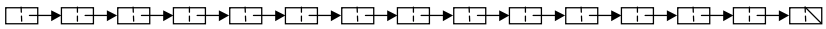
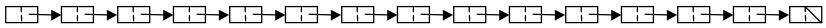


CSE 373: Data Structures and Algorithms

Brad Chamberlain
University of Washington
Autumn 1999

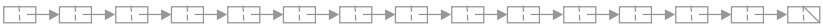
http://www.cs.washington.edu/education/courses/cse373/99au/



What is a Data Structure?

data structure –

Observation

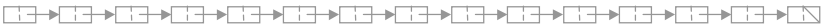


- Data is an attribute common to all programs
  - programs *process, manipulate, store, display, gather*
  - data may be *information*, *numbers*, *images*, *sound*
- Each program must decide how to store data
- Choice influences program at every level:
  - execution speed
  - memory requirements
  - maintenance (debugging, extending, etc.)

Course Goals

- To introduce several standard data structures
- To teach how data structures are evaluated
- To determine when each data structure is useful
- To give you the ability to design, build, and evaluate your own data structures

What about Algorithms?



algorithm – a description of a process useful for completing a specific task

Algorithms are often closely tied to the selection of a data structure (in this class anyway)

C++ Data Types

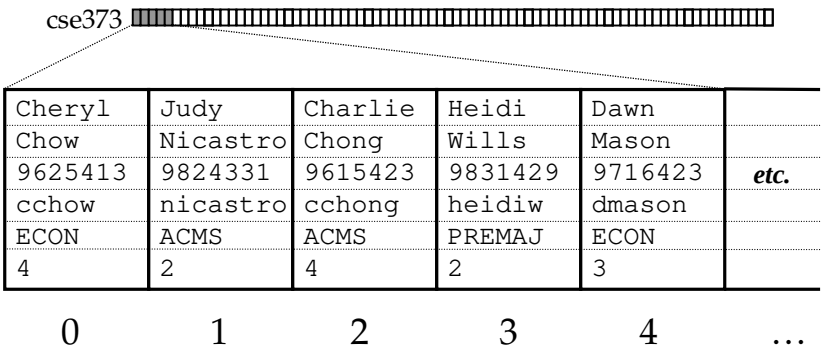
- *basic types:*
  - `char`, `int`, `double`, etc.
  - pointers (*e.g.*, `char *`, `int *`, `double *`)
- *compound types:*
  - arrays (*e.g.*, `int [26]`, `double [100][100]`)
  - structures (e.g., `struct vector { int x,y; double len; }`)
  - classes (*e.g.*, `class point { private: … };`)

Building Data Structures

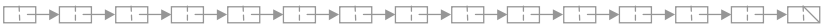
```
typedef char name[32];
typedef enum {ACMS, ECON, EE, MATH, PREMAJ} dept;
typedef struct _student {
          name first, last;
          int UWID;
          name email;
          dept major;
          int year;
        } student;
typedef student course[80];
```


Data Structure course

```
course cse373;
```



Abstract Data Types (ADTs)



abstract data type –

Is the `course` type an ADT?

Example: `FindMajor()`

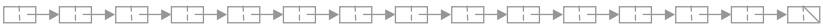
`void FindMajor(course,dept);`

- takes a course and a department as arguments
- prints all students taking the course in that major

How would `FindMajor()` be implemented for our current `course` implementation?

Could changing `course` improve the performance of `FindMajor()`?

ADT Tensions



Ideal: a fast, elegant ADT that uses little memory

Generates tensions:

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

Another Example

- Spring registry ADT for UW – stores which students are taking which courses
- Supported operations:

```
int TakingCourse(int UWID,int SLN);
```

  - tells if the student is taking the course specified by SLN

```
void PrintSchedule(int UWID);
```

  - prints the schedule of the student

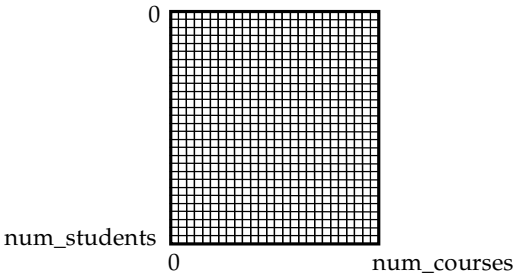
```
course MakeCourseList(int SLN);
```

  - creates a `course` type for the given SLN

A Naive Implementation

```
const int num_courses = 7000;
const int num_students = 33000;

typedef int registry[num_students][num_courses];
```



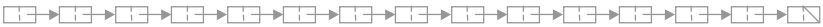
Evaluating this Implementation

What are the advantages of this implementation?

What are the disadvantages?

How could we improve the implementation?

The Myth of ADTs



Not a perfect black box:

- knowing how an ADT will be used can lead to a good choice of implementation
- also, knowledge of an ADT's implementation may change how a client uses it

But... ADTs are still a useful concept

Use motivates design

Todo

- Look over class web page
- Join class e-mail list (see web page for details)
- Find MSCC PC Lab and try logging in
- Read chapters 1 and 2 of the textbook