CS 2351 Data Structures

Course Overview

Prof. Chung-Ta King

Department of Computer Science

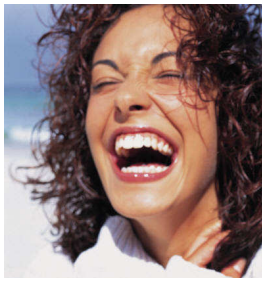
National Tsing Hua University

Outline

- Why is data structure important?
  (Why take this course?)
- Course information
- Why C++, not C? (Sec. 1.2, 1.3)

Why Is Data Structure Important?

- Suppose you keep a record of one million (10^6) data items for query, which are stored in a random order
  - You expect to receive 10 queries per second, each needs to search through ½ of the 10^6 data items in average
  - With a computer capable of 10^9 lookups per second, the 10 queries will take $0.5 \times 10^6 \times 10 / 10^9 = 0.005$ sec to answer
  - You are quite happy with your system!



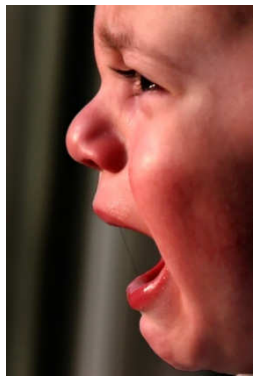
- As your business grows, you have 10^8 data items and receive 1000 queries per second
  - You now need $0.5 \times 10^8 \times 1000 / 10^9 = 50$ sec to answer these 1000 queries; by then, you have 49,000 queries in queue

(Open Data Structures (in C++), Pat Morin)

Why Is Data Structure Important?

- Real problems occur when your problem size becomes BIG
- To match the incoming queries, you could buy 50 computers or 1 computer that 50 times faster
- In fact, no hardware upgrade is needed if you organize the 10^8 data items in a proper **structure**
  - Unfortunately, your program is already online and changing its data structure may be too expensive

  → importance of choosing right data structure



(Open Data Structures (in C++), Pat Morin)

Why Is Data Structure Important?

- Data structure is important because it dictates
  - The types of operations that can perform on the data
  - How efficiently these operations can be carried out
  - How dynamic we can be in dealing with the data
    - For example, whether we can add additional data on the fly or if we need to know about all of the data up front
- It is often your insight in organizing the data determines how you solve a problem
- And, your way of solving a problem determines how *efficiently* the problem can be solved

What Is Data Structure?

- *Data structure* is concerned with the representation and manipulation of data
  - A data structure is **data representation + associated operations**
  - Often, we are more concerned about the organization or structuring for a *collection* of data items
- Representation:
  - How to organize data into a specialized structure such that it could be used and manipulated efficiently?
- Manipulation:
  - What operations can be performed? How efficiently?

Data Structure and Algorithm

- A problem solvable by a computer often requires some input data and produce some output data
  - Analog to a math function, y = f(x), or making a dish
- To solve a problem, you need to
  - Organize the data in a proper data structure
  - Manipulate the data (using the set of associated operations) step by step to produce output data
    - The process followed to solve the problem is called an algorithm
    - Need to differentiate between the operations to manipulate a data structure and the operations to solve a problem
      Ex.: traveling route planning

Outline

- Why is data structure important?
- Course information
- Why C++, not C? (Sec. 1.2, 1.3)

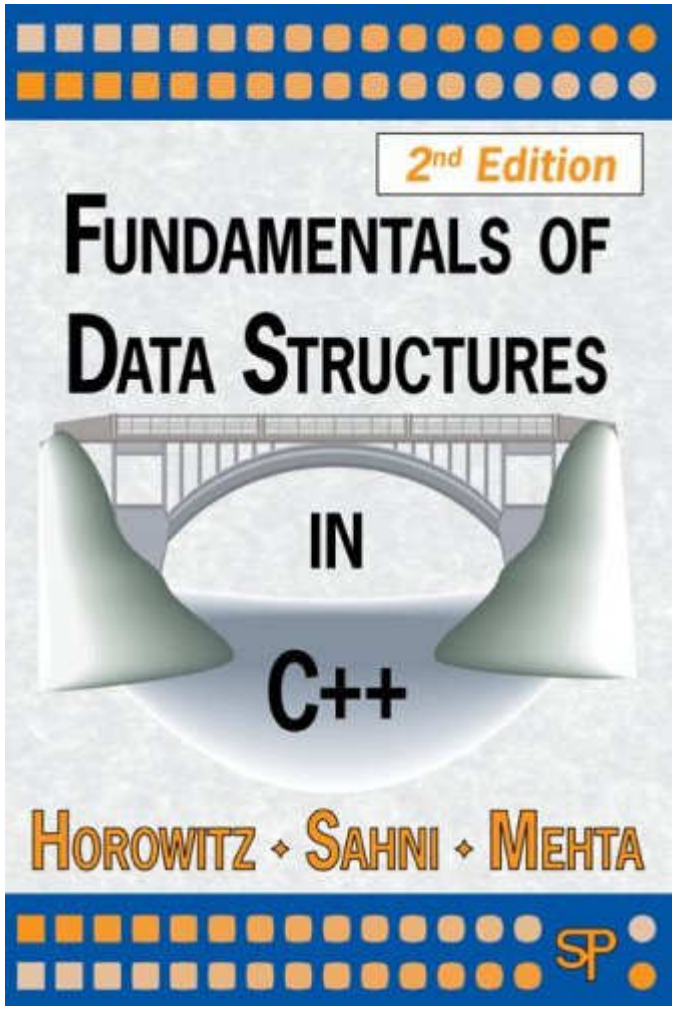
What Will We Learn in This Course?

- Learn techniques to design and implement large-scale computer programs
  - Data abstraction and encapsulation, algorithm specification, performance analysis and measurement
  - Basic data structures to represent data: arrays, stacks, queues, linked lists, trees, graphs, …
  - Basic algorithms to manipulate data structures: sorting, string matching, matrix multiplication, shortest paths, …
- Data structures play a key role in other courses:
  - Algorithms, Compilers, Image Processing, Computer Graphics, …

Textbook

Fundamentals of Data Structures in C++

2nd ed., Ellis Horowitz, Sartaj Sahni, Dinesh Mehta



You are expected to read the textbook!

Course Outline

Topics	Textbook
Intro. to C++ and Algorithm	Chapter 1
Arrays	Chapter 2
Stacks and Queues	Chapter 3
Linked Lists	Chapter 4
Trees	Chapter 5
Graphs	Chapter 6
Sorting	Chapter 7
Hashing	Chapter 8

Course Information

- Instructor: Prof. Chung-Ta King (金仲達教授)
  - Office: Delta 640 Phone: x42804
  - email: king@cs.nthu.edu.tw
- Teaching Assistants: 王泰元、吳亞潔、李荏敏、柯安琪
  - Office: CSEE 734 Phone: x33553
- Class Time:
  - Monday 10:10 - 12:00
  - Wednesday 9:00 - 9:50
- Classroom: Delta 104
- http://www.cs.nthu.edu.tw/~king/courses/cs2351.html

Expected Workload and Grading

- 6 assignments: 30%
- 5 (out of 10) on-line quizzes, 2 from easy part and 3 from difficult part, with highest class averages: 15%
  - On-line quizzes are held in the PC rooms 19:30 – 22:30 on selected Tuesdays; details to be announced
- Midterm exam: 20%
- Final project: 25%
- Class participation (in-class quizzes and Q&A): 6%
- Taking 程式檢定 at least once: 4%

Course Conducts

- You can discuss with your classmates about the assignments, but the final work must be your own!
- You are not allowed to discuss with your classmates during the exams or online tests
- Any caught cheating in assignment, quiz, exam or final project will receive a grade of **ZERO**

My Expectations

- For my part:
  - To help you to learn data structures well
  - Try to stimulate class interactions
- For your part:
  - Strong motivation to learn and persistence
  - Learn to learn proactively
  - Understand that there is often no single answer
  - Summarize in your own words whatever you learned
  - Ask me if you do not understand
  - Tell me if you cannot keep up
  - Take 程式檢定

Outline

- Why is data structure important?
- Course information
- Why C++, not C? (Sec. 1.2, 1.3)
  (encapsulation and abstraction, abstract data type, object-oriented programming)

Why Not C?

- C adopts *algorithmic* (*functional*) *decomposition*
  - You program by specifying the steps to solve the problem, similar to that of writing an *algorithm* or cooking recipe
  - Steps are often implemented as C functions and thus you decompose the software into *functions*
  - Data structures are secondary and often afterthought
    → visible and accessible to potentially all steps
    - Unclear by which function a data structure is manipulated
    - No way to prevent irrelevant code to access a data structure
  - Difficult to reuse code

Why Not C? (Review of Data Type)

- Consider following declaration in C or in any lang.:

```
int i,j;
```

  - You specify that **i** and **j** are of type **int**
- When you declare that a variable has a certain data type, you are saying that
  - The values that the variable can have are elements of a certain set and have a certain representations
  - There is a collection of operations that can be applied to those values, e.g., +, –, ×, ÷, …
- How about user-defined data types?

Why Not C?

- Suppose you define a type called **`building`**:

```
typedef struct {
  double long; // Longitude of building
  double lati; // Latitude of building
  char* owner; // owner of the building
} building;
building A,B;
```

  - Can you specify a set of operations that (i) only those operations (ii) in designated locations in your code can manipulate **A** and **B**, e.g., **`A-B`**?
  - Can you hide content of **`owner`** from other parts of code?

(1) data objects + operations; (2) hiding

Why Not C?

- Suppose later you want to define type **house**:

```
typedef struct {
  double long; // Longitude of building
  double lati; // Latitude of building
  char* owner; // owner of the building
  int rooms; // # of rooms in the house
} house;
house C,D;
```

  - How can you tell **house** is a specialization of **building**?
  - How can you specify that all operations valid for **building** are legal for **house**, except some?

inheritance

Why C++?

Ideal for studying data structure

- C++ follows *object-oriented decomposition*
  - Programming is to specify a set of well-defined objects that interact with each other to solve the problem
    → force you to focus on data first, process second
  - Objects are entities that contain data (local states) and operations (functions) to perform computation (vs variables in C), and are instances of classes (vs types in C)
- The **`class`** construct allows new data types defined
  - Have a concrete representation of the objects of the type
  - A set of operations for manipulating the objects
  - No other operations or part of code can manipulate the objects (they only know interface but not implementation)

Abstract data type

Why C++?

- Example specification of *abstract data types* (ADT)

```
ADT NaturalNumber is
objects: integers from 0 to maximum integer (MAXINT) on the computer
functions: // operations
  Zero( ):NaturalNumber     ::= 0
  IsZero(x):Boolean         ::= if (x==0) IsZero=TRUE else IsZero=FALSE
  Add(x, y):NaturalNumber ::= if (x+y <= MAXINT) Add= x+y
                                else Add=MAXINT
  Equal(x,y):Boolean        ::= if (x== y) Equal=TRUE else Equal=FALSE
  Successor(x):NaturalNumber ::= if (x == MAXINT) Successor=x
                                   else Successor= x+1
  Subtract(x,y):NaturalNumber ::= if (x<y) Substract=0 else Substract=x-y
end NaturalNumber
```

(page 10 of textbook)

Observations from the Example

- Declaration of data type and operations on objects of the type are defined in one syntactic unit
  - Allow you to specify a set of operations, and only this set of operations, that can operate on objects of the type
    - Ex.: traveling agents, ticket purchase on-line
  - For example:

```
building A,B;
distance = A - B;
```

Observations from the Example

- Only specify interface, not implementation
  - *Interface* describes what a data structure does
    - Defines the set of values and associated operations
    - Ex: piano, telephone, bicycle
  - *Implementation* describes how the data structure does it
    - Includes internal representation of the data structure and definitions of the algorithms that implement the operations
    - There can be many implementations for an interface
  - Data abstraction: separation between specification of a data object and its implementation → abstract data type
    - Manage complexity, facilitate code maintenance and reuse, reduce errors

Observations from the Example

- Implementation is hidden
  - Representation of objects of the type and implementation of operations are hidden from the program units that use these objects, so the only operations possible are those provided in the type's interface
  - Data encapsulation: hiding of the implementation details of a data object from the outside world
  - Ex.: hide **owner** and operation implementations:

```
typedef struct {
  double long; // Longitude of building
  double lati; // Latitude of building
  char* owner; // owner of the building
} building;
```

Why C++?

- C++ also supports inheritance
  - **`house`** can inherit from **`building`**

- C++ supports Object-Oriented Programming (OOP):
  - Objects are fundamental building blocks (**A** and **B**)
  - Each object is an instance of some type or class (**`house`**)
  - Objects are related to each other by inheritance

Thus, we use C++ in this course

Summary

- Data structure, data representation and associated manipulation operations, is critical to the efficiency of programs
- The course on data structure is of fundamental importance for computer science
- C++ is an object-oriented language that supports abstract data types and inheritance
- C++ forces us to focus on data objects, which makes it ideal for studying data structures