Data Types in Programming Languages

A Programming Language – Universal: All Solvable Computations

- integer values and arithmetic operators (arithmetic expressions)
- variables
- assignment statement
- selection statement
- loop statement/go to statement

Evolution of Data Types

- Built-in data types
- User-defined data types
- Abstract data types (ADT)

Data Types

- Primitive types
  - Indivisible values
- Structured types
  - Composed values

Kinds of Data Types in Programming Languages …

- Numeric types
  - Integer type, Floating –point type, Decimal type
- Boolean type
- Character type
- String type
- Ordinal types
  - Enumeration type, Subrange type

… Kinds of Data Types in Programming Languages

- Array type
- Associative array type
- Record type
- Union type
- Set type
- Pointer type

Primitive Data Types

- A data type that is not defined in terms of other data types.
- A programming languages provides a set of primitive built-in data types.
  - Numeric type
    - Integer numbers
    - Floating-point numbers
    - Decimal numbers
  - Boolean type
  - Character type

String Type

- Values are sequences of characters.
- Design issues:
  - Is it a primitive type or just a structured type (e.g. special kind of character array)?
  - Is the length of a string static or dynamic?

Ordinal Types (User Defined)

- A type in which the range of possible values can be easily associated with the set of positive integers.
  - Integer type, character type, boolean type
- User-defined ordinal types
  - Enumeration Type
  - Subrange Type

Enumeration Type

- Values are the values (symbolic constants) that are enumerated in the definition.
  - **`type DAYS is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);`**
- Aid to readability
  - No need to code a color as a number.
- Aid to reliability
  - Compiler can check operations and ranges of values.

Subrange Type

- Values are ordered contiguous subsequences of values of an ordinal type.
  - **`1..31`**
- All of the operations defined for the parent type are also defined for the subrange type.
  - Subtypes *inherit* operations from their parent types.

Subrange Type - Examples

- Pascal
  - Subrange types behave as their parent types;
  - **`type index = 1..100;`**
- Ada
  - Subtypes are not new types, just constrained existing types (so they are compatible);
  - **`subtype INDEX is INTEGER range 1..100;`**
  - **`type INDEX is new INTEGER range 1..100;`**

Ordinal Types - Implementation

- Enumeration types
  - As nonnegative integers.
- Subrange types
  - As the parent types with code inserted (by the compiler) to restrict assignments to subrange variables.

Type Constructors for Building Constructed (Structured) Types

- Operations to construct new types out of existing types.
- In a programming language, all types are constructed out of primitive/non-primitive types using **type constructors**.

Constructed (Structured) Types

- Array type
- Associative array type
- Record type
- Union type
- Set type
- Pointer/Reference type

Array Type

- Values are aggregates of ***homogeneous*** data elements s.t.
  - An individual element is identified by its *position* in the aggregate relative to the first element.
  - Introduced by FORTRAN.

Array Type – Design Issues

1. What types are legal for subscripts?
2. Are subscripting expressions in element references range checked?
3. When are subscript ranges bound?
4. When does allocation take place?
5. What is the maximum number of subscripts?
6. Can array objects be initialized?
7. Are any kind of slices allowed?

Array Type – Implementation

- The code to access array elements must be generated at compile-time.
- The code is executed to produce array element addresses.

Array Type – Access Function

- Access function maps a subscript expression to an address in the array.
- One dimensional arrays: A[k] with lower=1

address(A[k]) =

address(A[1]) +

(k - 1) * element_size

address(A[k]) =

(address(A[1]) - element_size) +

k * element_size

Associative Array Type

- Values are unordered collections of data elements
  - Indexed by an equal number of values called *keys.*
  - (A key + A value)
  - Set mapping (→)
  - Hash
- Perl:
  - Hash variable names begin with %

```
%hi_temps = ("Monday" => 77, "Tuesday" => 79,…);
```

  - Subscripting is done using braces and keys

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

Record Type

- Values are possibly **heterogeneous** aggregates of data elements
  - The individual elements are identified by names.
  - Introduced by COBOL.
  - Set Cartesian product (×)
  - To model collections of heterogeneous data elements.
- Design Issues:
  - What is the form of references?
  - What unit operations are defined?

Record Type – Record Field Reference

- Fully qualified references
  - must include all record names.
- Elliptical references
  - allow leaving out record names as long as the reference is unambiguous.

Record Type – Operations & Implementation

- Operations:
  - Assignment
    - Pascal, Ada, and C allow it if the types are identical.
    - In Ada, the RHS can be an aggregate constant.
  - Initialization
  - Comparison
- Implementation:
  - See Fig. 6.8 (p. 268)

Comparing Records and Arrays

- Component type:
  - Array – Homogeneous
  - Record – Heterogeneous
- Component selector:
  - Array – Expressions evaluated at run-time
  - Record – Names known at compile-time
  - Access to array elements is much slower than access to record fields.

Union Type

- A type whose variables are allowed to store different type values at different times during execution.
  - Set union (∪)
- Values are the set union of different type values. Two union types:
  - Undiscriminated unions (Free unions)
  - Discriminated unions
    - A tag or discriminator is added to each element field to distinguish the type.

Union Type – Evaluation

- Potentially unsafe
  - FORTRAN, Pascal, C, C++ and Modula-2 (not Ada)
  - Java, Modula-3 – No union.
- Flexibility

Set Type

- Values are unordered collections of distinct values from some ordinal type.
  - Powerset
  - Introduced by Pascal.
- Design Issue:
  - What is the maximum number of elements in any set base type?

Pointer Type

- Values are memory addresses and a special value nil (or null).
  - First introduced in PL/I.
- Pointing to
  - Heap memory cells
  - Non-heap memory cells
- Type operators
  - `*, access, ^`

Pointer Type – Design Issues

- What is the scope and lifetime of a pointer variable?
- What is the lifetime of a heap-dynamic variable?
- Are pointers restricted to pointing at a particular type?
- Are pointers used for dynamic storage management, indirect addressing, or both?
- Should a language support pointer types, reference types, or both?

Pointer Type - Operations

- Allocation
  - `new, new, malloc`
- Deallocation
  - `dispose, delete, free`
- Assignment of an address to a pointer
- Dereferencing (explicit versus implicit)
  - `my_ptr^, (*my_ptr)`

Pointer Type - Problems

1. **Dangling pointers (references)**
2. **Lost heap-dynamic variables (garbage)**

- **Why?**
  - Explicit heap storage deallocation (reclamation)

1. Dangling Pointers

- A pointer to storage that has been reclaimed (deallocated) and perhaps reallocated for another purpose.
- How?
  - Allocate a heap-dynamic variable and set a pointer to point at it.
  - Set a second pointer to the value of the first pointer.
  - Deallocate the heap-dynamic variable using the first pointer.
- A dangling pointer is undesirable (dangerous).

2. Lost Heap-Dynamic Variables

- A heap-dynamic variable that is no longer referenced by any program pointer.
- How?
  - Pointer p1 is set to point to a newly created heap-dynamic variable.
  - p1 is later set to point to another newly created heap-dynamic variable.
- It is undesirable (wasteful).

Memory Leak

- A situation in which memory continues to be used even though it is no longer needed by the program.
  - When a program fails to reclaim the heap memory cells that are allocated but no longer referenced and thus needed.
  - Run out of memory & crash!

Pointer Types

- Pascal:
  - Allocation - new
  - Explicit deallocation - dispose
  - Dangling pointers are possible
- C and C++:
  - Explicit dereferencing (*) and address-of operator (&)
  - Pointer arithmetic is possible.
  - **`new & delete`**
  - Dangling pointers are possible

Reference Types

- A special kind of pointer type.
- C++ Reference Types
  - Constant pointers that are implicitly dereferenced.
  - Used for parameters.
  - Advantages of both pass-by-reference and pass-by-value.

Reference Types

- Java Reference Types
  - Only references.
  - No pointer arithmetic.
  - Can only point at objects (which are all on the heap).
  - No explicit deallocator (garbage collection is used) - no dangling references.
  - Dereferencing is always implicit.

Automatic Reclamation of Garbage

- Implicit & automatic deallocation/reclamation
- **Garbage Collection**:
  - Eager approach
    - Reference counting
  - Lazy approach
    - Mark-and-Sweep Garbage collection

Abstract Data Type (ADT)

Data Abstraction

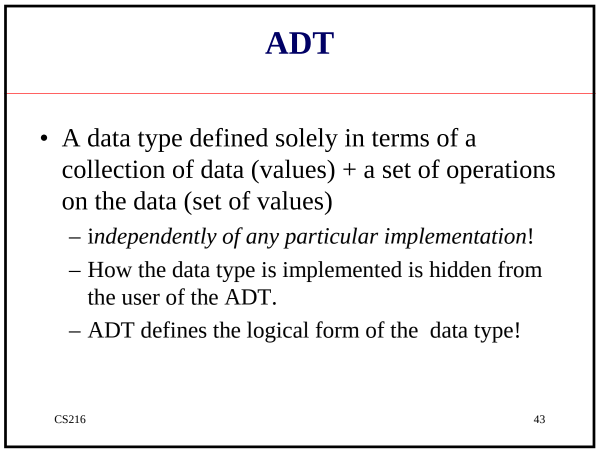
- The separation of a data type's logical properties from its implementation.
- The separation of the representation of data from the applications that use the data at a logical level.
- Data abstraction: Applying abstraction to data types!

Data Encapsulation

- The physical representation of the data is surrounded.
- The user of the data does not see the implementation.
- The user deals with the data ONLY in terms of its logical picture - its abstraction.
- Data encapsulation: Applying information hiding to data types!

Abstract Data Type

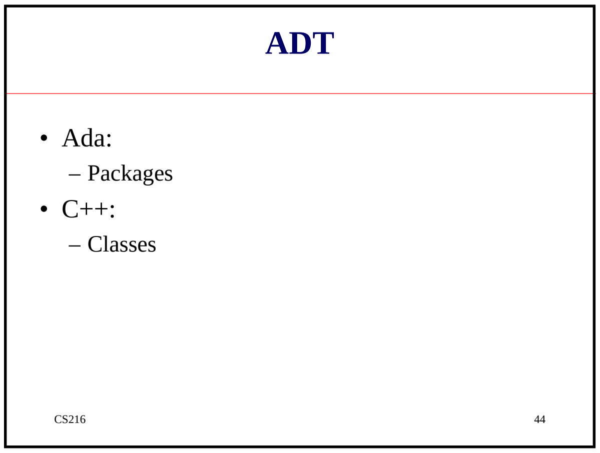
- Applying Abstraction & Information Hiding to Data Types!
- Built-in data types are ADT!



ADT

- A data type defined solely in terms of a collection of data (values) + a set of operations on the data (set of values)
  - *independently of any particular implementation*!
  - How the data type is implemented is hidden from the user of the ADT.
  - ADT defines the logical form of the data type!

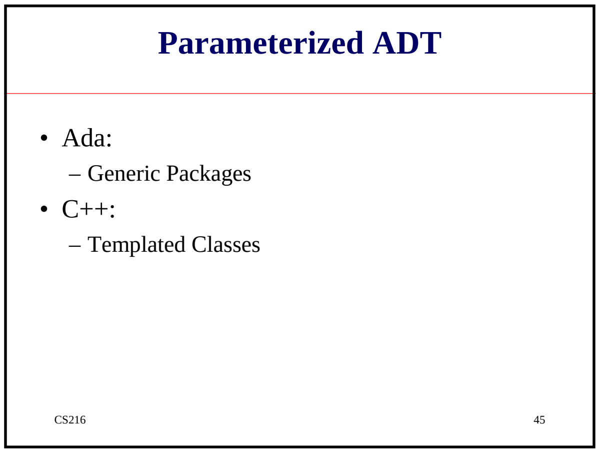
CS216 43



ADT

- Ada:
  - Packages
- C++:
  - Classes

CS216 44



Parameterized ADT

- Ada:
  - Generic Packages
- C++:
  - Templated Classes

CS216 45