Arrays and Structures

Hsuan-Tien Lin

Dept. of CSIE, NTU

March 08–09, 2011

What is an Array? (Sec. 2.1)

很多放桌面的空杯

有编號的置物櫃

```
1  int arr[30];
```

有編號的置物櫃

Arrays: Why?

下有条统的整理

- 方便記錄
- 同類放一起
- 如拿（編號）

  如收據

Arrays: Consecutive Memory Implementation

```
1  int a[30];
2  int* b = (int*)malloc(sizeof(int) * 30);
3
4  a[3] = 1; /* *(a+3) = 1; */
5  b[2] = 5; /* *(b+2) = 5; */
6
7  printf("%d", a[1]);
8  /* printf("%d", *(a+1)); */
```

- simple computation of locations
  - constant time retrieving
  - constant time storing

Arrays: from Implementation to Abstraction

C Implementation View

(One-dimensional) array is a block of consecutive memory that

- holds a list of N elements
- allows users to retrieve the k-th element
- allows users to store to the k-th location

An Abstract View

Abstract (one-dimensional) array

- holds a list of N elements
- allows users to retrieve the k-th element
- allows users to store to the k-th location

different implementations:
different space/time complexity

Dense Array versus Sparse Array

one abstract array, two possible implementations

```
1 int dense[10] = {1, 3, 0, 0, 0, 0, 0, 0, 0, 2};
2 int sparse[3][2] = {{0, 1}, {1, 3}, {9, 2}};
```

- dense array: store everything (consecutively), needs 10 positions
  - space: $O(N)$ for a length-N array
  - retrieving: $O(1)$
  - storing: $O(1)$
  - creating: $O(1)$
- sparse array: store only non-zero (index, element) pairs, needs 3 pairs
  - space: $O(E)$ for E elements, better than $O(N)$ if E small
  - retrieving: $O(\log E)$ if index ordered (HOW?)
  - storing: ???
  - creating: ???

note: often use **array** to mean dense array only

Concrete Data Type (Sec. 1.4)

array consists of ...

- objects: a set of (*index*, *element*) pairs (== a list of elements)
- actions: retrieve, store, create which sets/gets the objects

- concrete data type: the **actual outcome** of the type
  - object representation + action implementation
  - for actual coding, per-platform optimization, etc.

(dense 1-D) array in C

- object representation: a block of consecutive memory, with a chunk representing each *element* element for each *index*
- action implementation: [·] for retrieving and storing, `malloc` for creating, etc.

Abstract Data Type (Sec. 1.4)

array consists of ...

- objects: a set of (*index*, *element*) pairs (== a list of elements)
- actions: retrieve, store, create which sets/gets the objects

- abstract data type: the **pseudo essence (contract)** of the type
  - object specification + action specification
  - for illustration, high-level analysis, etc.

abstract 1-D array

- object specification: (*index*, *element*) pairs with $index \in \{0, \cdots, N-1\}$
- action specification:
  retrieve(*index*) returns the *element* associated with *index*;
  store(*index*, *element*) sets *element* to be associated with *index*;
  create(*N*) creates the objects, etc.
  (sometimes with time/space constraints)

will usually look at abstract data type first before going concrete

Reading Assignment

be sure to go ask the TAs or me if you are still confused