# DATA TYPE

## TODAY'S AGENDA

- INTRODUCTION TO DATA TYPES
- PRIMITIVE DATA TYPES
- CHARACTER STRING TYPES
- ENUMERATION TYPES
- ARRAY TYPES
- ASSOCIATIVE ARRAYS
- RECORD TYPES
- TUPLE TYPES

## INTRODUCTION

- A DATA TYPE DEFINES A COLLECTION OF DATA OBJECTS AND A SET OF PREDEFINED OPERATIONS ON THOSE OBJECTS
- A DESCRIPTOR IS THE COLLECTION OF THE ATTRIBUTES OF A VARIABLE
- AN OBJECT REPRESENTS AN INSTANCE OF A USER-DEFINED (ABSTRACT DATA) TYPE
- ONE DESIGN ISSUE FOR ALL DATA TYPES: WHAT OPERATIONS ARE DEFINED AND HOW ARE THEY SPECIFIED?

## PRIMITIVE DATA TYPES

- Almost all programming languages provide a set of primitive data types.
- Primitive data types: Those not defined in terms of other data types.
- Some primitive data types are merely reflections of the hardware.
- Others require only a little non-hardware support for their implementation.

## PRIMITIVE DATA TYPE: INTEGER

- Almost always an exact reflection of the hardware so the mapping is trivial
- There may be as many as eight different integer types in a language
- Java's signed integer sizes: byte, short, int, long
- C and Python uses int

## LOOK AT THE EXAMPLES

C

```c
#include<stdio.h>

int main(){

 int x = 4;
 int y = -3;

 return 0;
}
```

Java

```java
public class Main {
  public static void
main(String[] args) {

    byte a = -128;
    byte b = 127;
    short c = 32767;
    int number = 23;
    long verybignumber =
922337203;

  }
}
```

Python

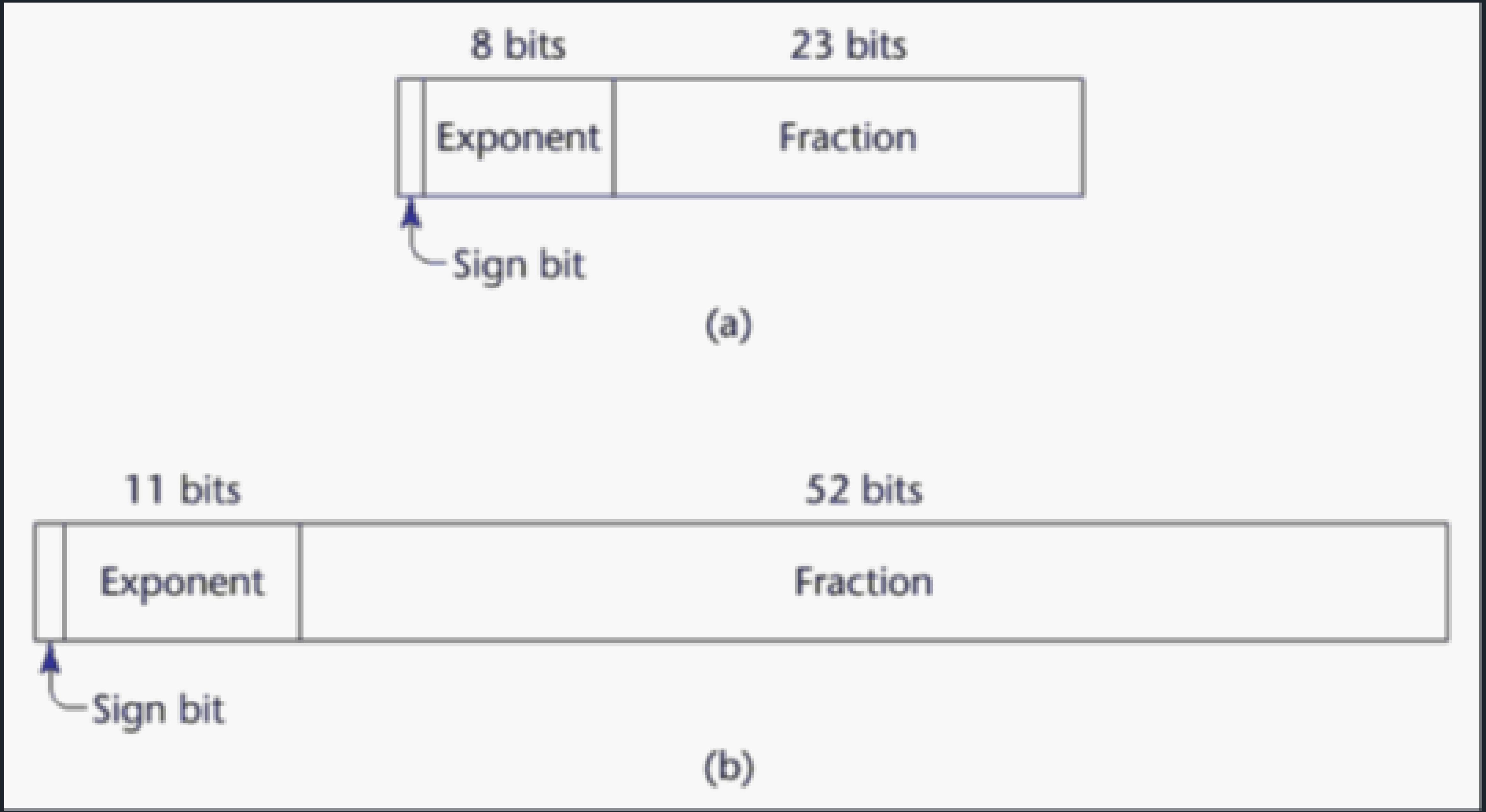
```python
x = 1

print(x)
print(type(x))
```

## PRIMITIVE DATA TYPE: FLOATING POINT

- Model real numbers, but only as approximations
- Languages for scientific use support at least two floating-point types (e.g., float and double; sometimes more
- Usually exactly like the hardware, but not always
- Floating-point arithmetic is a systematic approximation of real arithmetic (IEEE 745, 2008)

## IEEE STANDARD FOR FLOATING POINT ARITHMETIC:

## IEEE 754



## IEEE 754: ★ PROTOCOL

Sign of Mantissa (Sign Bit)

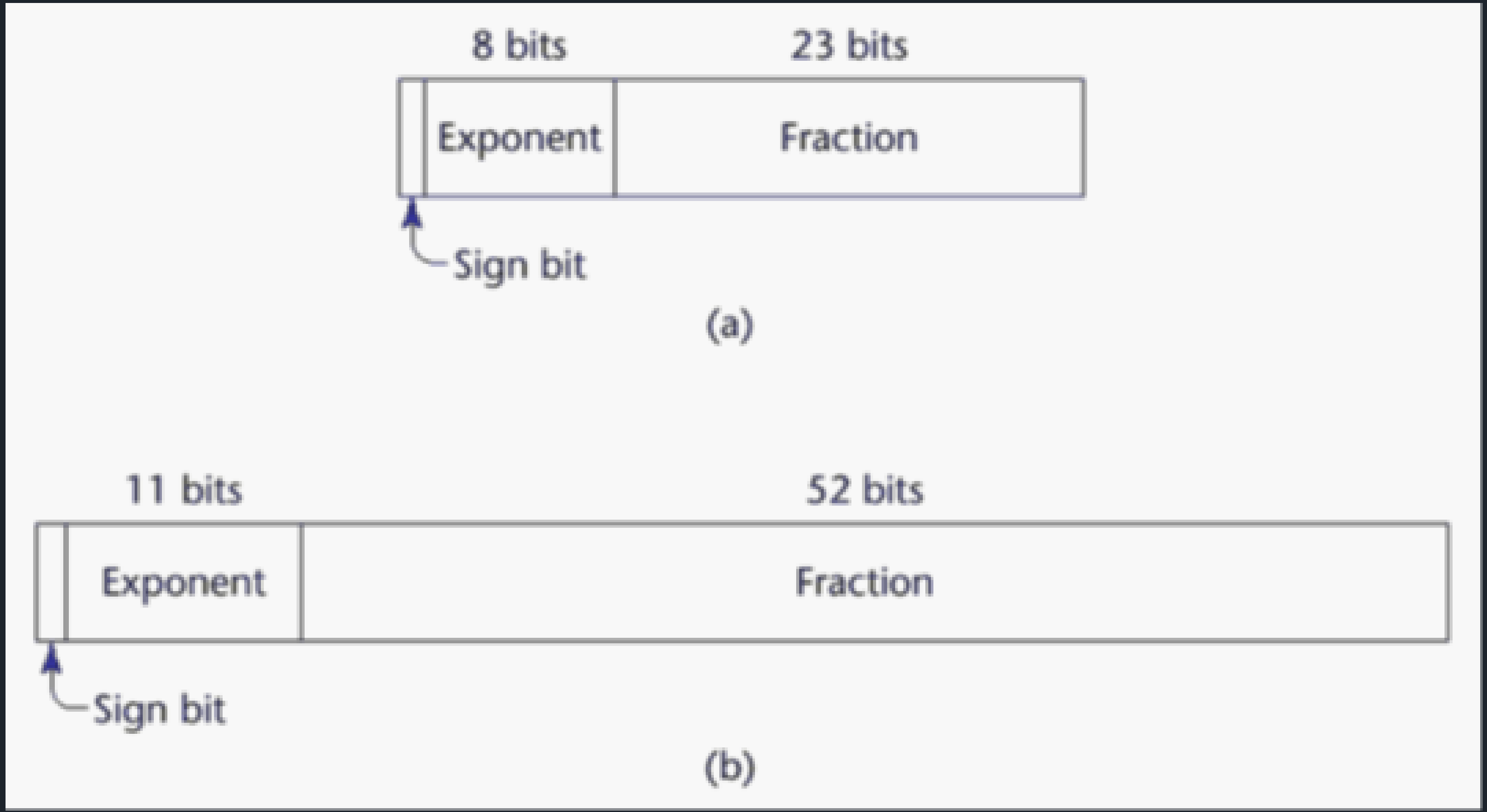
- 0 for possitive
- 1 for negative

Biased Exponent

- 8 bits (single precision)
- 11 bits (double precision)
- +127 for single precision
- +1023 for double precision

Normalized Mantissa (Fraction)

- 23 bits (single precision)
- 52 bits (double precision)



## IEEE 754:

## SPECIAL VALUES

| VALUE | EXPONENT | MANTISA |
|---|---|---|
| ZERO | 0 | 0 |
| DENORMALIZED (SUBNORMAL NUMBER) | 0 | !0 |
| INFINITE | 255 | 0 |
| NaN | 255 | !0 |

## IEEE 754:

## CONVERSION ★

1. Find the sign bit of the IEEE 754 standard
2. Express the original number in base-2 scientific notation
   a. Convert original number to binary
   b. Place the decimal point after the most significant bit
   c. The number of places moved will be the exponent of the base.
3. Find the exponent of the IEEE 754 standard
   a. exponent = n + 127
   b. where n is the exponent of the base
4. Place the mantissa as the fraction part of the IEEE 754 standard

## PRIMITIVE DATA TYPE: FLOATING POINT

- The collecction of values that can be represented by a floating-point type is defined in terms of precision and range

Precision - is the accuracy of the fractional part of a value measured as the number of bits

Range - is a combination of the range of fractions and, more important, the range of exponents

## LOOK AT THE EXAMPLES

| C | Java | Python |
|---|---|---|
| `#include <stdio.h>`<br><br>`int main() {`<br>`  float myFloatNum = 3.5;`<br>`  double myDoubleNum = 19.99;`<br><br>`  return 0;`<br>`}` | `public class Main {`<br><br>`  public static void main(String[] args) {`<br><br>`    float myNumA = 5.75f;`<br>`    double myNumB = 19.99d;`<br>`  }`<br><br>`}` | `x = 1.76` |

## PRIMITIVE DATA TYPE: COMPLEX

- Some languages support a complex type, e.g., C99, Fortran, and Python
- Each value consists of two floats, the real part and the imaginary part
- Literal form (in Python):
    - (7 + 3j), where 7 is the real part and 3 is the imaginary part

## LOOK AT THE EXAMPLES

C (Non-primitive)

```c
#include <stdio.h>
#include <complex.h>

int main(void)
{
    double complex z1 = I * I;
}
```

Python

```python
z = 2j
```

## PRIMITIVE DATA TYPE: DECIMAL

- For business applications (money)
  - Essential to COBOL
  - C# offers a decimal data type
- Store a fixed number of decimal digits, in coded form (BCD)
- Advantage: accuracy
- Disadvantages: limited range, wastes memory

## PRIMITIVE DATA TYPE: BOOLEAN

- Simplest of all
- Range of values: two elements, one for "true" and one for "false"
- Could be implemented as bits, but often as bytes.
    - Advantage: readability

## LOOK AT THE EXAMPLES

Java

```java
public class Main {
  public static void
main(String[] args) {

    boolean isJavaFun = true;
    boolean doesCHaveBool =
false;

  }
}
```

Python

```python
x = True
y = False
```

## PRIMITIVE DATA TYPE: CHARACTER

- Stored as numeric codings
- Most commonly used coding: ASCII
- An alternative, 16-bit coding: Unicode (UCS-2)
    - Includes characters from most natural languages
    - Originally used in Java
    - Now supported by many languages
- 32-bit Unicode (UCS-4)
    - Supported by Fortran, starting with 2003

## LOOK AT THE EXAMPLES

| C | Java | Python |
|---|---|---|
| `#include<stdio.h>`<br><br>`int main(){`<br><br>` char letter = 'a';`<br>` char c = 'B';`<br><br>` return 0;`<br>`}` | `public class Main {`<br>`  public static void main(String[] args) {`<br><br>`    char myVar1 = 'B';`<br><br>`  }`<br>`}` | • Does not have a character data type<br>• Uses a single character string for characters<br><br>`x = "P"` |

## CHARACTER STRING TYPES

- Values are sequences of characters
- Design issues:
    - Is it a primitive type or just a special kind of array?
    - Should the length of strings be static or dynamic?

# LOOK AT THE EXAMPLES

| C (via Character Array) | **Java (via String Class)** | **Python** |
|---|---|---|
| `#include<stdio.h>`<br><br>`int main(){`<br><br>`char ch[] = "abcd";`<br><br>`printf("%s", ch);`<br>`puts(ch);`<br><br>` return 0;`<br>`}` | `public class Main {`<br>`  public static void`<br>`main(String[] args) {`<br><br>`    String ch = "abcd";`<br><br><br>` System.out.printf("%s`<br>`", ch);`<br><br>`  }`<br>`}` | `ch = "abcd"`<br><br>`print(ch)`<br>`print(type(ch))` |

## CHARACTER STRING TYPES: OPERATION

- TYPICAL OPERATIONS:
  - ASSIGNMENT AND COPYING
  - COMPARISON (=, >, ETC.)
  - CATENATION
  - SUBSTRING REFERENCE
  - PATTERN MATCHING

## CHARACTER STRING TYPES: IN CERTAIN LANGUAGES

- C AND C++
  - NOT PRIMITIVE
  - USE CHAR ARRAYS AND A LIBRARY OF FUNCTIONS THAT PROVIDE OPERATIONS
- SNOBOL4 (A STRING MANIPULATION LANGUAGE)
  - PRIMITIVE
  - MANY OPERATIONS, INCLUDING ELABORATE PATTERN MATCHING
- FORTRAN AND PYTHON
  - PRIMITIVE TYPE WITH ASSIGNMENT AND SEVERAL OPERATIONS
- JAVA (AND C#, RUBY, AND SWIFT)
  - PRIMITIVE VIA THE STRING CLASS
- PERL, JAVASCRIPT, RUBY, AND PHP
  - PROVIDE BUILT-IN PATTERN MATCHING, USING REGULAR EXPRESSION

## LOOK AT THE EXAMPLES

| C* | Java* | Python |
|---|---|---|
| ASSIGNMENT & COPYING<br>• =<br>• strcpy() | ASSIGNMENT AND COPYING<br>• = | ASSIGNMENT AND COPYING<br>• = |
| COMPARISON<br>• strcmp() | COMPARISON<br>• compareTo()<br>• == | COMPARISON<br>• ==, <, <=, >, >=, != |
| CATENATION<br>• strcat() | CATENATION<br>• +, +=<br>• concat() | CATENATION<br>• +<br>• len() |
| SUBSTRING REFERENCE<br>• strncpy() | SUBSTRING REFERENCE<br>• substring() | SUBSTRING REFERENCE<br>• varname[start:end] |
| PATTRN MATCHING<br>• :( | PATTERN MATCHING<br>• contains()<br>• matches() | PATTERN MATCHING<br>• find()<br>• rfind() |
| OTHERS<br>• strlen()<br>• strcasecmp()<br>• ... | OTHERS<br>• length()<br>• ... | OTHERS<br>• len()<br>• split()<br>• ... |

## CHARACTER STRING LENGTH OPTION

- STATIC: PYTHON, JAVA'S STRING CLASS
- LIMITED DYNAMIC LENGTH: C AND C++
- IN THESE LANGUAGES, A SPECIAL CHARACTER IS USED TO INDICATE THE END OF A STRING'S CHARACTERS, RATHER THAN MAINTAINING THE LENGTH
- DYNAMIC (NO MAXIMUM): SNOBOL4, PERL, JAVASCRIPT

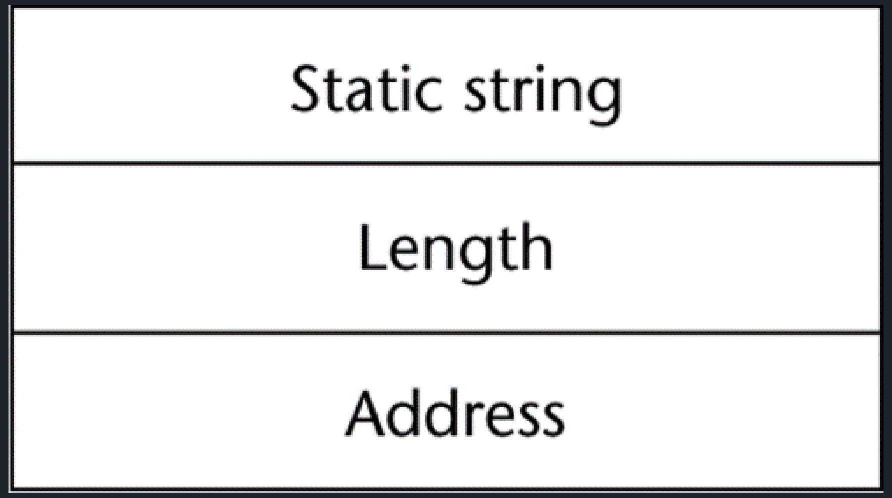
## CHARACTER STRING TYPE EVALUATION

- AID TO WRITABILITY
- AS A PRIMITIVE TYPE WITH STATIC LENGTH, THEY ARE INEXPENSIVE TO PROVIDE--WHY NOT HAVE THEM?
- DYNAMIC LENGTH IS NICE, BUT IS IT WORTH THE EXPENSE?

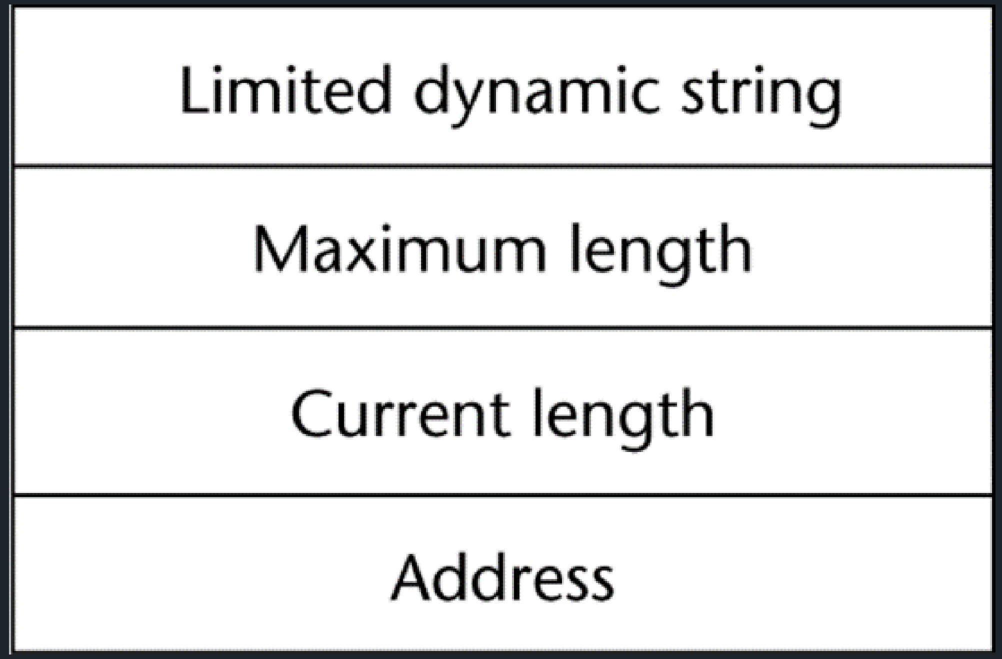
## CHARACTER STRING IMPLEMENTAION

- STATIC LENGTH: COMPILE-TIME DESCRIPTOR
- LIMITED DYNAMIC LENGTH: MAY NEED A RUN-TIME DESCRIPTOR FOR LENGTH (BUT NOT IN C AND C++)
- DYNAMIC LENGTH: NEED RUN-TIME DESCRIPTOR; ALLOCATION/DEALLOCATION IS THE BIGGEST IMPLEMENTATION PROBLEM

## COMPILE - AND RUN-TIME DESCRIPTORS



Compile-time descriptor for static strings



Run-time descriptor for limited dynamic strings

## ENUMERATION TYPES

- All possible values, which are named constants, are provided in the definition
- C# example
  - enum days {mon, tue, wed, thu, fri, sat, sun};
- Design issues
  - Is an enumeration constant allowed to appear in more than one type definition, and if so, how is the type of an occurrence of that constant checked?
  - Are enumeration values coerced to integer?
  - Any other type coerced to an enumeration type?

## LOOK AT THE EXAMPLES

**C**

```c
#include<stdio.h>


enum week{Mon, Tue, Wed,
Thur, Fri, Sat, Sun};


int main() {
    enum week day;
    day = Wed;
    printf("%d",day);
    return 0;
}
```

**Java**

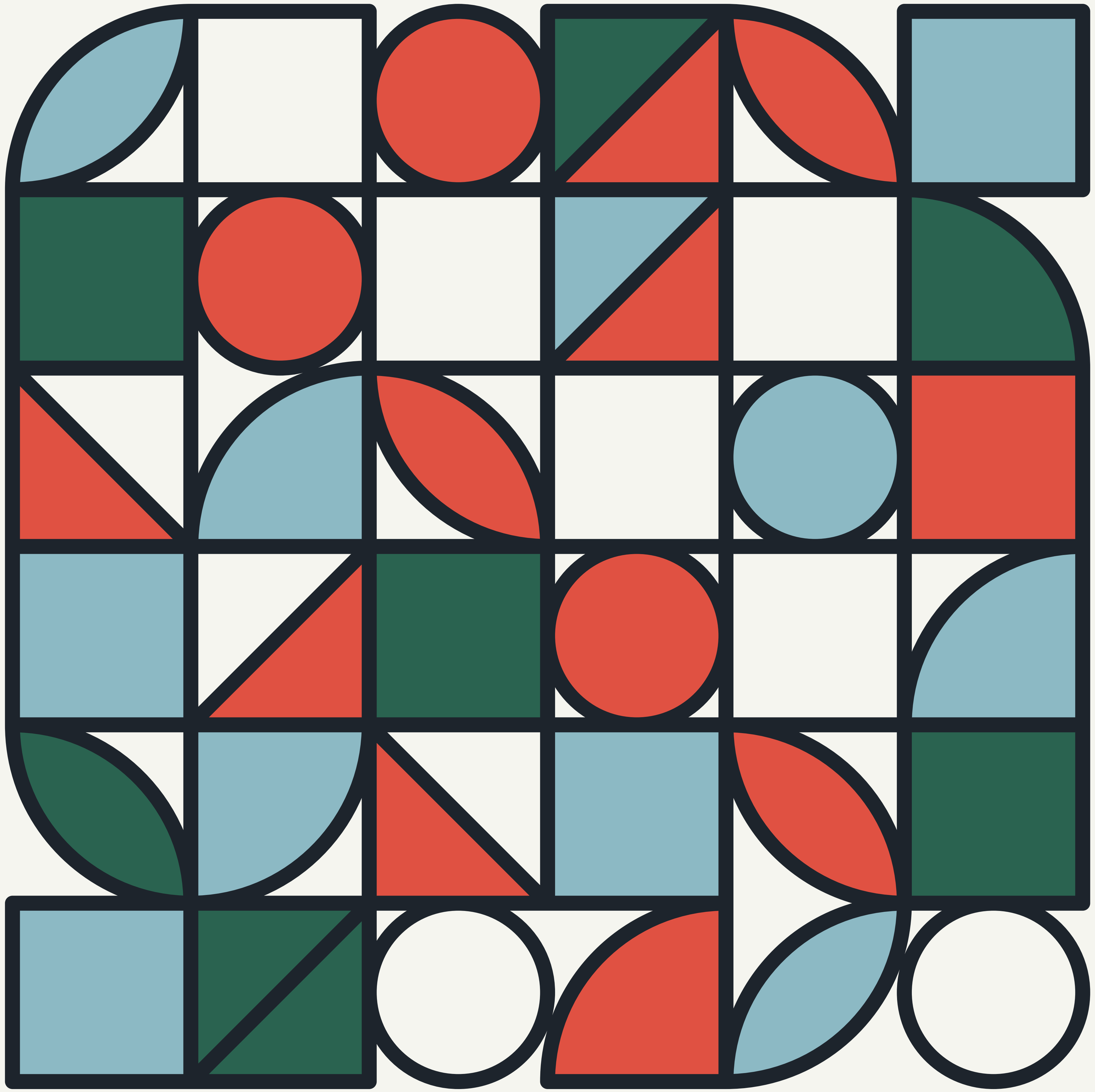
```java
enum Week {
    Mon, Tue, Wed, Thur,
Fri,
    Sat, Sun;
}

public class Test {
   public static void
main(String[] args){
        Week day =
Week.Wed;

 System.out.println(day);
   }
}
```

**Python**

```python
from enum import Enum

class Season(Enum):
 SPRING = 1
 SUMMER = 2
 AUTUMN = 3
 WINTER = 4

print("The enum member
associated with value 2 is :
", Season(2).name)

print("The enum member
associated with name AUTUMN
is : ",
Season['AUTUMN'].value)
```

## EVALUATION OF ENUMERATION TYPES

- AID TO READABILITY, E.G., NO NEED TO CODE A COLOR AS A NUMBER
- AID TO RELIABILITY, E.G., COMPILER CAN CHECK:
  - OPERATIONS
  - NO ENUMERATION VARIABLE CAN BE ASSIGNED A VALUE OUTSIDE ITS DEFINED RANGE
  - C#, F#, SWIFT, AND JAVA 5.0 PROVIDE BETTER SUPPORT FOR ENUMERATION THAN C++ BECAUSE ENUMERATION TYPE VARIABLES IN THESE LANGUAGES ARE NOT COERCED INTO INTEGER TYPES



## ARRAY TYPES

An array is a homogeneous aggregate of data elements in which an individual element is identified by its position in the aggregate, relative to the first element.

## ARRAY DESIGN ISSUES

- What types are legal for subscripts?
- Are subscripting expressions in element references range checked?
- When are subscript ranges bound?
- When does allocation take place?
- Are ragged or rectangular multidimensional arrays allowed, or both?
- What is the maximum number of subscripts?
- Can array objects be initialized?
- Are any kind of slices supported?

## ARRAY INDEXING

- Indexing (or subscripting) is a mapping from indices to elements

  array_name (index_value_list) -> an element
- Index Syntax
  - Fortran and Ada use parentheses
    - Ada explicitly uses parentheses to show uniformity between array references and function calls because both are mappings
  - Most other languages use brackets

## ARRAY INDEXING (SUBSCRIPTING) TYPES

- FORTRAN, C: integer only
- Java: integer types only
- Index range checking
  - C, C++, Perl, and Fortran do not specify range checking
  - Java, ML, C# specify range checking

## SUBCRIPT BINDING AND ARRAY CATEGORIES

- *Static*: subscript ranges are statically bound and storage allocation is static
    - Advantage: efficiency (no dynamic allocation)
- *Fixed stack-dynamic*: subscript ranges are statically bound, but the allocation is done at declaration time
    - Advantage: space efficiency

## SUBCRIPT BINDING AND ARRAY CATEGORIES

- *Fixed heap-dynamic*: similar to fixed stack-dynamic: storage binding is dynamic but fixed after allocation (i.e., binding is done when requested and storage is allocated from heap, not stack)
- Heap-dynamic: binding of subscript ranges and storage allocation is dynamic and can change any number of times
    - Advantage: flexibility (arrays can grow or shrink during program execution)

## SUBCRIPT BINDING AND ARRAY CATEGORIES

- *C and C++ arrays that include static modifier are static*
- *C and C++ arrays without static modifier are fixed stack-dynamic*
- *C and C++ provide fixed heap-dynamic arrays*
- *C# includes a second array class ArrayList that provides fixed heap-dynamic*

## SUBCRIPT BINDING AND ARRAY CATEGORIES

- *C and C++ arrays that include static modifier are static*
- *C and C++ arrays without static modifier are fixed stack-dynamic*
- *C and C++ provide fixed heap-dynamic arrays*
- *C# includes a second array class ArrayList that provides fixed heap-dynamic*
- *Perl, JavaScript, Python, and Ruby support heap-dynamic arrays*

## ARRAY INITIALIZATION

- *Some language allow initialization at the time of storage allocation*
    - *C, C++, Java, Swift, and C#*
    - *C# example:*

      *int list [] = {4, 5, 7, 83}*
    - *Character strings in C and C++*

      *char name [] = "freddie";*
    - *Arrays of strings in C and C++*

      *char *names [] = {"Bob", "Jake", "Joe"];*
    - *Java initialization of String objects*

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

## LOOK AT THE EXAMPLES

C

```c
#include<stdio.h>



int main() {
 int list[] =
{1,2,3,4,5};
 int lists[6] =
{5,4,3,2,1};
 return 0;
}
```

Java

```java
class arrrrr {
    public static void
main(String[] args)
    {
       int[] arr;
       arr = new int[5];

       arr[0] = 10;
       arr[1] = 20;
       arr[2] = 30;
       arr[3] = 40;
       arr[4] = 50;
       for (int i = 0; i <
arr.length; i++)

 System.out.println("Element at
          index " + i + " : " +
arr[i]);
    }
}
```

Python

```python
arr=[]
arr = [0 for i in range(5)]
print(arr)


import numpy as np
arr = np.empty(10,
dtype=object)
print(arr)


arr_num = [0] * 5
print(arr_num)

arr_str = ['P'] * 10
print(arr_str)
```

## HETEROGENOUS ARRAYS

- A *heterogeneous array* is one in which the elements need not be of the same type
- Supported by Perl, Python, JavaScript, and Ruby

## HETEROGENOUS ARRAYS INITIALIZATION

- C-based languages
  - int list [] = {1, 3, 5, 7}
  - char *names [] = {"Mike", "Fred", "Mary Lou"};
- Python
  - List comprehensions

```
list = [x ** 2 for x in range(12) if x % 3 == 0]
  puts [0, 9, 36, 81] in list
```

## LOOK AT THE EXAMPLES

| C | Java | Python |
|---|---|---|
| The programming language C does not support heterogenous arrays. | The programming language Java does not support heterogenous arrays. | `test_list = ['gfg', 1, 2, 'is', 'best']` |

## ARRAY OPERATIONS

- An array operation is one that operates on an array as a unit. The most common array operations are assignment, catenation, comparison for equality and inequality, and slices

## ARRAY OPERATIONS

- APL provides the most powerful array processing operations for vectors and matrixes as well as unary operators (for example, to reverse column elements)
- Python's array assignments, but they are only reference changes. Python also supports array catenation and element membership operations
- Ruby also provides array catenation

## LOOK AT THE EXAMPLES

Python

```
import numpy as geek

arr1 = geek.array([[2, 4], [6, 8]])
arr2 = geek.array([[3, 5], [7, 9]])

gfg = geek.concatenate((arr1, arr2), axis = None)

print (gfg)
```

## RECTANGULAR AND JAGGED ARRAYS

- A rectangular array is a multi-dimensioned array in which all of the rows have the same number of elements and all columns have the same number of elements
- A jagged matrix has rows with varying number of elements (supproted in C# and java)
- Possible when multi-dimensioned arrays actually appear as arrays of arrays

## LOOK AT THE EXAMPLES

C

```c
#include<stdio.h>

int main(void)
{
    int x[3][2] =
{{0,1}, {2,3}, {4,5}};

    return (0);
}
```

Java

```java
import java.io.*;
public class GFG {
    public static void
main(String[] args)
    {

        int[][] arr =
new int[10][20];
        arr[0][0] = 1;

}
```

Python

```python
a = [[2, 4, 6, 8, 10],
[3, 6, 9, 12, 15], [4,
8, 12, 16, 20]]




m = 4
n = 5
a = [[0 for x in
range(n)] for x in
range(m)]
```

## SLICES

- A slice is some substructure of an array; nothing more than a referencing mechanism
- Slices are only useful in languages that have array operations

## SLICES EXAMPLES

- Python

  ```
  vector = [2, 4, 6, 8, 10, 12, 14, 16]
  mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
  ```

  vector (3:6) is a three-element array

  mat[0][0:2] is the first and second element of the first row of mat
- Ruby supports slices with the slice method

  list.slice(2, 2) returns the third and fourth elements of list

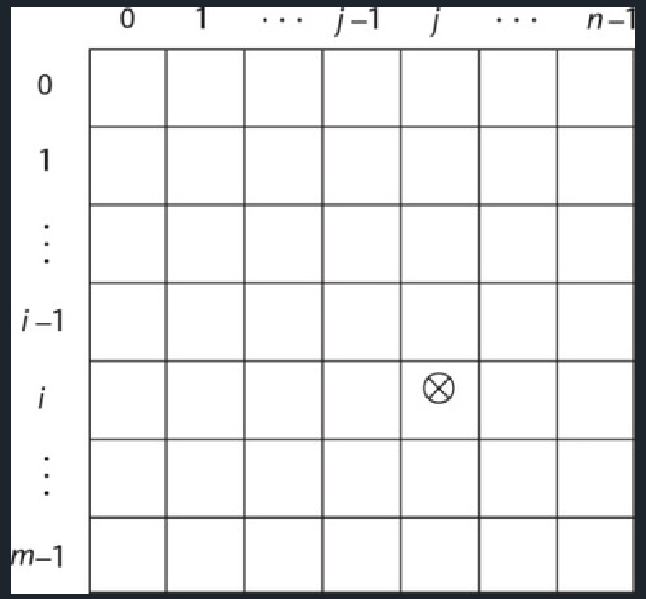
## LOOK AT THE EXAMPLES

Python

```
Lst = [50, 70, 30, 20, 90, 10, 50]



print(Lst[1:5])
```

## IMPLEMENTAION OF ARRAYS

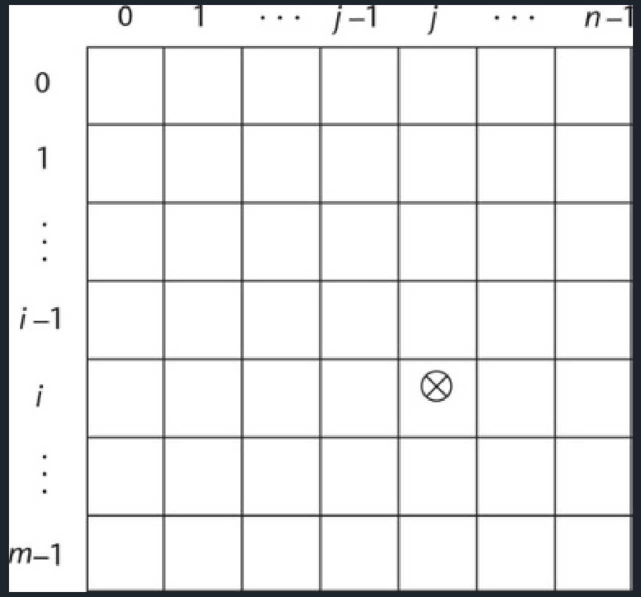
- Access function maps subscript expressions to an address in the array



# IMPLEMENTAION OF ARRAYS

- Access function maps subscript expressions to an address in the array
- Access function for single-dimensioned arrays:

  address(list[k]) = address (list[lower_bound])
  + ((k-lower_bound) * element_size)



## LOOK AT THE EXAMPLES

Example

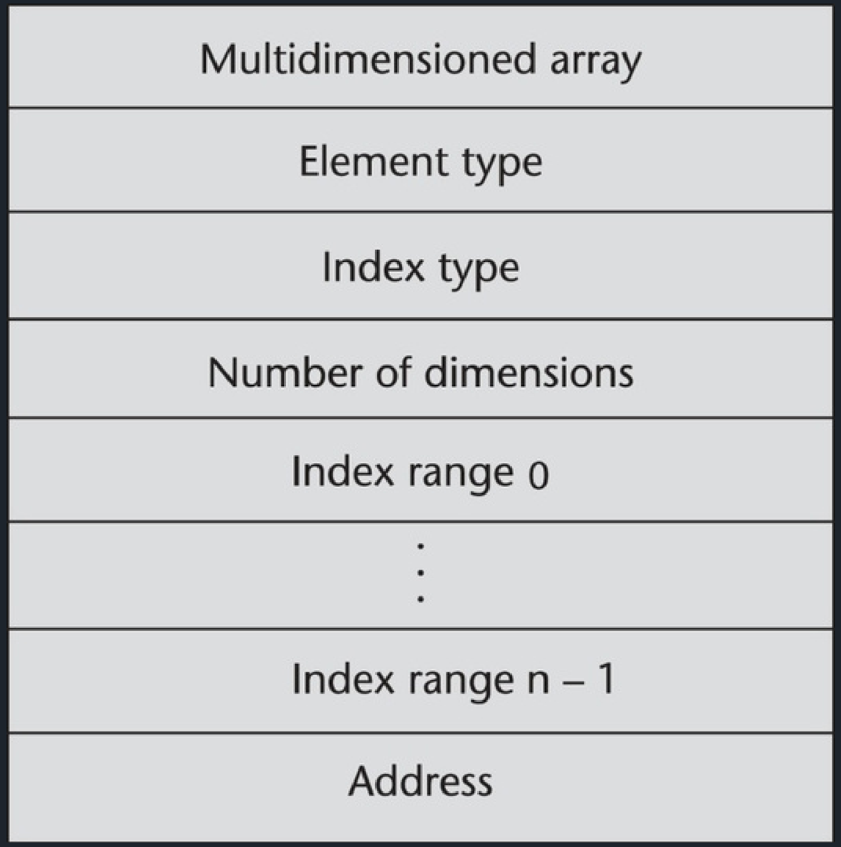
Given the base address of an array A[1300 ............ 1900] as 1020 and the size of each element is 2 bytes in the memory, find the address of A[1700].

address(list[k]) = address (list[lower_bound])
+ ((k-lower_bound) * element_size)

address(A[1700]) = 1020 + ((1700 - 1300) * 2)
= 1820

## ACCESSING A MULTI-DIMENTIONAL ARRAYS

- Two common ways:
  - Row major order (by rows) – used in most languages
  - Column major order (by columns) – used in Fortran
  - A compile-time descriptor for a multidimensional array

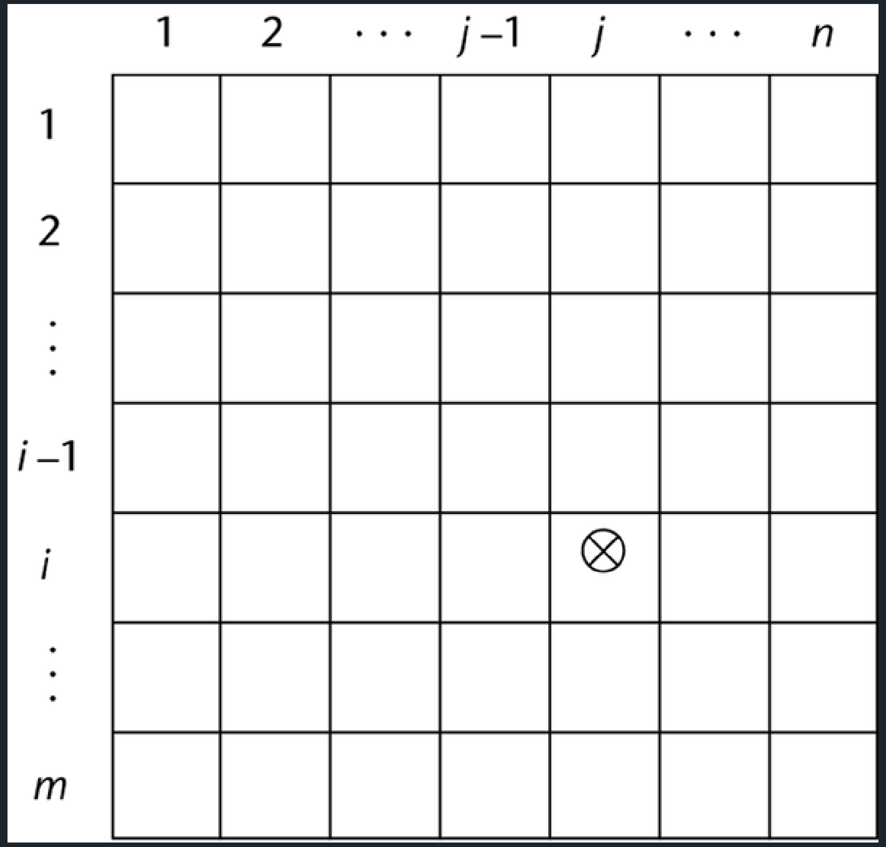


| Multidimensioned array |
|---|
| Element type |
| Index type |
| Number of dimensions |
| Index range 0 |
| ⋮ |
| Index range n – 1 |
| Address |

## LOCATING AN ELEMENT IN A MULTI-DIMENSIONED ARAY

- General format

  Location (a[I,j]) = address of a [row_lb,col_lb] + (((I - row_lb) * n) + (j - col_lb)) * element_size



## LOOK AT THE EXAMPLES

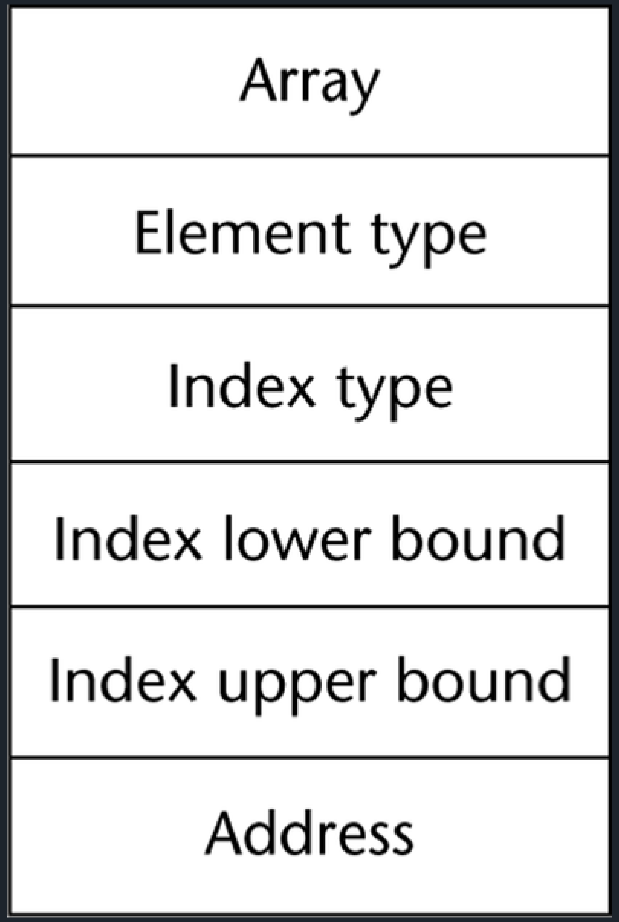
Example

Given an array, arr[1.........10][1.........15] with base value 100 and the size of each element is 1 Byte in memory. Find the address of arr[8][6] with the help of row-major order.

Address (mat[I,j]) = address of a [row_lb,col_lb] + (((I - row_lb) * n) + (j - col_lb)) * element_size

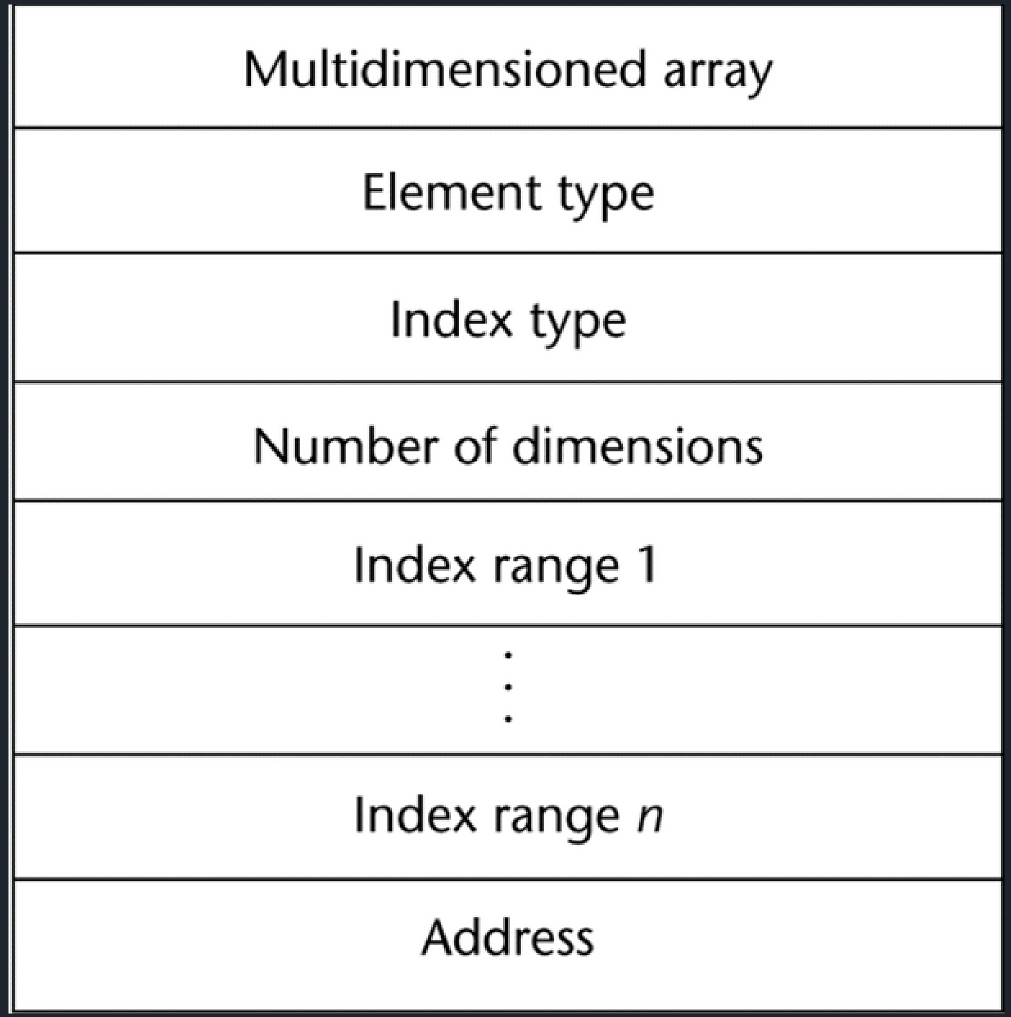
Address (arr[i][j]) = 100 + (((8 - 1) * 6) + (6-1)) * 1
= 147

## COMPILE-TIME DESCRIPTORS



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

**Single-dimensioned array**



| Multidimensioned array |
|---|
| Element type |
| Index type |
| Number of dimensions |
| Index range 1 |
| ⋮ |
| Index range *n* |
| Address |

**Multidimensional array**

## ASSOCIATIVE ARRAYS

- An associative array is an unordered collection of data elements that are indexed by an equal number of values called keys
  - User-defined keys must be stored
- Design issues:
  - What is the form of references to elements?
  - Is the size static or dynamic?
- Built-in type in Perl, Python, Ruby, and Swift

## ASSOCIATIVE ARRAYS

- Hashes (Perl) - In implementation, their elements are stored and retrieved with hash functions.
- Dictionaries (Python) - Similar to Perl, except the values are all references to objects.

## LOOK AT THE EXAMPLES

| Java | Python |
|---|---|
| see below | see below |

Java

```java
import java.util.HashMap;

public class Main {
  public static void main(String[] args) {

    HashMap<String, String> capitalCities = new
HashMap<String, String>();

    capitalCities.put("England", "London");
    capitalCities.put("Germany", "Berlin");
    capitalCities.put("Norway", "Oslo");
    capitalCities.put("USA", "Washington DC");


 System.out.println(capitalCities.get("USA"));
  }
}
```

Python

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

## ASSOCIATIVE ARRAYS IN PERL

- Names begin with %; literals are delimited by parentheses

```
%hi_temps = ("Mon" => 77, "Tue" => 79, "Wed"
 => 65, ...);
```

- Subscripting is done using braces and keys

```
$hi_temps{"Wed"} = 83;
```

  - Elements can be removed with **delete**

```
delete $hi_temps{"Tue"};
```

## RECORD TYPES

- A record is a possibly heterogeneous aggregate of data elements in which the individual elements are identified by names
- Design issues:
    - What is the syntactic form of references to the field?
    - Are elliptical references allowed

## LOOK AT THE EXAMPLES

C

```
#include <stdio.h>
#include <string.h>

typedef struct {
   char name[50];
   int citNo;
   float salary;
} person;

int main() {
   person person1;
   strcpy(person1.name, "George    Jones");
   person1.citNo = 1928;
   person1.salary = 2500;

   return 0;
}
```

Java

```
public class PersonClass {

   private final String name;
   private final int citNo;

   public PersonClass (String name,
int citNo) {

   this.name = name;
   this.citNo = citNo;

}
```

## DEFINITION OF RECORDS IN COBOL

- COBOL uses level numbers to show nested records; others use recursive definition

```
01 EMP-REC.
  02 EMP-NAME.
    05 FIRST PIC X(20).
    05 MID PIC X(10).
    05 LAST PIC X(20).
  02 HOURLY-RATE PIC 99V99.
```

## REFERENCES TO RECORDS

- Record field references
  1. COBOL

     field_name OF record_name_1 OF ... OF record_name_n
  2. Others (dot notation)

     record_name_1.record_name_2. ...

     record_name_n.field_name
- Fully qualified references must include all record names
- Elliptical references allow leaving out record names as long as the reference is unambiguous, for example in COBOL

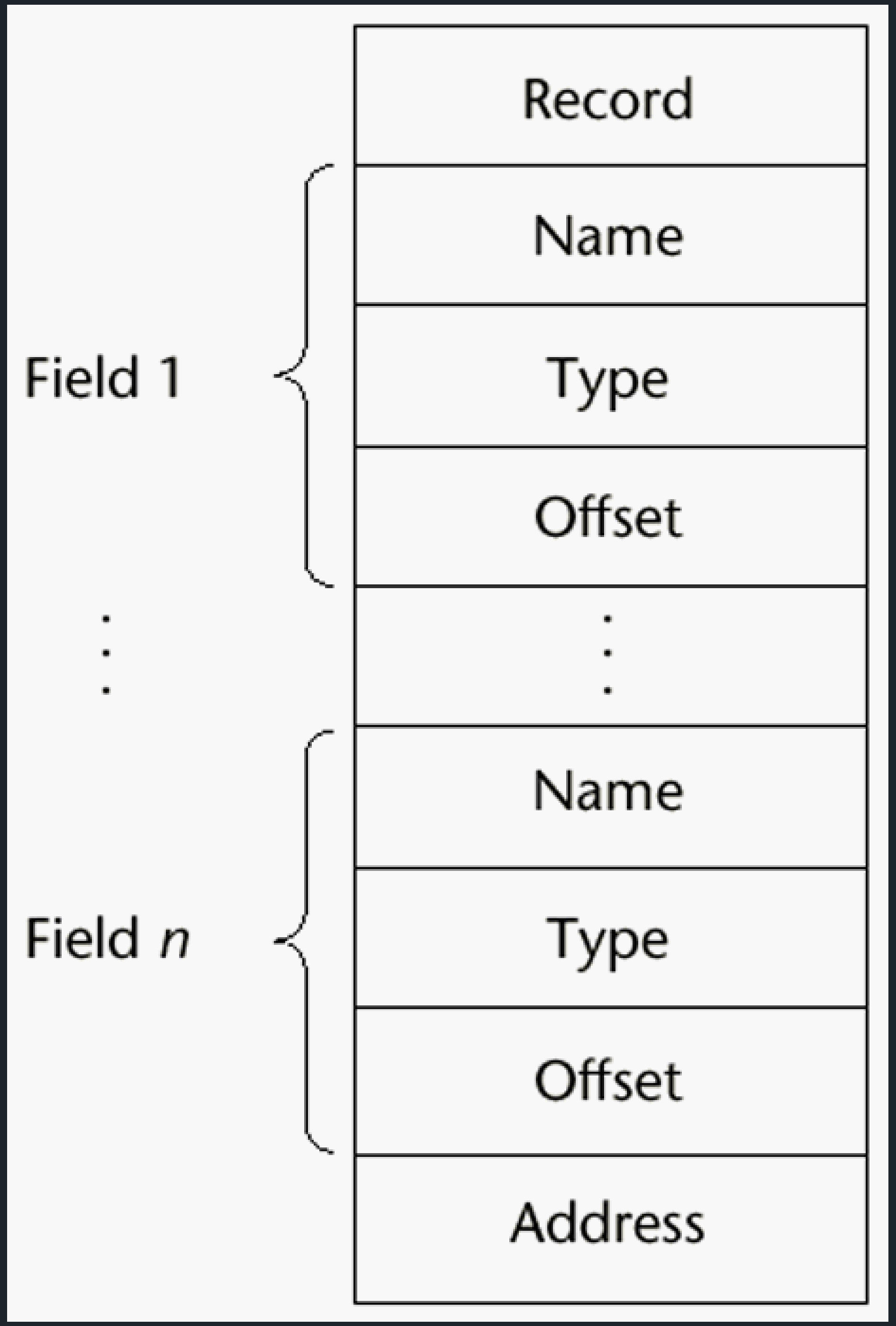
  FIRST, FIRST OF EMP-NAME, and FIRST of EMP-REC are elliptical references to the employee's first name

## EVALUTION AND COMPARISON TO ARRAYS

- Records are used when collection of data values is heterogeneous
- Access to array elements is much slower than access to record fields, because subscripts are dynamic (field names are static)
- Dynamic subscripts could be used with record field access, but it would disallow type checking and it would be much slower

# IMPLEMENTATION OF RECORD TYPE

Offset address relative to the beginning of the records is associated with each field



## TUPLE TYPES

- A tuple is a data type that is similar to a record, except that the elements are not named
- Used in Python, ML, and F# to allow functions to return multiple values
  - Python
    - Closely related to its lists, but immutable
    - Create with a tuple literal

      myTuple = (3, 5.8, 'apple')

      Reference with subscripts (begin at 1)

      Catenation with + and deleted with del

- ML

  val myTuple = (3,5.8, 'apple');

  -Access as follows:

  #1 (myTuple) is the first element

  -A new tuple type can be defined

  type intReal = int * real;

  ( the asterisk is just a separator)
- F#

  let tup = (3,5,7)

  let a,b,c = tup

  This assigns a tuple to a tuple pattern (a,b,c)

## THANK YOU!