Chapter 6

Data Types

Topics

- Introduction
- Type Information
- Data Type
  - Specification of a Data Type
- Primitive Data Types
  - Numeric Data Types
    - Integers
    - Floating-Point Numbers
    - Fixed-Point Real Numbers
  - Boolean Types
  - Character Types

Introduction

- Every program uses data, either explicitly or implicitly, to arrive at a result.
- All programs specify a set of operations that are to be applied to certain data in a certain sequence.
- Data in its most primitive forms inside a computer is just a collection of bits.

Introduction

- Basic differences among languages exist in the types of data allowed, in the types of operations available, and in the mechanism provided for controlling the sequence in which the operations are applied to the data.
- Most programming languages provide a set of simple data entities as well as mechanism for constructing new data entities from these.

Type Information

- Program data can be classified according to their types.
- Type information can be contained in a program either implicitly or explicitly.

Type Information: implicit

- Implicit type information includes the types of constants and values, types that can be inferred from a name convention, and types can be inferred from context.
  - Example: number 2 is implicitly an integer in most languages
  - Example: TRUE is Boolean
  - Example: variable I in FORTRAN is, in the absence of other information, an integer variable.

Type Information: explicit

- Explicit type information is primarily contained in declarations.
  - Variables can be declared to be of specific types.
  - Example: `var x: array[1..10] of integer;`
  - Example: `var b: boolean;`

Data Type

- A data type is a set of values that can be specified in many ways:
  - Explicitly listed
  - Enumerated
  - Given as a subrange of known values
  - Borrowed from mathematics
- A set of values also has a group of operations that can be applied to the values.
  - These operations are often not mentioned explicitly with the type, but are part of its definition.

Data Type: definition

- A ***data type*** is a set of values, together with a set of operations on those values having certain properties.
- Every language comes with a set of **predefined types** from which all other types are constructed.
  - Provide facility to allow the programmer defined new data types.

Specification of a Data Type

- Basic elements of a specification of a data type:
  1. The *attributes* that distinguish data objects and types.
  2. The *values* that data objects of that type may have, and
  3. The *operations* that defined the possible manipulations of data objects of that type.

Specification of a Data Type: example

- Specification of an array:
  1. The *attributes* might include the number of dimensions, the subscript range for each dimension, and the data type of the components.
  2. The *values* would be the sets of numbers that form valid values for array components.
  3. The *operations* would include subscripting to select individual array components and possibly other operations to create arrays, change their shape, access attributes such as upper and lower bounds of subscripts, and perform arithmetic on pair of arrays.

Primitive Data Types

- Algol-like languages (Pascal, Algol68, C, Modula-2, Ada, C++), all classify types according to a basic scheme, with minor variations.
  - Names used are often different, even though the concepts are the same.
- Primitive types are also called *base types* or *scalar types* or *unstructured types* or *elementary types*.
- A scalar type is a type whose elements consist of indivisible entities (a single data value or attribute).

Numeric Data Types

- Some form of numeric data is found in almost every programming language.
  - Integers
  - Floating-Point Real Numbers
  - Fixed-Point Real Numbers

Integers: specification

- The set of integer values defined for the type forms an ordered subset, within some finite bounds, of the infinite set of integers studied in mathematics.
  - The maximum integer value is sometimes represented as a defined constant
    - Example: In Pascal, the constant *maxint*
    - The range of values is defined to be from *–maxint* to *maxint*
  - Some languages, such as C, have different integer specifications: *short, long*

Integers: specification

- Operations on integer data objects include the main groups:
  - Arithmetic Operations:
    - Binary arithmetic operations such as *addition* (+), *subtraction* (-), *multiplication* (*), *division* (*/* or div), *remainder* (mod).
      BinOp: integer x integer → integer
    - Unary arithmetic operations such as *negation* (-) or *identity* (+).
      UnaryOp: integer → integer

Integers: specification

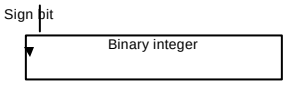
- Relational Operations:
  - Includes *equal*, *not equal*, *less-than*, *greater-than*, *less-than-or-equal*, and *greater-than-or-equal*.
    RelOp: integer x integer → Boolean
- Assignment
  assignment: integer x integer → integer
- Bit Operations
  - In some languages, integers fulfill many roles.
  - In C, integer also play the role of Boolean values.
  - C includes operations to *and* the bits together (&), *or* the bits together (|), and *shift* the bits (<<).

Integers: implementation

- Implemented using hardware-defined integer storage representation and set of hardware possible storage representations for integers.
- Some possible storage representation for integers.

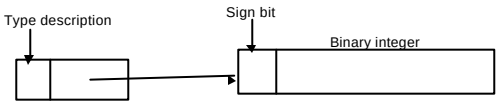
Integers: implementation

- No descriptor
  - Only the value is stored.
  - Possible if the language provides declarations and static type checking for integer data objects.



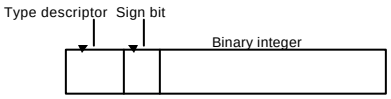
Integers: implementation

- Descriptor separated
  - Stores the description in a separate memory location, with a pointer to the full-word integer value.
  - Often used in LISP.
  - Disadvantage: It may double the storage required.
  - Advantage: because the value is stored using the build-in hardware representation, the hardware arithmetic operations may be used.



Integers: implementation

- Descriptor in the same word
  - Descriptor and value are stored in a single memory location by shortening the size of the integer sufficiently to provide space for the descriptor.
    - Storage is conserved.
    - Hardware operations: clearing the descriptor from the integer data object, perform the arithmetic, and then reinserting the descriptor.
    - Arithmetic is inefficient.



Floating-Point Numbers: specification

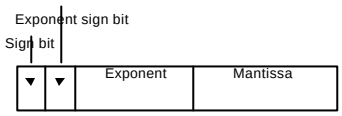
- Sometimes called *real*, as in FORTRAN.
- Also called *float*, as in C.
- As with type integer, the values form an ordered sequence from some hardware-determined minimum negative value to a maximum value, but the values are not distributed evenly across this range.

Floating-Point Numbers: specification

- Same arithmetic, relational, and assignment operations as for integers.
  - Boolean are sometimes restricted.
  - Due to roundoff issues: equality between two real numbers is rarely achieved.
    - Equality may be prohibited by the language designer.
  - Most languages provide other operations as build-in functions:
    - *Sine*: Sin: real → real
    - *Maximum value*: max: real x real → real

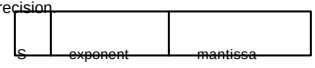
Floating-Point Numbers: implementation

- Storage representation based on an underlying hardware representation.
  - Mantissa (i.e. the significant digits of the number)
  - Exponent



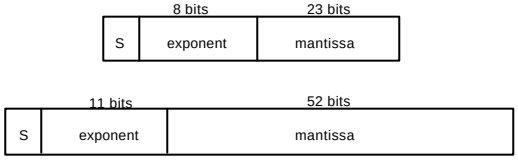
Floating-Point Numbers: implementation

- This model emulates scientific notation
  - Number N can be expressed as $N=mx2^k$
  - IEEE Standard 754
    - 32 and 64 bit standard
  - Numbers consist of 3 fields:
    - S : a one-bit sign field. 0 is positive
    - E: Values (8 bits) range from 0 to 355 corresponding to exponents of 2 that ranges from –127 to 128
    - M: a mantissa of 23 bits. The first number is always 1 then it is inserted automatically by the hardware yielding an extra 24th bit of precision



Floating-Point Numbers: implementation

- A double-precision form is available in many programming languages.
  - Additional memory word is used to store an extended mantissa.



Fixed-Point Real Numbers: specification

- A fixed-point number is represented as a digit sequence of fixed length, with the decimal point position at a given point between two digits.
  - Example: In COBOL:

    X PICTURE 999V99

    declares X as a fixed-point variable with 3 digits before the decimal and 2 digits after.

Fixed-Point Real Numbers: implementation

- A fixed-point type may be directly supported by the hardware or may be simulated by software.
  - Example: In PL/I, fixed data are of type Fixed Decimal.

    DECLARE X FIXED DECIMAL (10,3),
    Y FIXED DECIMAL (10,2),
    Z FIXED DECIMAL (10,2);

    - Data is stored as *integers*, with the decimal point being an *attribute* of the data object.

Fixed-Point Real Numbers: implementation

- If X has the value 103.421, then r-value of X will be 103 421, and the object X will have an attribute *scale factor (SF)* of three (the decimal point is 3 places to the left).
- The statement Z = X + Y
  - Shift Y left one position (equivalent to multiplying the integral r-value of Y by 10)
  - The sum will have 3 decimal digits (SF=3)
  - Because Z has only 2 decimal places (SF=2) and the sum has 3, remove one place (divide by 10).
- Subtraction and division are handled in an analogous manner.

Boolean Types: specification

- Data objects having one of two values (*true* of *false*).
- The most common operations on Boolean types include assignment and the following logical operations:
  - *and:* Boolean x Boolean → Boolean (conjunction)
  - *or*: Boolean x Boolean → Boolean (inclusive disjunction)
  - *not*: Boolean → Boolean (negation or complement)
- Other Boolean operations: equivalence, exclusive or, implication, nand (not-and), and nor (not-or).

Boolean Types: implementation

- Storage is a single bit providing no descriptor
- Because single bits may not be separately addressable in memory, its storage is extended to be a single addressable unit such as a byte.
- The values *true* and *false* might be represented in 2 ways within this storage unit:
  - A particular bit is used for the value (often, the sign bit of the number representation), with 0=false and 1=true, and the rest of the byte is ignored.
  - A zero value in the entire storage unit represents false, and any other nonzero value represents true.

Character Types: specification

- The set of possible character values is usually taken to be a language-defined enumeration corresponding to the standard character sets supported (I.e. ASCII).
- Operations on character data include only the relational operations, assignments, and sometimes operations to test whether a character is one of the special classes letter, digit, or special character.