Ch 2. A Tour of C++

- 2.1 Built-in array type
- 2.2 Dynamic memory allocation
  - Memory management and memory leak
- 2.3 An object-based design
  - Information hiding: Public interface and private implementation.
  - Member functions and data members
  - Function overloading
- 2.4 An object-oriented design
  - Inheritance and polymorphism, virtual functions
2.5 A generic design
   - Template functions and classes

Programming in the large
   - 2.6 An exception-based design
   - 2.7 Name space

2.8 Standard template library
C++ built-in array

- An array is a sequential collection of elements of one type.

- Limitations
  - Cannot assign two arrays
    ```
    int array0[10], array1[10];
    ...
    // error: cannot directly assign one array to another
    array0 = array1;
    ```
  - Array has no knowledge of its own size
    e.g., void sort (int a[], int size);
    - separation of data and the algorithms;
    - procedural paradigm
- Size fixed at compile-time
  - Dynamic allocation can solve the problem at the following costs:
    - Less efficient memory allocation
    - Memory management: heap space must be freed after use, or else *memory leak.*
    - Unnamed variable vs named variable of built-in array; manipulated through pointers.
int numDays = 365;
int * p = &numDays;
int a[4] = {2,3,5,7};

What is the type of a?

1. int
2. int *
3. const int *
4. int * const

A constant pointer to integers

const double pi = 3.14159;
const double *const pi_ptr = &pi;
An object-based array

Objectives

- Self knowledge of size
- Array copy and comparison
- Size at creation time, with initialization
- Subscript operator
- Provision of services: \textit{min, max, find, sort}
- Bound checking
C++ features to support the object-based paradigm

- Class
  - Member functions and data members
  - Function and operator overloading
  - Constructor and destructor
  - Call by reference

- Modeling principle:
  - public interface and private implementation
  - Objects that *do* things
  - Objects use services of other objects by sending *messages* (calling member functions)
C++ features to support the object-oriented paradigm

- Class and subclass
  - Inheritance (derivation, subclassing)
  - Virtual functions
- Modeling principle:
  - Polymorphism: the ability to treat different type of classes identically.
  - Separation of interface and implementation: *program to an interface, not an implementation*
Class diagram

IntArray eq. Array<int>
To have polymorphism in your C++ program, you need

- A base class in which a member function intended to be polymorphic is declared virtual;
- A derived class that override the definition of the virtual member function;
- Client code that handles these objects by pointer or reference to the base class. E.g.,
  
  ```cpp
  void swap(IntArray & ia, int i, int j); or
  void swap(IntArray * ia, int i, int j);
  ```
C++ features to support the generic programming

- Template function
- Template class
- Component based solution: standard template library
  - Containers
  - Algorithms that act on containers
Exceptions
Namespace