Overview

What is a good program?

- Run _____
  - i.e., run in accordance with the specifications
- Run _____ (time vs. space)
  - Better running times will generally be obtained from use of the most appropriate data structures and algorithms, rather than through "hacking" (, i.e. removing a few statements by some clever coding - or even worse, programming in assembler!)
- Easy to read and understand
- Easy to _____
- Easy to _____

Definition of an Algorithm

- ***Definition:*** _____

  _____,

  In addition, all algorithms must satisfy the following criteria:

1) **Input**
   - There are zero or more quantities that are externally supplied.
2) **Output**
   - At least one quantity is produced.

Definition of an Algorithm (cont'd)

3) **Definiteness (확실성)**
   - Each instruction is clear and unambiguous.
4) **Finiteness (한정성)**
   - If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.
5) **Effectiveness (효율성)**
   - Every instruction must be basic enough to be carried out, in principle, by a person using pencil and paper. (It is not enough that each operation be definite as in 3); it also must be feasible.)

Program vs. Algorithm

- A program does not have to satisfy the _____ condition: e.g. operating systems, embedded systems, etc.
- However, since our programs will always terminate, we will use algorithm and program interchangeably in this course.

How to specify an Algorithm?

- Use a natural language like English or Korean.
  - Hard to make sure the statements are definite.
- Use graphic representation such as flowcharts.
  - Only suitable for small and simple algorithms.
- Use a ***pseudo-language*** or a _____ ; a combination of the constructs of a programming language (e.g., Java language) together with informal natural statements.
  - We will describe an algorithm in Java, or sometimes in a pseudo-language.

Pseudocode

- High-level description of an algorithm
- More structured than English prose
- Less detailed than a program
- Preferred notation for describing algorithms
- Hides program design issues

Example: find max element of an array

Algorithm *arrayMax*(*A*, *n*)
  Input array *A* of *n* integers
  Output maximum element of *A*

  currentMax ← *A*[0]
  for *i* ← 1 **to** *n* − 1 **do**
    if *A*[*i*] > *currentMax* **then**
      currentMax ← *A*[*i*]
  return *currentMax*

Pseudocode Details

- Control flow
  - **if** … **then** … [**else** …]
  - **while** … **do** …
  - **repeat** … **until** …
  - **for** … **do** …
  - Indentation replaces braces
- Method declaration

  Algorithm *method* (*arg* [, *arg*…])
    Input …
    Output …
- Method call

  var.method (*arg* [, *arg*…])
- Return value

  return *expression*
- Expressions
  - ← Assignment (like = in Java)
  - = Equality testing (like == in Java)
  - n^2 Superscripts and other mathematical formatting allowed

What is a data type?

- ***Definition:*** *A data type is a category of data characterized by _____ and _____ that act on those values.*
- A data type is a notion used in programming language.

Example: **int** data type in Java

- value: {-2^31, ..., -1, 0, 1, 2, 2^31-1}
- operations: +, -, / (division), * (multiplication), <<, >> etc.

Example: **boolean**

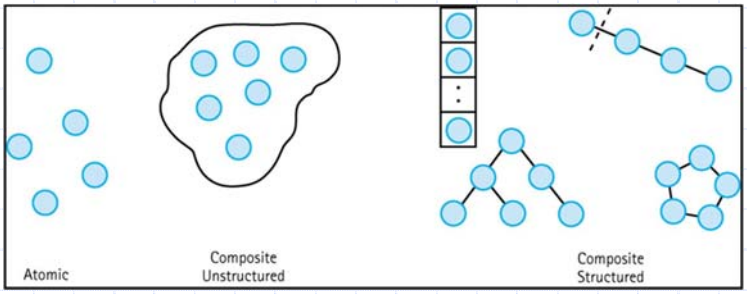
- value: {true, false}
- operations: and, or, not etc.

Data Types in Java

- **Primitive (simple or atomic) types**
  - Predefined data types
  - boolean, byte, char, short, int, long, float, double
- **Reference types**
  - Interfaces and classes
    - User-defined types
  - Arrays

Collection (or Composite) Data Types

- Elements are composed of multiple data items.
  - **Structured**: A collection of components that are organized with respect to one another
  - **Unstructured**: A collection of components that are not organized with respect to one another



Data Structures

- Logical organization of a collection of data elements reflecting their relationships
  - Set, One-to-One, One-to-Many, Many-to-Many etc.
- Implementation dependent data structures
  - _____ and _____
- Implementation independent data structures
  - Stack, Queue, Tree, Hash Table, Heap, Graph, etc.
- Implementation independent data structures are normally defined first as _____.

Abstract Data Type (ADT)

- ***Definition:*** *An ADT is a data type whose properties (_____) are specified independently of any particular _____.*
  - *Separation of _____ from implementation*
- Java interface and class mechanism provides the means to define ADTs.

What is an ADT for?

- **Generalization**
  - ADTs can be used just like primitive types in PL.
- **Encapsulation** (**information hiding**):
  - The definition of an ADT and its operations can be localized to one section of the program.
  - Functions that make use of the ADT can safely ignores its implementation details.
- **Note**: An ADT can be built on some ADTs which can also be build on other ADTs as well.

Specification of Collection ADT

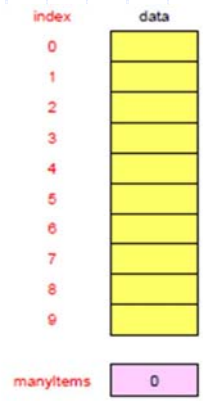
- A Collection is a data type that is capable of holding a group of other items.
- Bag ADT
  - There can be many instances of the same item in the bag.

Operation	Action
initialize():	Creates an empty collection of fixed capacity = 10.
add(item):	Adds one item to the collection.
countOccur(item):	Checks how many occurrences of a certain item are in the collection.
remove(item):	Removes one item from the collection.
size():	checks how many items are in the collection.

Implementation Using Java Array

Representation:

- Use a partially filled array of fixed capacity
- Use one integer variable called manyItems, which stores the number of items currently in the bag
- An empty bag is initialized by a constructor, dynamically creating the array, and setting manyItems = 0.



Code:

```
public class IntArrayBag
{
  private int[ ] data;
  private int manyItems;
```

```
public IntArrayBag( )
{
  final int INITIAL_CAPACITY = 10;
  manyItems = 0;
  data = new int [INITIAL_CAPACITY];
}
```

Remove

Code:

```
/**
* Remove one copy of a specified element from this bag.
* @param target
*   the element to remove from the bag.
* @postcondition
*   If target is in the bag, one copy is removed, returns true.
*   Otherwise the bag remains unchanged, returns false.
**/
public boolean remove(int target)  {
  int i;                                  // Find target
  for (i = 0; (i < manyItems) && (target != data[i]); i++) ;
  if (i == manyItems) return false;  // Not found.
  data[i] = data[--manyItems];       // Found. So remove.
  return true;
}
```