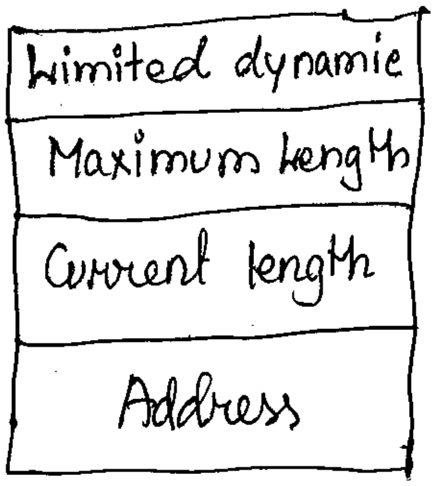
- Dealing with strings as arrays can be cumbersome.

  For instance, consider a case where the language treats strings as arrays of characters and does not have ~~for loop~~ strcpy ( ). as in C programming. would require a loop.

A descriptor for a static character string type:



limited dynamic Strings



Enumeration Type :

- In enumeration type, all the possible values are enumerated; provided

  Example : "named constant"

- Enumeration types provide a way of defining and grouping collection of named constants.

  As in C# :

  ```
  enum days {Mon, Tue, Wed, Thu, Fri, Sat,
             Sun};
  ```

  ↳ Typically, integer values 0, 1, ... are implicitly assigned.

Design Issues :

Are enumeration values coerced to

↳ If enumeration variable is coerced to a numeric type, then there is little control over its range.

For instance, if an int type value is coerced to an enumeration type, then the enumeration type variable could be assigned <u>any integer value.</u>,

↳ could be constant or not.

⊞ Examples of enumeration type

As in C++, which has inherited enumeration type from C, has enumeration example as—

enum $\boxed{\text{colors}}$ {red, blue, green, yellow, black}

↳ enumeration

colors MyColor

↳ so, MyColor denotes {red, blue, green, yellow, black}

We also can initiate the value of MyColor:

colors MyColor = blue, YourColor = red;

Now, $\boxed{\text{MyColor ++}}$ if used, green will be assigned to MyColor.

⊞ Evaluation

It provides advantages in both readability and reliablity.

readability → Named values are easily recognized.

Reliablity : No arithmetic operations are legal on enumeration type.

→ Prevents adding of days

No enumeration variable can be assigned a value outside its defined range.

ARRAY

By definition, an array is a homogeneous aggregate of data elements, where individual element is identified by its position in the aggregate.

Reference to individual element of an array is specified through subscript expression.

If subscript expressions in a reference are specified using variables, reference need to perform run-time calculation to determine the address.

Design Questions/Issues :

What types are legal for subscripts?

Are subscripting expressions in element references range checked?

When are the subscript range bound?

When does allocation take place?

Are ragged or rectangular multidimensional arrays allowed or both?

What is the maximum number of subscripts?

Are any kind of slices supported?

Array Indexing:

Indexing is $\overset{a}{\wedge}$ mapping from indices to elements.

array_name(index_1) $\longrightarrow$ an element

Syntax of Index:

- Fortran and Ada use parentheses ( )
- Most other languages use brackets

Array Index (subscript) Type:

- FORTRAN, C : Integer type
- Java : Integer type

- Index range checking
  - No range checking in C/C++, Fortran
  - Java, C# specify range checking
  - Ada : default is range checking, but could be turned off.

Array Categories :

- Static Array :

  Subscript of arrays are statically bound and storage allocation is static. before, run-time

  Example: static int myarray[2] = {5,6}

  Advantage: efficiency (No dynamic Allocation)

- Fixed Stack-dynamic :

  subscript ranges are statically bound, but allocation is done at declaration time.

  Disadvantage: Allocation deallocation needed

  Same space allocated in two-subprograms can be

  Advantage: Space efficiency

  Example: int array[3] = {2, 5, 7} ranges are done

- Stack-dynamic : Fixed Stack-dynamic

  subscript ranges are dynamically bound and the storage allocation is dynamic. (done at run-time)

  Advantage: flexibility

  Array size need not be known until the array is to be used.

-

Fixed heap-dynamic :

- Similar to Fixed ~~he~~ Stack-dynamic
  - ↳ subscript ranges and the storage bindings are both fixed after storage is allocated

- Both the subscript ranges and storage binding are done when user program requests during execution.) After the request is made
  - ↙ storage is allocated from the heap; not from the stack.

- Advantage :
  Flexibility
  - ↳ Array size always fits the problem.

- Disadvantage :
  Allocation time from heap, which is generally higher than the allocation time from the stack.

Heap-dynamic array:

- Subscript range and storage allocation is dynamic and can change any number of times during array life-time.
- Provides flexibility: Arrays can grow and shrink during program execution.
- Disadvantage: Allocation and deallocation takes longer and may happen many times during execution of the program.

Array Initialization

Some language allow initialization at the time of storage allocation

For instance, as in C, C++, Java, C# example

```
int list [ ] = {4, 5, 7, 83}
```

```
char name [ ] = "test";
```

Character strings in C and C++

Arrays of strings in C and C++

```
char *names [ ] = {"Bob", "Jake",
                   "Joe" };
```

Array operations:

- Operations that work on array as a unit.
  - Assignment
  - Concatenation
  - Comparison for equality and inequality
  - Slices
- Not all programming languages offer all the aforementioned array operations.

For instanee,

In python, where arrays are called lists, the arrays are heterogeneous types

⊞ A heterogeneous array is one in which the elements need not of the same type. ↳ Supported by Python, Perl, JavaScript and Ruby

test= array([12, 7.5, -7, 'three'])

Python allows:

Array assignment

```
course = ['CSE173', 'CSE425', 'CSE499', 'CSE417']
             0         1         2         3
```

Array concatenation

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list3 = list1 + list2
      = [1, 2, 3, 4, 5, 6]
```

Other ways for concatenation of arrays in python are numpy.concatenate

```
>>> a = np.array([[1,2], [3,4]])
    b = np.array([[5,6]])

    np.concatenate((a,b), axis=0)
    array([[1,2],
           [3,4],
           [5,6]])
```

axis=0 → added as a new row

```
    np.concatenate((a,b), axis=1)
    array([[1,2,5],
           [3,4,6]])
```

⊞ Rectangular and Jagged Arrays

Rectangular array : It is a multi-dimensioned array in which all of the rows have the same number of elements and all columns have same number of elements.

Jagged Arrays:

- Length of rows need not be the same.
- For example, a jagged matrix may consist of 3 rows, one with 5 elements, one with 7 elements and one with 12 elements.

Example: C#

- So, Jagged array is an array of arrays
- The elements of a jagged array can be of different dimensions and size.

Let's consider that we define a single-dimensional array that has three elements, each of which is a single-dimensional array of integers.

```
int [ ][ ] JaggedArray = new int [3] [ ]
```

Syntax: data_type [ ][ ] name = new data-type [rows][ ]

Here, a single-dimensional array is defined, which has 3 rows. Each row is 1-D array of integers. Now, we define the elements of 1-D integer array.

```
JaggedArray ~~[~~0] = new int [5]
JaggedArray [1] = new int [4]
JaggedArray [2] = new int [2]
```

```
JaggedArray [0] = new int [ ]
                  {1, 3, 5, 7, 9}
JaggedArray [1] = new int [ ]
                  {0, 2, 4, 6}
JaggedArray [2] = new int [ ]
                  {11, 22}
```

It is possible to define multi-dimensional array. For instance,

$$\texttt{int [,] array = new int } [\underbrace{4}_{\text{rows}}, \underbrace{2}_{\text{columns}}]$$

$$\underbrace{\texttt{int [, ,] array-name = new int [4, 2, 3]}}_{\text{three-dimensional}}$$

⊞ Implementation of Arrays:

- No way to precompute the address of the array element to be accessed by a reference, such as:
  $\underline{\text{, List [K] ,}}$

- At compile time, the code to allow accessing array elements must be generated.

- At run time, the allowed code is executed to produce element address.

The access function for List [K] is of the form:

address (list [K]) = address (list [0]) + K*element-size

↳ single-dimensioned array is implemented as a list of adjacent memory cells.

- First part address (list [0]) is constant
- Second part is variable K*element-size.

If element type is statically bound and storage is statically bound to storage, then

the constant part can be computed before run-time.

However, the addition and multiplication must be done during run time.

One can generalize the lower-bound:

$$\text{address}(\text{list}[K]) = \text{address}(\text{list}[\text{lower-bound}]) + ((K - \text{lower\_bound}) * \text{element\_size})$$

↳ Access function

↳ Necessary for address calculation to access the array elements.

⊞ For the defined arrays, we need compile time descriptor :

Array
Element type
Index type
Index lower bound
Index upper bound
Address

If run-time checking of index ranges is not done, and the attributes are all static, then only the access function is required during execution.

In general, if any entries in the descriptor is dynamically bound, then the descriptor should be present at the run-time.