Unit #1: Abstract Data Types

CPSC 221: Algorithms and Data Structures

Lars Kotthoff[1]
larsko@cs.ubc.ca

[1]With material from Will Evans, Steve Wolfman, Alan Hu, Ed Knorr, and Kim Voll.

Abstract Data Type

formally mathematical description of an object and the set of operations on the object

in practice interface of a data structure without implementation

Example: Dictionary ADT

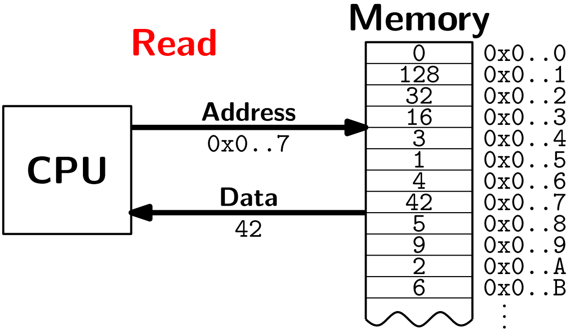
- stores pairs of strings: (word, definition)
- operations:
  - insert(word,definition)
  - delete(word)
  - find(word) $\rightarrow$ definition

Another Example: Array ADT

- store things like integers, (pointers to) strings, etc.
- operations:
  - initialize an empty array that can hold n things
    `thing A[n];`
  - access (read or write) the ith thing in the array $(0 \le i \le n-1)$
    `thing1 = A[i];` Read
    `A[i] = thing2;` Write

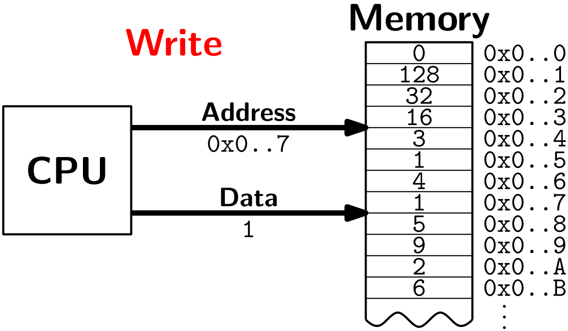
Why Arrays?

- computer memory is an array
- read: CPU provides address i, memory unit returns the data stored at i



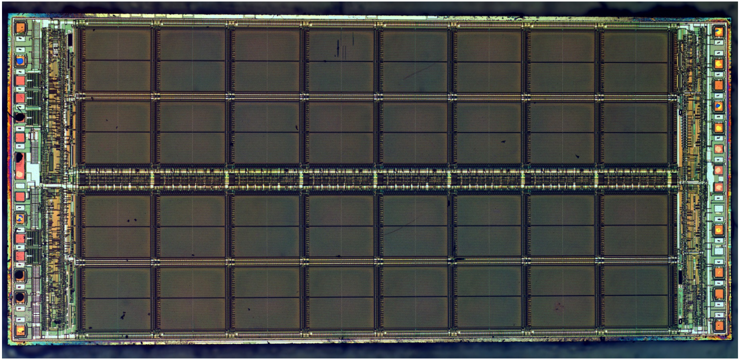
Why Arrays?

- computer memory is an array
- write: CPU provides address i and data d, memory unit stores data d at i



Why Arrays?

Computer memory is an array. Every bit has a physical location.



http://zeptobars.ru/en/read/how-to-open-microchip-asic-what-inside licensed under Creative Commons Attribution 3.0 Unported.

Why Arrays?

- computer memory is an array
- simple and fast
- used in almost every program
- used to implement other data structures

Array limitations

- need to know size when array is created

  Fix: resizeable arrays
  If the array fills up, allocate a new, bigger array and copy the old contents to the new array.

- Indices are integers 0,1,2,...

  Fix: hashing
  (more later)

How would you implement the Array ADT?

Arrays in C++

Create: `int A[100];`

Access:
```
for(int i=0; i<100; i++)
    A[i] = (i+1) * A[i-1];
```

How would you implement the Array ADT?

Arrays in C++

Create: `int A[100];`

Access:
```
for(int i=0; i<100; i++)
    A[i] = (i+1) * A[i-1];
```

Warning No bounds checking!

Data Structures as Algorithms

Algorithm

a high level, language independent description of a step-by-step process for solving a problem

Data Structure

a way of storing and organizing data so that it can be manipulated as described by an ADT

A data structure is defined by the algorithms that implement the ADT operations.

Why so many data structures?

Ideal data structure

fast, elegant, memory efficient

Trade-offs

- time vs. space
- performance vs. elegance
- generality vs. simplicity
- one operation's performance vs. another's

Data structures for Dictionary ADT

- List
- Skip list
- Binary search tree
- AVL tree
- Splay tree
- B-tree
- Red-Black tree
- Hash table

...

Code Implementation

Theory

- abstract base class (interface) describes ADT
- concrete classes implement data structures for the ADT
- data structures can change without affecting client code

Practice

- different implementations sometimes suggest different interfaces (generality vs. simplicity)
- performance of a data structure may influence the form of the client code (time vs. space, one operation vs. another)

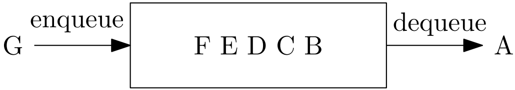
ADT Presentation Algorithm

1. present an ADT
2. motivate with some applications
3. repeat
   3.1 develop a data structure for the ADT
   3.2 analyze its properties
      - efficiency
      - correctness
      - limitations
      - ease of programming
4. contrast data structure's strengths and weaknesses
   - understand when to use each one

Queue ADT

Queue operations

- create
- destroy
- enqueue
- dequeue
- is_empty



Queue property

If x is enqueued before y is enqueued, then x will be dequeued before y is dequeued.

FIFO: First In First Out

Applications of the Queue

- hold jobs for a printer
- store packets on network routers
- hold memory "freelists"
- make waitlists fair
- breadth first search

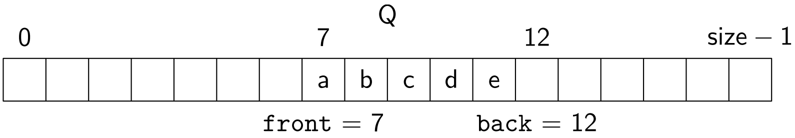
Abstract Queue Example

enqueue R
enqueue O
dequeue
enqueue T
enqueue A
enqueue T
dequeue
dequeue
enqueue E
dequeue

In order, what letters are dequeued?

a. OATE
b. ROTA
c. OTAE
d. None of these, but it **can** be determined from just the ADT.
e. None of these, and it **cannot** be determined from just the ADT.

Circular Array Queue Data Structure




```
void enqueue(Object x) {
  Q[back] = x;
  back = (back + 1) % size;
}

bool is_empty() {
  return (front == back);
}
```

```
Object dequeue() {
  x = Q[front];
  front = (front + 1) % size;
  return x;
}

bool is_full() {
  return (front ==
    (back + 1) % size);
}
```

Circular Array Queue Example

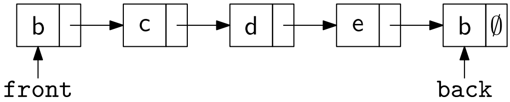
Size = 4

enqueue R
enqueue O
dequeue
enqueue T
enqueue A
enqueue T
dequeue
dequeue
enqueue E
dequeue

What are the final contents of the array queue?

a. RTE
b. RTET
c. TETA
d. TE
e. None

Linked List Queue Data Structure




```
void enqueue(Object x) {
  if (is_empty())
    front = back = new Node(x);
  else {
    back->next = new Node(x);
    back = back->next;
  }
}

bool is_empty() {
  return (front == NULL);
}
```

```
Object dequeue() {
  assert(!is_empty());
  Object ret = front->data;
  Node *temp = front;
  front = front->next;
  delete temp;
  return ret;
}
```

DIY memory management

Circular Array vs. Linked List

- ease of implementation
- generality
- speed
- memory use