Engineering 4892
Data Structures

Dennis Peters
dpeters@engr.mun.ca
http://www.engr.mun.ca/~dpeters/
EN-3061, 737-8929

Summer 2003

Administrivia

Evaluation:  Assignments (6): 15%
Quizzes (2): 30% June 4, July 16
Final: 55%

Lab/Tutorial:  Tuesday, 1400-1650 EN-3000/29
Office hour:  Wednesday 1400-1500, or by appointment (or not).
Web page:  http://www.engr.mun.ca/~dpeters/4892/

Textbook


Software

We will be using ANSI/ISO C++.

We’ll be testing using Cygwin GNU C++, so it’s your responsibility to be sure your code works with it.

• Available on CD for a small fee.
• Download it from http://sources.redhat.com/cygwin/.

Assignments

• Mostly programming.
• Due Thursdays at 0900 (9 am).
• Electronic submission using “Web Submit”
• Questions posted on the web page.
• Do your own work!
Motivation

Consider a product from another Engineering discipline (e.g., building, bridge, car, boat etc.):

We can consider it from several points of view:

Components
- What are the components?
- What are their specifications?

Implementation
How are the components constructed (from raw materials and other components)?

Architecture
Arrangement of components.

The components are based on well-known (often mathematical) models: circle, plane, cone, arch, triangle.

Chosen for beauty, simplicity, cost, function.

In software, what are the components?

- Subroutines
- Variables
- Modules
- Types

Types are the main topic of this course.

If the implementation of a type is “hidden” it’s called an abstract data type (ADT).

Abstract Data Types

By abstract we mean that there’s more than one way that it could be implemented — which one we use doesn’t matter to the user of the type.

ADT may be built in or user defined.

Consider two implementations of a simple ADT:

```c++
class Complex {
    private:
        double re, im;

    public:
        // ...
        Complex operator +(Complex r);
        Complex operator *(Complex r);
};
```

```c++
class Complex {
    private:
        double mag, theta;

    public:
        // ...
        Complex operator +(Complex r);
        Complex operator *(Complex r);
};
```

The ADT is defined by what these have in common — their interface.

How would you choose between the implementations?

- Cost of implementing.
- Chance of making errors.
- Cost of use: One is fast for additions, other is fast for multiplications.
- Accuracy.
Programming in the Large vs. in the Small

In the small . . .

- Creating a single component or a small number of related components.
- E.g., a small program (< 1000 lines)

In the large . . .

- Programming by putting together other components.
- E.g., a medium or large program (> 100,000 lines) consisting of many classes.

In programming in the large abstraction is essential to success.

Big vs. Little Changes

A big change is one that will “break” code (i.e., require changes) in other modules.

From the C++ FAQ:

Q: How do developers determine if a proposed change will be big or little?
A: Specification

With proper specification, maintenance programmers can easily distinguish between big or little changes. Ill-specified systems typically suffer from “change phobia”: if anyone even contemplates changing anything, everyone starts sending out their résumés for fear that the system will collapse. Unfortunately, changes often do make the world fall apart in ill-specified systems. It’s called maintenance cost, and it eats software companies alive.

Objectives

- More advanced programming
- Architecture: Learn to think about (and solve) programming problems in terms of separate components.
- Component design: Learn some standard varieties of ADTs
  - their interface (specification),
  - how to use them, and
  - how to implement them.
- Learn some common forms of algorithms (another kind of component).
- Understand abstraction (a.k.a. information hiding, separation of concerns): Do not let the implementation of components (ADTs and algorithms) show in the interface.