Data Structure

South China University of Technology

College of Software Engineering

Huang Min

教 学 安 排

本课程为华南理工大学SPOC课程，采用线上线下混合式教学方式。

- 基本理论知识（48学时）
  - 线上SPOC视频等资料的学习(18学时)(1-9周，每周2学时)
  - 线下课堂面授(30学时)(1-10周，每周3学时)

- 实验课（线下，16学时，共4次，每次4学时）

SPOC和MOOC有什么区别？

- MOOC是Massive Open Online Course，大规模在线开放课程，面向所有学员，译为“慕课”;

- SPOC是Small Private Oline Course，小规模限制性在线课程，译为“私播课”。（同学们在平台上的学习数据，包括作业成绩等，均可从SPOC平台导出。）

翻转课堂线上和线下课程安排

- 线下翻转课堂　（30学时）

（1） 时间：第1-10周

20级软件工程卓越班：每周周二晚上第9-11节课；

20级软件工程2班：每周周一下午第5-7节课；

（2） 地点：　A2-402 （大学城智慧教室）

- 线上SPOC视频等资料学习 （18学时）

（1） 时间：第1-9周，每周周三-周五 （2学时）；

（2） 线上网址：学堂在线平台：

https://scut.yuketang.cn/pro/portal/about/849VCAVqXy7

（3）访问登录方式：见《学堂云学生操作手册（长江雨课堂环境）》

课程考核方式

- 总评成绩

线下课堂表现（20%）+线上SPOC视频学习和作业情况（15%）+实验（15%）+期末考试（50%）。

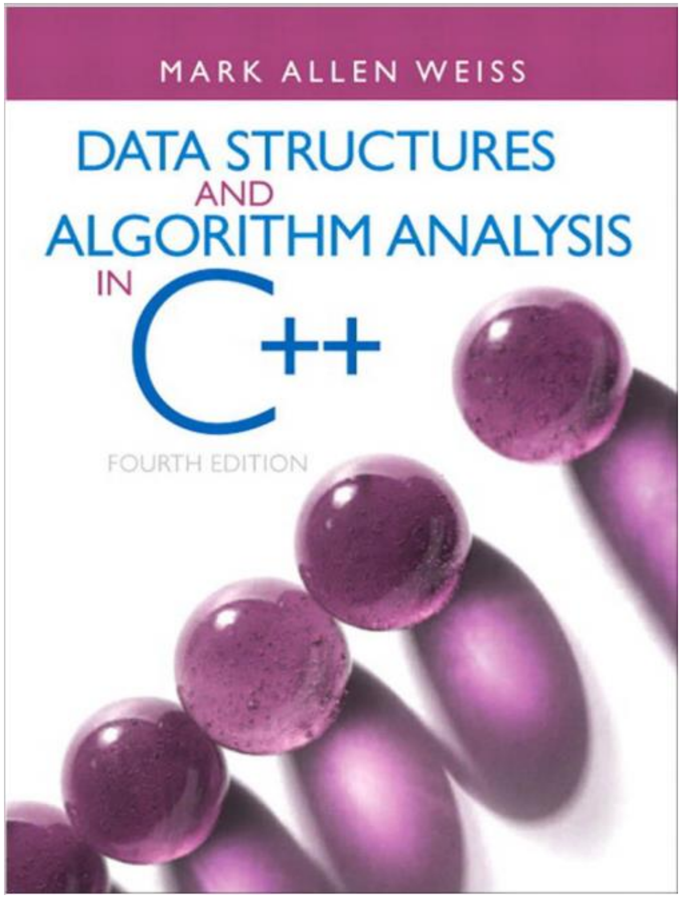
（1）线下课堂表现为翻转课堂的出勤、问题回答、参与讨论等情况。

（2）线上SPOC视频学习和作业等情况，由SPOC系统导出，包括SPOC线上课程资源学习情况、单元作业及讨论完成情况。未观看完学习视频的，将酌情减分。

（3）实验的考核内容为线下实验课出勤和实验完成情况。

（4）期末考试为线下闭卷考试。

Textbook



- 《Data Structures and Algorithm Analysis in C++ (4e)》
- by Mark Allen Weiss

Reference Books

- Data Structures and Algorithm Analysis (C++) (3e), by Clifford A. Shaffer

Class Overview

- Introduction to many of the basic data structures used in computer software
    - Understand the data structures
    - Analyze the algorithms that use them
    - Know when to apply them
- Practice design and analysis of data structures.
- Practice using these data structures by writing programs.
- Data structures are the plumbing and wiring of programs.

Goal

- You will understand
  - what the tools are for storing and processing common data types
  - which tools are appropriate for which need
- So that you will be able to
  - make good design choices as a developer, project manager, or system customer

Course Topics

- Introduction to Algorithm Analysis
- Lists, Stacks, Queues
- Search Algorithms and Trees
- Hashing and Heaps
- Sorting
- Disjoint Sets
- Graph Algorithms

Self-reading

- Chapters 1-Programming: A general overview
  - Very important sections:
    - Section 1.2 on Mathematics Review
    - Section 1.3 on Recursion
    - Section 1.6 on Templates

Data Structures: What?

- Need to organize program data according to problem being solved
- Abstract Data Type (ADT) - A data object and a set of operations for manipulating it
  - List ADT with operations **`insert`** and **`delete`**
  - Stack ADT with operations **`push`** and **`pop`**
- Note similarity to C++/Java classes
  - private data structure and public methods

ADT　　线性表的抽象数据类型描述

ADT Linear List {

数据对象：D={ai| ai /ElemSet，i=1，2，…，n，n≥0}

数据关系：R1={<ai-1，ai>|ai-1，ai/D，i = 2，…，n}

基本操作：

InitList（&L）

初始条件：线性表L不存在。

操作结果：构造一个空的线性表L。

ListEmpty（L）

初始条件：线性表L已存在。

操作结果：若L为空表，则返回TRUE，否则返回FALSE。

ListLength（L）

初始条件：线性表L已存在。

操作结果：返回L中数据元素个数。

……

ListDelete（&L，i，&e）

初始条件：线性表L已存在且非空，1≤i≤ListLength（L）。

操作结果：删除L中第i个数据元素，并用e返回其值，L的长度减1。

}ADT LinearList

Data Structures: Why?

- Program design depends crucially on how data is structured for use by the program
  - Implementation of some operations may become easier or harder
  - Speed of program may dramatically decrease or increase
  - Memory used may increase or decrease
  - Debugging may be become easier or harder

Terminology

- Abstract Data Type (ADT)
  - Mathematical description of an object with set of operations on the object. Useful building block.
- Data structure
  - A specific family of algorithms for implementing an abstract data type.
- Implementation of data structure
  - A specific implementation in a specific language
- Algorithm
  - A high level, language independent, description of a step-by-step process

Algorithm Analysis: Why?

- Correctness:
  - Does the algorithm do what is intended.
- Performance:
  - What is the running time of the algorithm.
  - How much storage does it consume.
- Different algorithms may correctly solve a given task
  - Which should I use?

Iterative Algorithm for Sum

- Find the sum of the first **num** integers stored in an array **v**.

```
sum(v[ ]: integer array, num: integer): integer{
    temp_sum: integer ;
    temp_sum := 0;
    for i = 0 to num - 1 do
        temp_sum := v[i] + temp_sum;
    return temp_sum;
}
```

Note the use of pseudocode

Programming via Recursion

- Write a *recursive* function to find the sum of the first **num** integers stored in array **v**.

```
sum (v[ ]: integer array, num: integer): integer {
    if num = 0 then
        return 0
    else
        return v[num-1] + sum(v,num-1);
}
```

base case

recursive case

Pseudocode

- In the lectures algorithms will be presented in pseudocode.
  - This is very common in the computer science literature
  - Pseudocode is usually easily translated to real code.
  - This is programming language independent
- Pseudocode should also be used for homework

Proof by Induction

- **Basis Step:** The algorithm is correct for the base case (e.g. n=0) by inspection.
- **Inductive Hypothesis (n=k):** Assume that the algorithm works correctly for the first k cases, for any k.
- **Inductive Step (n=k+1):** Given the hypothesis above, show that the k+1 case will be calculated correctly.

Program Correctness by Induction

- **Basis Step:** sum(v,0) = 0. ✓
- **Inductive Hypothesis (n=k):** Assume sum(v,k) correctly returns sum of first k elements of v, i.e. **`v[0]+v[1]+…+v[k-1]`**
- **Inductive Step (n=k+1):** sum(v,n) returns **`v[k]+sum(v,k)`** which is the sum of first k+1 elements of v. ✓

Algorithms vs Programs

- Proving correctness of an algorithm is very important
  - › a well designed algorithm is guaranteed to work correctly and its performance can be estimated
- Proving correctness of a program (an implementation) is fraught with weird bugs
  - › Abstract data type, which is implemented by programming language, constitutes a bridge between logical structure and storage structure, so is a way to bridge the gap between mathematical algorithms and programs