CS1003: Intro to CS, Summer 2008

Lecture #06
Data structures II

Instructor: Arezu Moghadam
arezu@cs.columbia.edu

Agenda

- Data structures
  - Arrays
  - Lists
    - Stacks
    - Queues
  - Trees
    - Binary trees
- How to implement data structures
- HW2 review
Data structures

- A computer’s memory is a large open space, and we can organize information in it
- A data structure is an organized entity in this memory space
- It facilitate interpreting the data or performing operations on it
- The most primitive data structures: primitive types

Primitive types

- int, char, double, etc.
- Occupy a well-known amount of memory
  - For 32-bit machines, an char takes 1 byte, an int takes 4 bytes, a double takes 8 bytes
  - Not always the case, but enough for this class
  - sizeof(float);
- The variable refers to that block of memory in its entirety
  - Can’t typically store decimal places inside an int; “won’t fit”
- But what if we want something more complicated?
Why data structures? – Abstraction!

- Primitive datatypes are too primitive
- Often, need to deal with large data
- Enable the application to access and manipulate complex data
- But main memory is not organized as arrays, lists or other proprietary datatypes
- Application doesn’t need to know about details of actual data storage
- Simulate those other data structures!

Arrays

- Arbitrarily defined as a block of memory divided into cells
- To be more precise, an array is a static structure in memory
  - Memory is organized “contiguously” when you define an array
    - 10 integers => 10 * 4 => 40 bytes on a 32-bit machine
  - The variable referring to the array actually just points to the beginning of the appropriate memory location
Arrays (2)

- The programming language then does some math when you use [ ] to access an index in that array...
  - An array of integers, length 10 is at memory location “4000”.
  - How many bytes is this array in total?
  - What’s the position of the 5th integer?
  - Rationale for 0-based makes a little more sense

Heterogeneous Arrays

- Blocks of data items containing different types
- Items within blocks are called components
- Example; structures in C

```c
typedef struct {
    char name[64];
    char course[128];
    int age;
    int year;
} student;

student alex;
```
Lists

- A collection whose entries are arranged sequentially
- Beginning: head
- The end: tail

Stacks can be implemented efficiently and are very useful in computing
- Stacks exhibit the LIFO behavior.
Queue

- A list whose entries are added at the tail and removed from the head
- FIFO discipline

Trees

- A collection with hierarchical organization
- Nodes
  - Root node
  - Leaf nodes
Tree example 1

- Head quarter
  - Seattle office
  - NYC office
  - Boston Office
  - L.A. Office
  - Miami office
  - Sales branches
    - Women’s clothing
    - 5th Ave location
  - Union Square Location
    - Men’s clothing

Tree example 2

- Unix / Windows file system
  - user/courses/
    - cSE16/
      - grades
        - hw1
        - hw2
        - hw3
    - p1
    - p2
    - p3
  - cSE50/
    - grades
    - projects/
      - papers/
    - demos/
    - market
    - buysell
Binary trees

- **Children** are intermediate descendants of a **parent**
- **siblings**, nodes with the same parent
- **Binary trees**, each parent has at most two children

Height of the tree $\log n$

Search, insert and remove information in binary search trees:
$\Theta(\log n)$
Pointers

- How can we implement these data structures?
- Static vs. dynamic data structures
  - An array of 20 names?
  - A queue?
- Every location in the memory is accessed by some address
- Pointer is a storage area that contains such an address

Pointers example

```c
int andy, fred;
int *ted;
fred = andy;
ted = &andy;
```
Bad pointers

- Each pointer should be assigned a pointee
- No error at compile time
- Crash sooner or later at run time!

```c
void BadPointer() {
    int* p;     // allocate the pointer, but not the pointee
    *p = 42;    // this dereference is a serious runtime error
} // What happens at runtime when the bad pointer is dereferenced...
```

Implementing homogenous arrays

- Hourly temperature readings
- An array with 24 elements
Implementing matrices

Access $i$th row and $j$th column:
$$x + (c \times (i - 1)) + (j - 1)$$

Implementing heterogeneous arrays

typedef struct {
    char name[25];
    int age;
    float rate;
} employee;

typedef struct {
    char *name;
    int *age;
    float *rate;
} employee;

a. Array stored in a contiguous block

b. Array components stored in separate locations
Implementing lists

- Linked lists
  - Lists with dynamic structures
  - Head pointer to point to the head of the list
  - NIL pointer to the end of the list

```c
struct list_el {
    int val;
    struct list_el * next;
};

typedef struct list_el item;
```

Deleting an entry from a linked list
Inserting an entry into a linked list

Implementing queues
Implementing queues

b. After inserting entries A, B, and C

c. After removing A and inserting D
Implementing queues

Binary trees

Conceptual tree
Binary trees

Implementation using a linked list

Conceptual tree

Actual storage organization

Root pointer

Binary trees

Implementation using an array

Conceptual tree

Actual storage organization

1 2 3 4 5 6 7

1

A

B

C

D

E

F
Next time

- Midterm
- Lab 5