Principles of Programming Languages

Unit 2

Third Semester
MScCSIT, TU

2.1

- Introduction to data types
- Specification and Implementation of primitive and other data types
- Declaration
- Initialization and assignment
- Type checking and conversion.

Basic differences among programming languages:

- types of data allowed
- types of operations available
- mechanisms for controlling the sequence of operations

- **Elementary data types:** built upon the available hardware features
- **Structured data types:** software simulated

Data objects, variables, and constants

Data object:

- **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

- **Types of data objects**:
  - Programmer defined data objects - variables, arrays, constants, files, etc.
  - System defined data objects - set up for housekeeping during program execution, not directly accessible by the program. E.g. run-time storage stacks.

- **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.
- **Location:** the storage location in memory assigned by the system.
- **Value:** the assignment of a bit pattern to a name.
- Type, name and component are bound at translation, location is bound at loading, value is bound at execution

Data objects in programs

- In programs, data objects are represented as variables and constants
- **Variables:** Data objects defined and named by the programmer explicitly.
- **Constants:** a data object 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 the data object.

Persistence

- Data objects are created and exist during the execution of the program. Some data objects exist only while the program is running. They are called **transient data objects**.

- Other data objects continue to exist after the program terminates, e.g. data files. They are called **persistent data objects.**

- In certain applications, they need a mechanism to indicate that an object is persistent. Languages that provide such mechanisms are called **persistent languages**.

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 elementary data types

Attributes that distinguish data objects of that type

- Data type, name - invariant during the lifetime of the object
  - stored in a descriptor and used during the program execution
  - used only to determine the storage representation, not used explicitly during execution

- **Values** that data object of that type may have
  - Determined by the type of the object
    Usually an ordered set, i.e. it has a least and a greatest value

- **Operations** that 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)

- An operation is defined by:
    - Domain - set of possible input arguments
    - Range - set of possible results
    - Action - how the result is produced

- The domain and the range are specified by the **operation signature**
    - 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 → result type
- The action is specified in the operation implementation

- **Sources of ambiguity** in the definition of programming language operations
    - Operations that are undefined for certain inputs.
    - Implicit arguments, e.g. use of global variables
    - Implicit results - the operation may modify its arguments

- **Subtypes :** a data type that is part of a larger class.
  Examples: in C, C++ int, short, long and char are variations of integers.

  The operations available to the larger class are available to the subtype.
  This can be implemented using inheritance.

Implementation of a data type

- **Storage representation :** Influenced by the hardware,
  Described in terms of:
  - Size of the memory blocks required
  - Layout of attributes and data values

- Two methods to treat attributes:
  - determined by the compiler and not stored in descriptors during execution
  - stored in a descriptor as part of the data object at run

- **Implementation of operations**
  - Directly as a hardware operation. 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.

Declarations

- **Declarations provide information about the name and type of data objects needed during program execution.**
  - Explicit – programmer defined
  - Implicit – system defined

  e.g.

  FORTRAN - the first letter in the name of the variable determines the type
  Perl - the variable is declared by assigning a value

  $abc = 'a string' $abc is a string variable
  $abc = 7 $abc is an integer variable

- **Operation declarations:** prototypes of the functions or subroutines that are programmer-defined.

  Examples:

  declaration: **float Sub(int, float)**
  signature: **Sub: int x float --> float**

- **Purpose of declaration**
  - Choice of storage representation
  - Storage management
  - Declaration determines the lifetime of a variable, and allowes for more efficient memory usage.
  - Specifying polymorphic operations.

- Depending on the data types operations having same name may have different meaning, e.g. integer addition and float addition

- In most language +, -. *, / are overloaded

- Declarations provide for static type checking

Assignment and Initialization

Assignment - the basic operation for changing the binding of a value to a data object.

- Two different ways to define the assignment operation:
  - does not return a value
  - returns the assigned value
- The assignment operation can be defined using the concepts **L-value** and **R-value**
- Location for an object is its **L-value.**
  Contents of that location is its **R-value.**

Consider executing: A = A + B;

1. Pick up contents of location A: **R-value of A**
2. Add contents of location B: **R-value of B**
3. Store result into address A: **L-value of A**.

For each named object, its position on the right-hand-side of the assignment operator (=) is a *content-of* access, and its position on the left-hand-side of the assignment operator is an *address-of* access.

- **address-of is an L-value**
- **contents-of is an R-valueValue,**
- **by itself, generally means R-value**

Initialization

Uninitialized data object - a data object has been created, but no value is assigned, i.e. only allocation of a block storage has been performed.

- An explicit assignment of a valid value to a named data object is termed as initialization.
  - automatically
  - Explicitly (optional)

Type Checking and 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.

Dynamic type checking – Perl and Prolog
Implemented by storing a type tag in each data object

Advantages: Flexibility
Disadvantages:

- Difficult to debug
- Type information must be kept during execution
- Software implementation required as most hardware does not provide support

- Concern for static type checking affects language aspects:
  Declarations, data-control structures, provisions for separate compilation of subprograms

Strong typing: all type errors can be statically checked
Type inference: implicit data types, used if the interpretation is unambiguous. Used in ML

- **Explicit type conversion**
  - It is a routines to change from one data type to another.
  - Pascal: the function round - converts a real type into integer
    C - cast, e.g. (int)X for float X converts the value of X to type integer
- **Coercion:** implicit type conversion, performed by the system.
  - Pascal: + integer and real, integer is converted to real
    Java - permits implicit coercions
    C++ - and explicit cast must be given.
- Two opposite approaches to type coercions:
  - No coercions, any type mismatch is considered an error : Pascal, Ada
  - Coercions are the rule. Only if no conversion is possible, error is reported.
- **Advantages** of coercions: free the programmer from some low level concerns, as adding real numbers and integers.
- **Disadvantages**: may hide serious programming errors.

Scalar Datatypes

- Scalar data types represent a single object, i.e. only one value can be derived. Their objects follow the hardware architecture of a computer.
  - **Numeric data types**
    - **Integers**

      Specification

      - Maximal and minimal values - depending on the hardware.
        In some languages these values represented as defined constants.

      Operations:

      - Arithmetic
        Relational
        Assignment
        Bit operations

      Implementation : They use the hardware-defined integer storage representation and a set of hardware arithmetic and relational primitive operations on integers.

      - **Subranges**
      - **Specification:** A subtype of integer, consists of a sequence of integer values within some restricted range. e.g. a 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

- Ordered sequence of some hardware-determined minimum negative value to a maximum value.
- Similar arithmetic, relational and assignment operations as with integers. Roundoff issues - the check for equality may fail due to roundoff.

Implementation: Mantissa - exponent model.
The storage is divided into a mantissa - the significant bits of the number, and an exponent.

- Example: $25.5 = 0.255 \times 10^2$,

- Mantissa: 255
  Exponent: 2

- Fixed-point real numbers

Specification: Used to represent real numbers with predefined decimal places such as Rupees and paisa.

Implementation: Directly supported by hardware or simulated by software.

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

- **Rational numbers:** the quotient of two integers.
- **Enumerations:** ordered list of different values.

- **Booleans**
  - **Specification**: Two values: true and false. Can be given explicitly as enumeration, as in Pascal and Ada. Basic operations: and, or, not.
  - **Implementation**: A single addressable unit such as byte or word. Two approaches:

    - 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:** usually directly supported by the underlying hardware.

- What is a scalar data type? Give examples.
- Describe briefly the specification various scalar datatypes.
- Describe briefly the implementation various scalar datatypes.

Composite Data Types

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

1. Character strings: Data objects that are composed of a sequence of characters

– **Specification and syntax**

- Three basic methods of treatment:
- **Fixed declared length** - storage allocation at translation time
  The data object is always a character string of a declared length.
  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. Strings can be of any length.
  Special case: C/C++
  - Strings are arrays of characters
  - No string type declaration
  - Null character determines the end of a string.

- Operations

- Concatenation : appending two strings one after another
- Relational operation on strings : 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.

- 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 tun-time storage allocation.

- What is a composite data type? Give examples.

- Describe briefly the approaches to specification, operation and implementation of Pointer datatype and File datatype

- What implementation problems exist with data objects referred to by pointers?