Principles of Programming Languages
Lecture 16

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgment: parts of these slides are based on material by Diane Horton & Eric Joanis @ UoT

References: Scheme by Dybvig
PL Concepts and Constructs by Sethi
Concepts of PL by Sebesta
ML for the Working Prog. By Paulson
Prog. in Prolog by Clocksin and Mellish
PL Pragmatics by Scott

Components of an Imperative Language

- **Data types**
- **Variables, Operators, & Expressions**
- **Assignment construct**
- **Iteration construct**
- **Branching construct**
- **Function construct**
- **Container construct**

Data types: introduction

- **A language designer must provide a specific set of data types.**

- **Data types specification:**
  - Name
  - Implementation
  - Operations
  - Exceptions/errors

Data types: classification

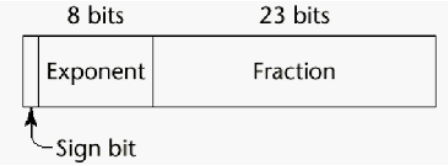
- **Primitive data types:**
  - Mostly Supported by hardware
  - Examples: integer, float,…

- **Structured data types(SDT):**
  - Constructed as aggregation of other primitive data types.
  - Either *language defined* data types or *user defined* data types

Data types: primitive

- **Integer**
  - Different sizes, hardware support
  - Examples:
    - Java: byte(8bit-signed), short(16-bit signed), int(32-bit signed),long(64-bit signed)
  - Implementation:
    - Most computers use twos complement

- **Floating-point**
  - Difficult to represent by finite number of binary digits
  - Examples:
    - Java: float(32bit-IEEE754), double(64bit-IEEE754)
  - Implementation:
    - Most computers use IEEE754



Data types: primitive cont'd

- **Decimal**
  - Stores fixed number of decimal digits, with decimal point at a fixed position.
  - Restricted range but more precise than floats.
  - Implementation:
    - Stored like character strings, using binary codes for digits (BCD)
    - If hardware support is not provided, simulate in software

- **Boolean**
  - ALGOL 60 is the first to introduce it.
  - Only C/C++: numeric expressions can be used as conditionals
  - Implementation:
    - Stored in the smallest efficiently addressable cell of memory

- **Character**
  - Implementation
    - Byte or word
    - ASCII or Unicode

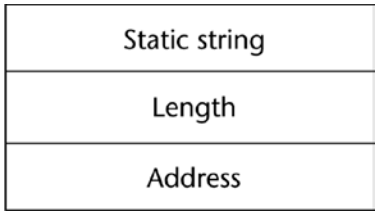
Structured Data types

- **Constructed as aggregation of other primitive data types.**

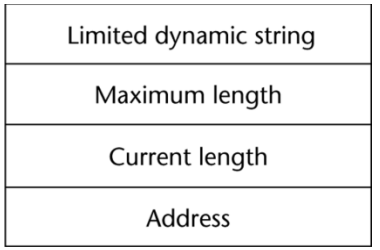
- **Strings**
- **Ordinal**
- **Arrays**
- **Associative Arrays**
- **Records**
- **Union**
- **Lists**

SDT: Strings

- **Strings**
  - ASCII or Unicode
  - Implementation:
    - Static or dynamic length?
    - Descriptors:
      - Compile-time descriptor for static strings



      - Run-time descriptor for dynamic strings



SDT: User-defined ordinal types

- **Unordered collections of user defined distinct values**

- **Classification:**
  - Enumeration:
    - All of the possible values of a variable are enumerated in the definition
    - E.g.

```
type DAYS is (Mon, Tue, Wed, Thu, Fri, Sat,  Sun);          // Ada
```

  - Subranges:
    - A contiguous subsequence of an ordinal type.
    - E.g.

```
type uppercase = 'A' .. 'Z';                                // Pascal
     index     = 1..100;

subtype WEEKDAYS is DAYS      range Mon..Fri;               // Ada
subtype INDEX    is INTEGER range 1..100;
```

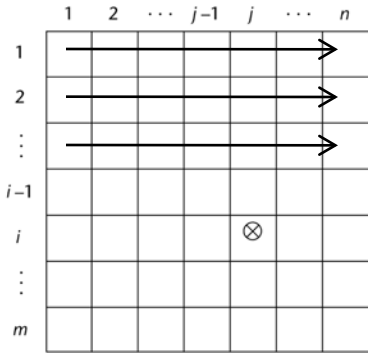
SDT: Arrays

- **Arrays:**
  - Homogenous arrays of elements in which each element is identified by its position, relative to the first element.
  - Contiguous structured data type
  - Operations:
    - Initialization
      - `int list[] = { 4,5,6,7 };` // C and C++
      - `int[] list = { 4,5,6,7 };` // Java
      - Not all languages all initialization (e.g. Pascal, Modula-2)

SDT: Arrays cont'd

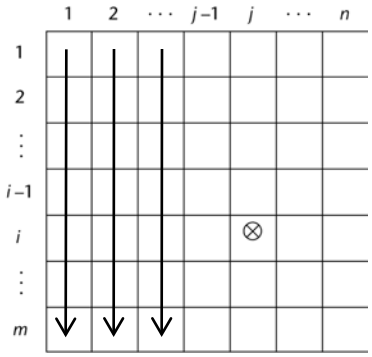
- Multidimensional arrays:
  - Row-major order (Pascal, C/C++, Java, Ada, Modula-2)
    - Layout as a sequence of consecutive rows, rightmost subscript varies fastest
    - E.g.

```
A[1,1],A[1,2],A[1,3]
```



  - Column-major order (Fortran, Basic)
    - Lay out as a sequence of consecutive columns, left most subscript varies fastest
    - E.g.

```
A[1,1],A[2,1],A[1,2]
```

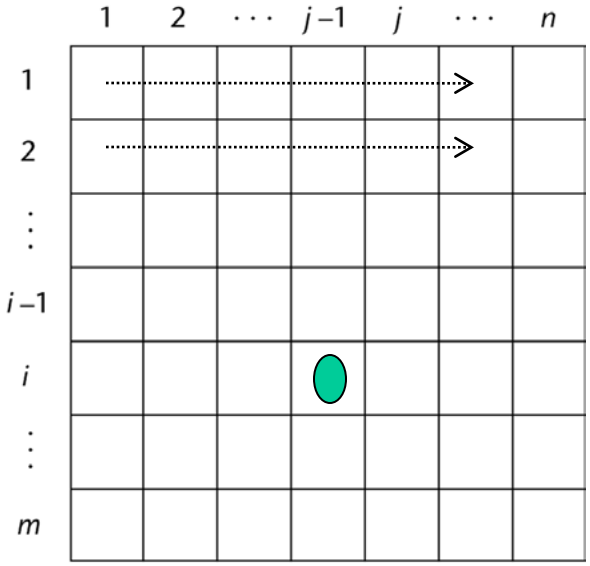


SDT: Arrays cont'd

- Multidimensional array referencing:

  - How do we compute address of $A[i_1][i_2]$?

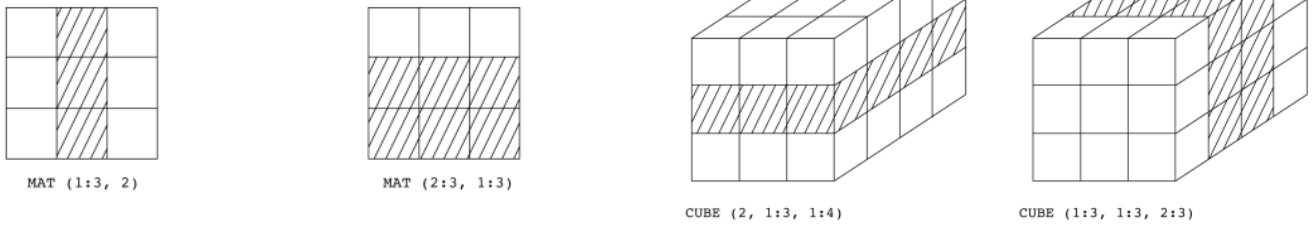
    Target_Address = Base_address
    + row_index *number _of _element
    * size_of_an_element
    + col_index * size_of_an_element



A[4][3] = 20 // A is array of 5x5, int is 1 byte
Element_Address = 1000 + (4 * 5 x 1) + 3 * 1=
= 1023

SDT: Arrays cont'd

- **Slices:**
  - A substructure of an array.
  - Note that this is not a new data type, it is just a mechanism for referencing part of an array
  - E.g.: Fortran, Ada, Python



  - Operations:
    - Reference
    - Assignment

SDT: Associative Arrays

- **An unordered collection of data elements that are indexed by an equal number of values called** *keys***.**

- **Each element of an associative array is in fact a pair of entities:** *key* **&** *value***.**

key1	val1
key2	val2
key3	val3
.....	
....	

- **Languages:**
  - Java (Map), Python (dictionary)

- **Implementation:**
  - Implemented by hash functions to speed the retrieval of the value.

SDT: Records

- **Records:**
  - Possibly heterogeneous aggregates where elements are referenced by name.
    - A record can have a function defined within it
    - Introduced by COBOL. C struct adopted it.
  - Examples:
    - COBOL

```
01 EMP-REC.
    05 EMP-NAME.
       10 FIRST       PICTURE IS X(20).
       10 LAST        PICTURE IS X(20).
    05 HRLY-RATE  PICTURE IS 99V99.
```

    - Ada

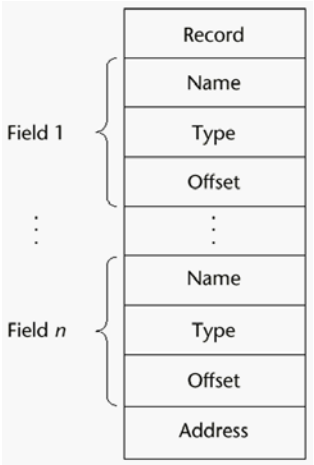
```
EMP_REC :
 record
   EMP_NAME :
    record
      FIRST :          STRING(1..20);
      LAST :           STRING(1..20);
    end record
   HRLY_RATE : FLOAT;
 end record
```

SDT: Records cont'd

- **Reference:**
  - COBOL:
    - Name the field and enclosing record(s)
    - E.g.: LAST OF EMP-NAME OF EMP-REC
  - Pascal:
    - Uses dot notation, fully qualified reference.
    - EMP_REC.EMP_NAME.LAST

- **Operations:**
  - Pascal, Modula-2, C, C++: assignment.
  - Ada: assignment and comparison
  - C: field reference and pointer assignment

- **Implementation:**
  - Fields are stored in adjacent memory cells.
  - Offset address associated with each field.



SDT: Sets

- **A set type is one whose variables can store unordered collections of distinct values from some ordinal type called its *base type*.**

- **Example:**
  - Modula-2

```
setype1 = set of  [blue,green,red];
setype2 = set of  [blue,red];
var  setvar1 = setype1;
```

  - Pascal

```
type colors     = (red , green , blue , yellow, orange, white);
colorset        = set of colors( );
var set1, set2 : colorset;

set1 := [red,blue,yellow,white];          // ok
set2 := [black,blue];                     // error?
```

- **Operations:**
  - *in* operator

```
if(var in set1) ….
```

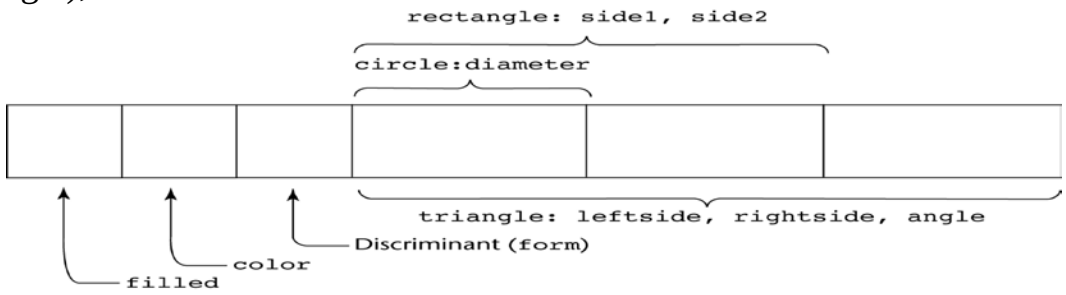
SDT: Union

- **Allow to store different type values at different times during program execution:**
  - Discriminated Union
    - Has a *tag field* or *discriminate* that tells the current type value
    - E.g.: Pascal, Modula-2, Ada
  - Free union:
    - No tag.
    - E.g.: C, C++

SDT: Union cont'd

- **Example:**

```
type shape = (circle, triangle, rectangle);
     colors = (red, green, blue);
     figure  = record
        filled: boolean;
        color:  colors;
        case form: shape of
            circle: (diameter: real);
            triangle: (leftside: integer; rightside:integer;angle:real);
            rectangle: (side1: integer; side2:integer)
        end;
var myfigure : figure;
case myfigure.form of
        circle:      begin
                        writeln("It is a circle", myfigure.diameter);
                     end;
        triangle:    begin
                        writeln("It is a traingle" , myfigure.leftside);
                     end;
        rectangle:  begin
                        writeln("It is a rectangle", myfigure.side1 );
                        end;
```



SDT: Lists

- **An ordered sequence of data structures.**
  - Usually does not have a fixed length.
  - Data type of each member may differ

- **Examples:**
  - Python

```
lst = (1,2,3,4)
print lst[0]
```

- **Implementation:**
  - Linked list storage management often used