# PRINCIPLES OF PROGRAMMING LANGUAGES

UNIT 2

# Presentation Outline

- Specification and Implementation of Elementary Data Types
- Specification and Implementation of Structured Data Types
- Type Equivalence, Checking and Conversion
- Vectors and Arrays
- Lists ,Structures
- Sets ,Files

## Elementary Data Types :Data Objects

✓**a run-time grouping** of one or more pieces of data in a **virtual computer**.

✓**a location in memory** with an assigned name in the **actual computer**.

- Programmer defined data objects
- System defined data objects

## Concepts

**Data value**: a bit pattern that is recognized by the computer.

**Elementary data object**: contains a data value that is manipulated as a unit.

**Data structure**: a combination of data objects.

**Attributes**: determine how the location may be used.

Most important attribute - the **data type**.

## Attributes and Bindings

**Type:** determines the set of data values that the object may take and the applicable operations.

**Name:** the binding of a name to a data object.

**Component:** the binding of a data object to one or more data objects.

These attributes are bound at translation

## Attributes and Bindings

**Location: the storage location in memory assigned by the system (bound at loading)**

**Value: the assignment of a bit pattern to a name (bound at execution)**

## Variables and Constants

**In programs, data objects are represented as variables and constants**

**Variables** : Data objects defined and named by the programmer explicitly.

## Variables and Constants

**Constants:**

Data objects with a name that is permanently bound to a value for its lifetime.

**Literals:** constants whose name is the written representation of their value.

**A programmer-defined constant**: the name is chosen by the programmer in a definition of data object.

**Persistence:** existence of data beyond run time

## Data Types

***A data type is a class of data objects with a set of operations for creating and manipulating them.***

**Examples** of elementary data types:

integer, real, character, Boolean, enumeration, pointer.

## Specification of a data type

- Attributes
- Values
- Operations

## Attributes

**Distinguish data objects of a given type** Data type and name - invariant during the lifetime of the object

**Approaches:**

- stored in a descriptor and used during the program execution
- used only to determine the storage representation, not used explicitly during execution

## Values

The data type determines the **values** that a data object of that type may have

**Specification:** Usually an ordered set, i.e. it has a least and a greatest value

## Operations

**Operations** define the possible manipulations of data objects of that type.

- **Primitive** - specified as part of the language definition
- **Programmer-defined** (as subprograms, or class methods)

## Operations

**An operation is defined by:**

- **Domain - set of possible input arguments**
- **Range - set of possible results**
- **Action - how the result is produced**

## Operation signature

**Specifies the domain and the range**

• the number, order and data types of the arguments in the domain,

• the number, order and data type of the resulting range mathematical notation for the specification:

op name: arg type x arg type x … x arg type $\rightarrow$result type

**The action is specified in the operation implementation.**

## Sources of ambiguity in operations

✓ Undefined operations for certain inputs.

✓ Implicit arguments, e.g. use of global variables

✓ Implicit results - the operation may modify its arguments

✓ Self-modification - usually through change of local data between calls, e.g. random number generators change the seed.

## Implementation of a data type

- **Storage representation**
- **Implementation of operations**

## Storage representation

**Influenced by the hardware Described in terms of:**

**Size of the memory blocks required**
**Layout of attributes and data values within the block**

## Methods to treat attributes

- ✓ determined by the compiler and not stored in descriptors during execution - C

- ✓ stored in a descriptor as part of the data object at run time - LISP Prolog

## Implementation of operations

- **Hardware operation direct implementation.** E.g. integer addition
- **Subprogram/function**, e.g. square root operation
- **In-line code** Instead of using a subprogram, the code is copied into the program at the point where the subprogram would have been invoked.

## Elementary Data Types

- **Scalar Data Types**
  - **Numerical Data Types**
  - **Other**

- **Composite Data Types**
  - **Character Strings**
  - **Pointers and Programmer-Constructed Objects**
  - **Files**

## Scalar Data Types

Scalar data types represent **a single object**, i.e. only one value can be derived.

In general, scalar objects follow the hardware architecture of a computer.

## Scalar Data Types

**Numeric Data Types**

- **Integers**
- **Sub -ranges**
- **Floating-point real numbers**
- **Fixed-point real numbers**

**Other Data Types -**

- **Complex numbers**
- **Rational numbers**
- **Enumerations**
- **Booleans**
- **Characters**

## Integers

**Specification**

Maximal and minimal values

**Operations:**

Arithmetic
Relational
Assignment
Bit operations

**Implementation** - hardware defined

## Sub-ranges

**Specification:** subtype of integer

a sequence of integer values within some restricted range

Example:

Pascal declaration **A: 1..10** means that the variable A may be assigned integer values from 1 through 10.

**Implementation**

smaller storage requirements, better type checking

## Floating-point real numbers

**Specification:** Minimum and maximal value

Round-off issues - the check for equality may fail due to round -off

**Implementation**

Mantissa - exponent model.

Example: $10.5 = 0.105 \times 10^2$,

Mantissa: 105, Exponent: 2

## Fixed-point real numbers

**Specification:**

real numbers with predefined decimal places

**Implementation :**

directly supported by hardware or simulated by software

# Other Scalar Data Types

**Complex numbers:** software simulated with two storage locations one the real portion and one for the imaginary portion.

**Rational numbers:** the quotient of two integers.

**Enumerations:** Ordered list of different values

**Booleans**

**Characters**

## Enumerations

**Example:**

```
enum StudentClass
        {Fresh, Soph, Junior, Senior}
```

the variable *StudentClass* may accept only one of the four listed values.

**Implementation:** represented during run time as integers, corresponding to the listed values.

## Booleans

**Specification:** Two values: true and false.

Can be given explicitly as enumeration

Basic operations: and, or, not.

**Implementation:** A single addressable unit such as byte or word.

Use a particular bit for the value, e.g. the last bit; 1 - true, 0 -false.

Use the entire storage; a zero value would then be false, otherwise - true.

## Characters

**Specification:** Single character as a value of a data object.

Collating sequence - the ordering of the characters, used for lexicographic sorting.

Operations:

Relational

Assignment

Testing the type of the character - e.g. digit, letter, special symbol.

**Implementation supported by the underlying hardware**

## Composite Data Types

**Characterized by a complex data structure organization, processed by the compiler.**

- **Character Strings**
- **Pointers and Programmer- Constructed Objects**
- **Files**

## Character Strings

**Specification:**

**Fixed declared length : storage allocation at translation time. Strings longer than the declared length are truncated.**

**Variable length to a declared bound: storage allocation at translation time. An upper bound for length is set and any string over that length is truncated**

**Unbounded length: storage allocation at run time. String can be any length**

## Character Strings - operations

- Concatenation – appending two strings
  - Relational operations – equal, less than, greater than
- Substring selection using positioning subscripts
- Substring selection using pattern matching
- Input / output formatting
- Dynamic strings - the string is evaluated at run time.

## Character Strings - implementation

- **Fixed declared length**: A packed vector of characters
- **Variable length to a declared bound**: a descriptor that contains the maximum length and the current length
- **Unbounded length**: Either a linked storage of fixed-length data objects or a contiguous array of characters with dynamic run-time storage allocation

## Pointers and Programmer-Constructed Objects

**Specification:**

Reference data objects only of **a single type** – C, Pascal, Ada.

Reference data objects of **any type** – Smalltalk

C, C++: pointers are data objects and can be manipulated by the program

Java: pointers are hidden data structures, managed by the language implementation

## Pointers - implementation

**Absolute addresses** stored in the pointer. Allows for storing the new object anywhere in the memory

**Relative addresses**: offset with respect to some base address.

Advantages: the entire block can be moved to another location without invalidating the addresses in the pointers, since they are relative, not absolute.

## Pointers – implementation problems

- **Management of a general heap storage area:** to create objects of different size
- **Garbage** - the contents of pointer is destroyed, and the object still exists
- **Dangling references**: the object is destroyed however the pointer still contains the address of the used location, and can be wrongly used by the program.

## Files

**Characteristics:**

- Usually reside on secondary storage devices as disks, tapes.
- Lifetime is greater than the lifetime of the program that has created the files.

**Implementation** – as part of the operating system

## Types of files

**Sequential file:** **a data structure composed of a linear sequence of components of the same type.**

**Interactive Input-Output:** **sequential files used in interactive mode.**

## Types of Files

**Direct Access Files: Any single component can be accessed at random just as in an array.**

**Key:** the subscript to access a component. **Implementation:** a key table is kept in main memory

**Indexed Sequential Files: Similar to direct access files using a key combined with being able to sequentially process the file. The file must be ordered by the key**

## Structured Data Types

**A data structure is a data object that contains other data objects as its elements or components.**

**Mechanisms to create new data types:**

- **Structured data**
  - **Homogeneous: arrays, lists, sets,**
  - **Non-homogeneous: records**
- **Subprograms**
- **Type declarations – to define new types and operations (Abstract data types)**
- **Inheritance**

## Data specifications

- **Number of components and size**
- **Type of each component**
- **Selection mechanism**
- **Maximum number** of components
- **Organization** of the components

## Data specifications

- **Number of components and size**

  Fixed size - Arrays

  Variable size – stacks, lists. Pointer is used to link components.

- **Type of each component**

  Homogeneous – all components are the same type

  Heterogeneous – components are of different types

## Data specifications - selection

**Selection mechanism** to identify components – index, pointer

**Two-step process:**

- referencing the structure
- selection of a particular component

## Data specifications - organization

- **Simple linear sequence** - arrays, stacks, lists
- **Multidimensional structures**:
  - separate types (Fortran)
  - a vector of vectors (C++)

## Operations on data structures

**Component selection** operations

Sequential

Random

**Insertion/deletion** of components

**Whole-data structure** operations

Creation/destruction of data structures

## Implementation of structured data types

- **Storage representations**
- **Implementation of operations on data structures**
- **Storage management**

## Storage representation

- storage for the components
- optional descriptor, contains some or all of the attributes

- **Sequential representation**
- **Linked representation**

## Sequential representation

✓The data structure is stored in a **single contiguous block of storage**, that includes both descriptor and components.

✓Used for **fixed-size structures**, homogeneous structures (arrays, character strings)

## Linked representation

✓The data structure is stored in **several noncontiguous blocks of storage**, linked together through pointers.

✓Used for **variable-size** structured (trees, lists)

✓Flexible, ensures true variable size, however it has to be **software simulated**

# Implementation of operations on data structures

Component selection in **sequential** representation

Base address plus offset calculation. Add component size to current location to move to next component.

Component selection in **linked** representation

Move from address location to address location following the chain of pointers.

## Storage management

**Access paths to a structured data object** - to endure access to the object for its processing. Created using a name or a pointer.

**Two central problems:**

**Garbage** – data object is bound but access path is destroyed. Memory cannot be unbound.

**Dangling references**: the data object is destroyed, but the access path still exists.

## Declarations and type checking for data structures

**What is to be checked:**

- **Existence of a selected component**
- **Type of a selected component**

## Type equivalence and equality of data objects

Two questions to be answered:

- When are two types the same?
- When do 2 objects have the same value?

## Name equivalence

**Two data types are considered equivalent only if they have the same name.**

**Issues**

Every object must have an assigned type, there can be no anonymous types.

A singe type definition must serve all or large parts of a program.

## Structural equivalence

**Two data types are considered equivalent if they define data objects that have the same internal components.**

**Issues**

- Do components need to be exact duplicates?
- Can field order be different in records?
- Can field sizes vary?

## Data object equality

**Two objects are equal if each member in one object is identical to the corresponding member of the other object.**

The compiler has no way to know how to compare data values of user-defined type. It is the task of the programmer that has defined that particular data type to define also the operations with the objects of that type.

## Type Checking versus Type Conversion

**Type checking: checking that each operation executed by a program receives the proper number of arguments of the proper data types.**

**Static type checking** is done at **compilation**.

**Dynamic type checking** is done at **run-time.**

**Strong typing:** all type errors can be statically checked

**Type inference:** implicit data types, used if the interpretation is unambiguous.

## Type Conversion and Coercion

**Coercion**: Implicit type conversion, performed by the system.

**Explicit conversion** : routines to change from one data type to another.

## Vectors and Arrays

**Vector** - one dimensional array

**Matrix** - two dimensional array

**Multidimensional arrays**

**Slice** - a substructure in an array that is also an array, e.g. a column in a matrix

**Associative Arrays** - elements are selected by a key value

## Implementation of array operations

**Access** - can be implemented efficiently if the length of the components of the array is known at compilation time.

The address of each selected element can be computed using an arithmetic expression.

**Whole array operations**, e.g. copying an array - may require much memory.

## Records

**A record is data structure composed of a fixed number of components of different types.**

The components may be heterogeneous, and they are named with symbolic names.

## Other structured data objects

**Records and arrays with structured components**

**Lists and sets**

**Executable data objects**

Data structures are considered to be a special type of program statements and all are treated in the same way (Prolog).