# CSE 373: Data Structures and Algorithms

Pete Morcos
University of Washington
Spring 2000

http://www.cs.washington.edu/education/courses/cse373/00sp

## What you know already

- control flow (`if`, `while`, `for`)
- basic data types, `struct`
- arrays
- pointers
- memory management (`malloc(), free()`)
  - we'll review the last two next week



## Data Structures

- Programs input, manipulate, and output data
- Need to organize data in a natural way
- Size of data unknown, may vary during execution
- Choice of organization central to program design
  - Some operations become easier or harder
  - Speed of program
  - Memory usage
  - Ease of program maintenance, debugging



## Course goals for data structures

- Study several very important structures, and different implementation techniques
- Learn how to choose the "best" one
- Teach you how to modify standard structures for specific purposes, or create new ones



## Algorithm Analysis

- Algorithm: the sequence of steps a program takes to accomplish a task
- Choice of algorithm has a *huge* impact on efficiency
- Often a tight connection between choice of data structure and choice of algorithm



## Course goals for algorithm analysis

- A bit of theory will give us a framework for comparing algorithms
- See how to weigh advantages and disadvantages
  - usually performance-related
- Study a number of standard algorithms that you'll use often

## Example: expense tracking

- A program to track your spending, so you can see where the money is going

```
typedef char name[120];
typedef enum {FOOD, BOOKS, MUSIC, OTHER} category;
typedef struct _transaction {
   double amount;
   name who;
   category what;
   date when;
} transaction;
```

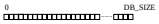


## Expense tracking implementation

- Operations we might want:
  - add()
  - delete()
  - find()
  - subtotal()
- OK, how about:

```
const int DB_SIZE = 10000;
transaction database[DB_SIZE];
```

  - pros? cons?






## Abstract Data Types (ADTs)

- A black box that supports a set of operations
  - In principle, user doesn't know what goes on inside
- Desirable properties:
  - high speed on all operations
  - low memory usage
  - general purpose
- In reality, tension between all these goals forces us to make practical engineering decisions



## Example 2: billiard ball simulator

- It's often useful to use computers to simulate a physical process
  - difficult or expensive to do in reality
  - easier to extract data about the process
- Done by scientists all the time (e.g. galactic collisions); also by engineers (e.g. bridges, new microprocessors)
- Suppose you wanted to simulate billiard balls colliding on a pool table?



## Billiard balls cont'd

```
typedef struct _ball {
   int number;
   double vel_x, vel_y;
   double spin, mass;
} ball;
```

- Decide we want approx. 1 mm accuracy in x, y
- Surface is several feet wide, long
- Important to quickly check for nearby objects



## Billiard balls cont'd

- Operations:
  - move_ball()
  - find_nearest()
  - check_for_collision()
- Try an array, as we did with expense tracker:
  - ball *table[3000][2500];
- Pros? Cons?

## Billiard ball evaluation

- Hmm
- Not obvious how to keep the pros
- We will see some data structures later that will help



## For you to do

- Visit course website
- Sign up for mailing list (details on web page)
- Read Chapter 1
- Find the lab, make sure you can run Visual C++
- Next time: math and C review