Of Studies

Francis Bacon

王佐良译　　(通过文化学习语言，语言会学的更好，是学语言后面的大的精神世界)

Studies serve for delight, for ornament, and for ability. Their chief use for delight, is in privateness and retiring; for ornament, is in discourse; and for ability, is in the judgment and disposition of business.

读书足以怡情，足以傅彩，足以长才。其怡情也，最见于独处幽居之时；其傅彩也，最见于高谈阔论之中；其长才也，最见于处世判事之际。

Histories make men wise; poets witty; the mathematics subtile; natural philosophy deep; moral grave; logic and rhetoric able to contend. Abeunt studia in morse. (Studies go to make up a man's character.)

读史使人明智，读诗使人灵秀，数学使人周密，科学使人深刻，伦理学使人庄重，逻辑修辞之学使人善辩；凡有所学，皆成性格。

Information Science and Technology College of Nort
heast Normal University

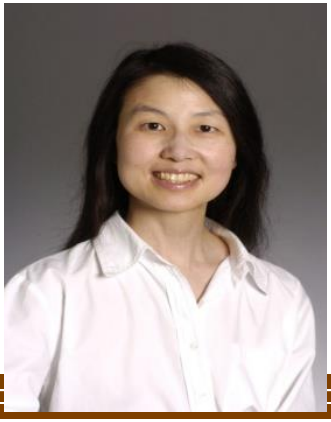
Compiling and Running of Program

Dr. Zheng Xiaojuan
Professor

September. 2019

计算思维
Computational Thinking

- 计算思维基本概念
  - Jeannette M. Wing, （周以真）Computational Thinking, Communications of ACM, Vol.49, No.3, 2006, pp.33-35.
  - 被认为是近十年来产生的最具有基础性、长期性的学术思想，成为21世纪计算机科学研究和教育的热点。



Viewpoint | Jeannette M. Wing

Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is

计算思维
Computational Thinking

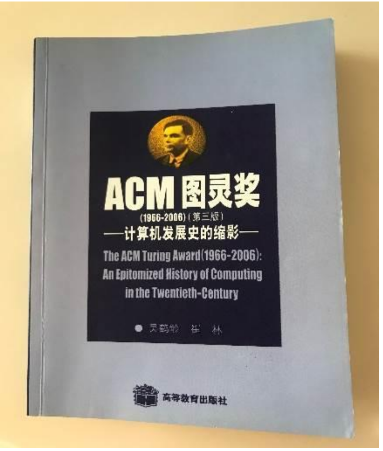
- 计算思维是什么[J. Wing, 2006]
  - 计算思维是运用计算机科学的基础概念去求解问题、设计系统和理解人类的行为，它包括了一系列广泛的计算机科学的思维方法
  - 计算思维和阅读、写作和算术一样，是21世纪每个人的基本技能，而不仅仅属于计算机科学家
  - 计算思维在生物、物理、化学、经济学、统计学等其他学科中的影响已经显现

计算思维

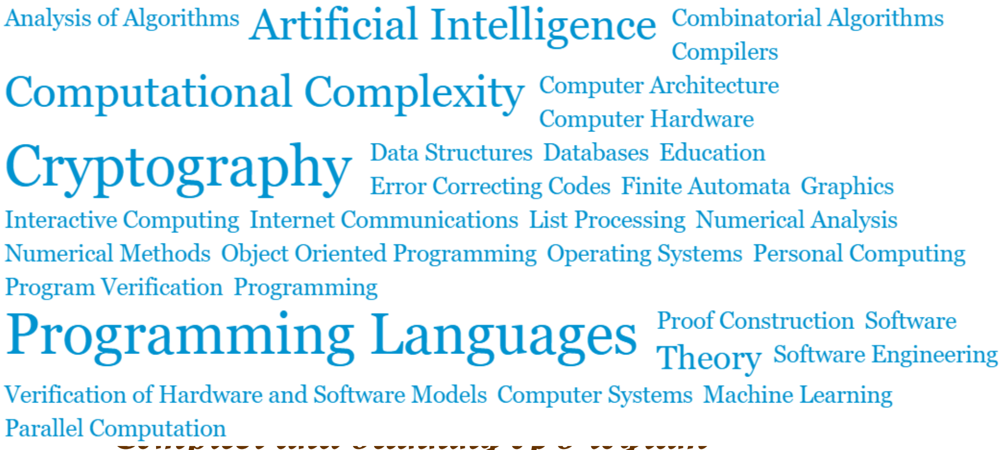
Computational Thinking

- 包括一系列广泛的计算机科学的思维方法
  - 抽象
  - 自动化
  - 问题分解
  - 递归
  - 权衡
  - 保护、冗余、容错、纠错和恢复
  - 利用启发式推理来寻求解答
  - 在不确定情况下的规划、学习和调度
  - . . .

计算思维 vs. 编译



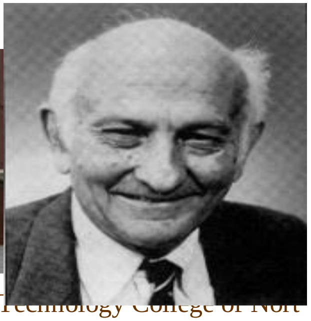
- 编译理论与技术
  - 计算机科学与技术中理论和实践相结合的最好典范
  - 体现了很多典型的计算思维方法
- ACM 图灵奖
  - 授予在计算机技术领域作出突出贡献的科学家
  - 程序设计语言、编译相关的获奖者是最多的



Analysis of Algorithms Artificial Intelligence Combinatorial Algorithms Compilers Computational Complexity Computer Architecture Computer Hardware Cryptography Data Structures Databases Education Error Correcting Codes Finite Automata Graphics Interactive Computing Internet Communications List Processing Numerical Analysis Numerical Methods Object Oriented Programming Operating Systems Personal Computing Program Verification Programming Programming Languages Proof Construction Software Theory Software Engineering Verification of Hardware and Software Models Computer Systems Machine Learning Parallel Computation

http://amturing.acm.org/bysubject.cfm

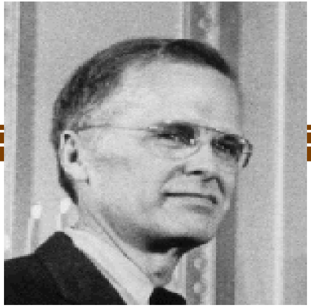
Information Science and Technology College of Nort
heast Normal University



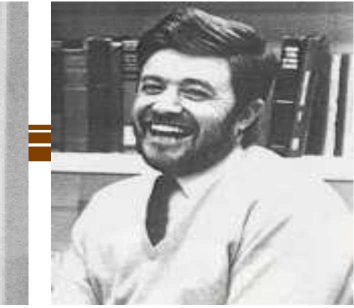
John Cocke

Alan J. Perlis

Edsger W. Dijkstra



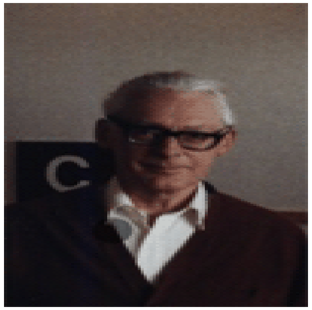
John W. Backus



Kenneth E. Iverson

- Alan J. Perlis (1966) -- ALGOL
- Edsger Wybe Dijkstra (1972) -- ALGOL
- Michael O. Rabin & Dana S. Scott (1976) --非确定自动机
- John W. Backus (1977) -- FORTRAN
- Kenneth Eugene Iverson (1979) -- APL程序语言
- Niklaus Wirth (1984) -- PASCAL
- John Cocke (1987) -- RISC & 编译优化
- O. Dahl, K. Nygaard (2001) -- Simula语言和OO概念
- Alan Kay (2003) -- SmallTalk语言和面向对象程序设计
- Peter Naur (2005) -- ALGOL60以及编译设计
- Frances E. Allen(2006)-- 优化编译器
- Barbara Liskov(2008)--编程语言和系统设计的实践与理论
- …

K.Nygaard

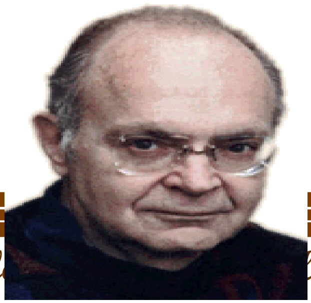


O. Dahl

Dana S. Scott

Michael O. Rabin

Niklaus Wirth

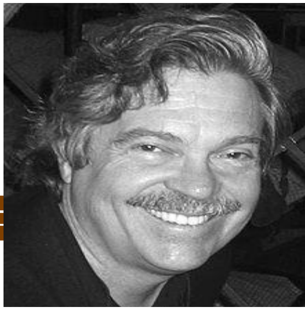


Donald E. Knuth

Barbara Liskov

Frances E. Allen

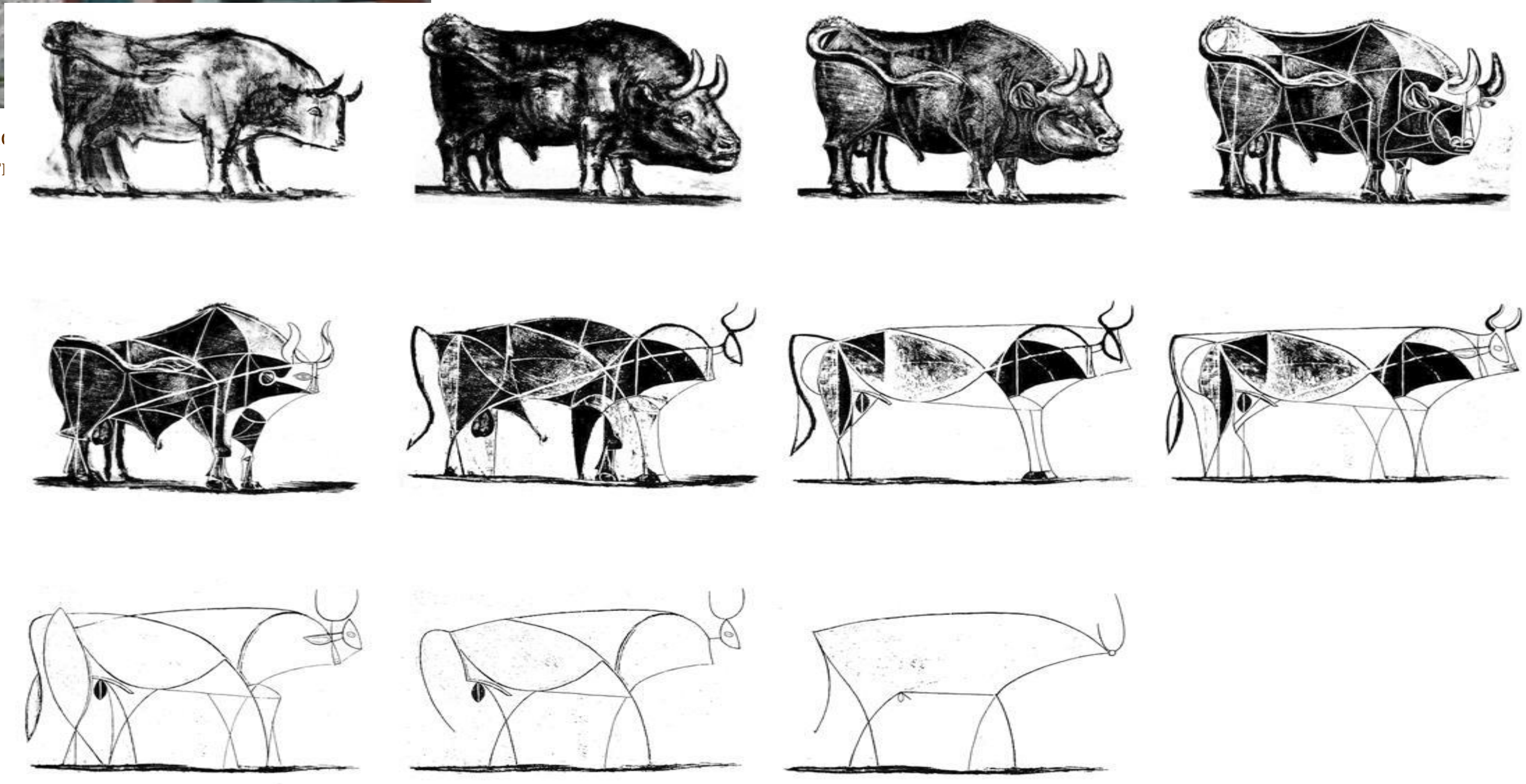
Peter Naur



Alan Kay

计算思维 vs. 编译

- 抽象(Abstraction)
  - 忽略一个主题中与当前问题(或目标)无关的那些方面，以便更充分地注意与当前问题(或目标)有关的方面
  - 从众多的事物中抽取出共同的、本质性的特征，舍弃其非本质的特征
  - 是一种从个体把握一般、从现象把握本质的认知过程和思维方法

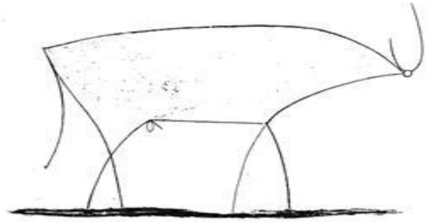
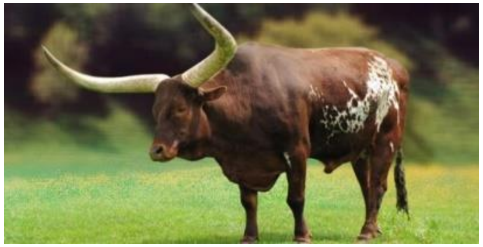


毕加索《牛》

毕加索认为，画家的职责不是借助具体形象反映现实, 而是以抽象的形象表达科学的真实。

抽象的例子

- 从抽象的观点看

 是对  的抽象

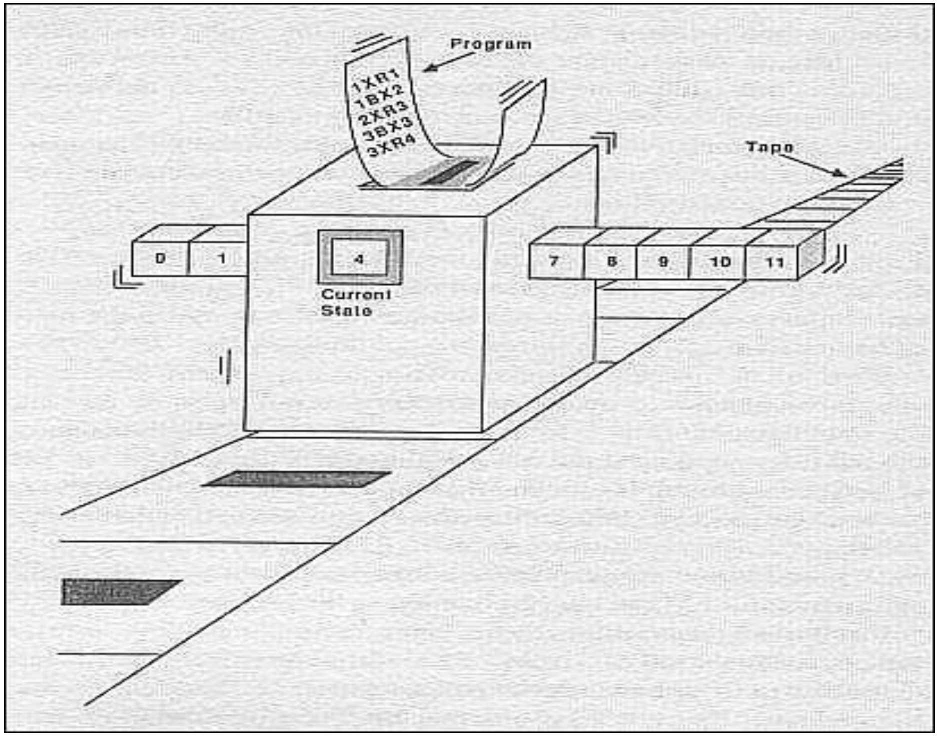
质点 是对 物理对象 的抽象

化学分子式 是对 化合物 的抽象

计算机科学中的抽象？

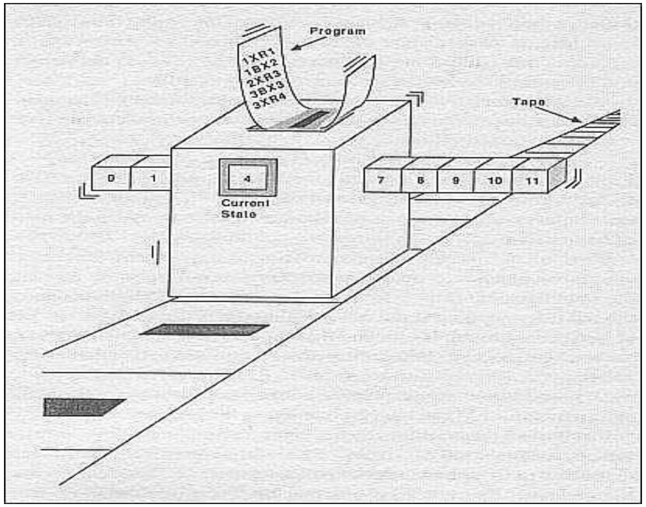
计算思维 vs.编译

- 抽象(Abstraction)
  - 图灵机



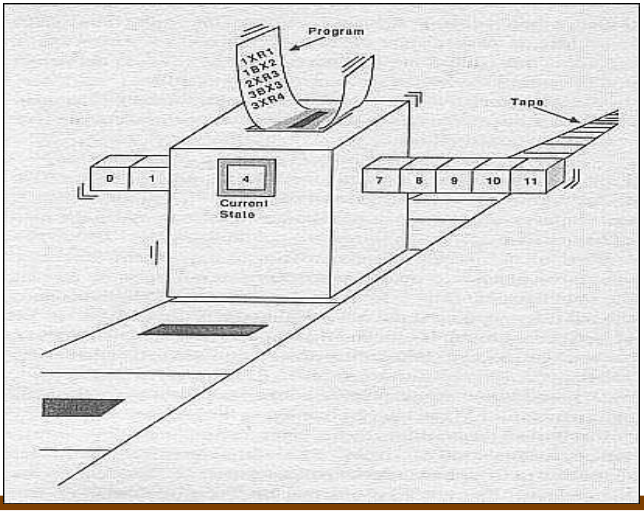
计算思维 vs. 编译

- 抽象(Abstraction)
  - 图灵机
    - 一条无限长的纸带
    - 一个读写头
    - 一个状态寄存器
    - 一套控制读写头工作的规则



计算思维 vs.编译

- 抽象(Abstraction)
  - 图灵机
  - 邱奇-图灵论题(The Church-Turing thesis)
    - 所有计算或算法都可以由一台图灵机来执行
  - 可计算=图灵可计算



计算思维 vs. 编译

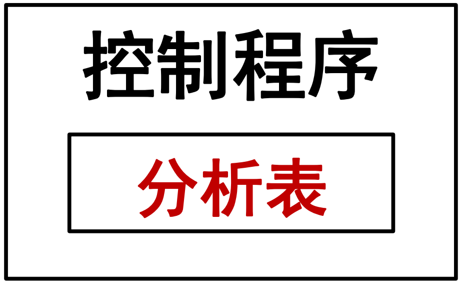
- 编译原理中的"抽象"
  - 有限自动机
  - 形式文法
  - …

计算思维 vs. 编译

- 自动化(Automation)
  - 将抽象思维的结果在计算机上实现，是一个将计算思维成果物化的过程，也是将理论成果应用于技术的实践
  - 自动化的思维方法不仅体现在编译程序本身的工作机制上，更体现在了编译程序的生成工具的研究和设计上

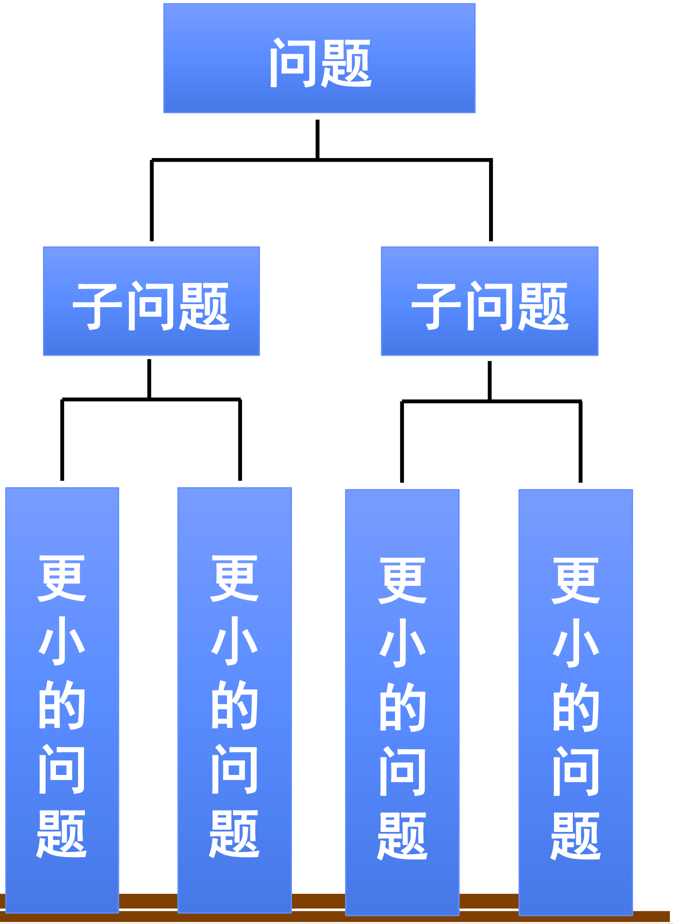
计算思维 vs. 编译

- 编译原理中的"自动化“
  - 有限自动机
  - 预测分析程序---LL（1）分析
  - LR分析
  - …



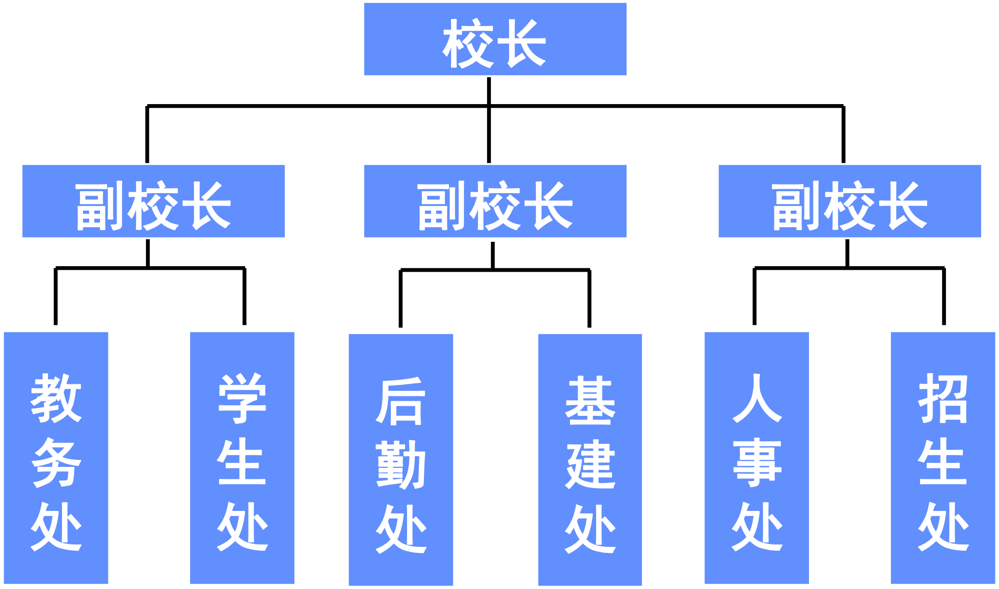
计算思维 vs. 编译

- 分解(Decomposition)
  - 将大规模的复杂问题分解成若干个较小规模的、更简单的问题加以解决
    - 对问题本身的明确描述，并对问题解法作出全局性决策
    - 把问题分解成相对独立的子问题
    - 再以同样的方式对每个子问题进一步精确化，直到获得对问题的明确解答



计算思维 vs. 编译

- 分解(Decomposition)
  - 层次化管理



校长

副校长

副校长

副校长

教务处

学生处

后勤处

基建处

人事处

招生处

计算思维 vs. 编译

- 编译原理中的"问题分解"
  - 为什么编译程序引入中间语言？
  - 为什么编译分成多个阶段？
  - 为什么分析过程分成多遍？
  - …

计算思维 vs. 编译

- 递归(Recursion)
  - 问题的解决依赖于类似问题的解决，只不过后者的复杂程度或规模较原来的问题更小
  - 一旦将问题的复杂程度和规模化简到足够小时，问题的解法其实非常简单

计算思维 vs. 编译

- 编译原理中的"递归"
  - 递归下降分析
  - 基于树遍历的属性计算
  - 语法制导翻译
  - …

计算思维 vs. 编译

- 权衡(折衷, Tradeoff)
  - 理论可实现 vs. 实际可实现
  - 理论研究重在探寻问题求解的方法，对于理论成果的研究运用又需要在能力和运用中作出权衡

计算思维 vs. 编译

- 编译原理中的"权衡"
  - 用上下文无关文法来描述和处理高级程序设计语言
  - 优化措施的选择
  - . . .

计算思维 vs. 编译

- 计算思维包括一系列计算机科学的思维方法
  - 抽象
  - 自动化
  - 问题分解
  - 递归
  - 权衡
  - …

学习编译原理
训练计算思维
享受计算之美

What we have already introduced

- What this course is about?
- What is a compiler?
- The ways to design and implement a compiler;
- General functional components of a compiler;
- The general translating process of a compiler;

What will be introduced

- **Scanning**
  - **General information about a Scanner (scanning)**
    - **Functional requirement (input, output, main function)**
    - **Data structures**
  - **General techniques in developing a scanner**
    - **How to define the lexeme of a programming language?**
    - **How to implement a scanner with respect to the definition of lexeme?**

What will be introduced

- **Scanning**
  - **General techniques in developing a scanner**
    - **Two formal languages**
      - **Regular expression**
      - **Finite automata (DFA, NFA)**
    - **Three algorithms**
      - **From regular expression to NFA**
      - **From NFA to DFA**
      - **Minimizing DFA**
    - **One implementation**
      - **Implementing DFA to get a scanner**

Outline

2.1 Overview

- **2.1.1 General Function of a Scanner**
- **2.1.2 Some Issues about Scanning**

2.2 Finite Automata

- **2.2.1 Definition and Implementation of DFA**
- **2.2.2 Non-Determinate Finite Automata**
- **2.2.3 Transforming NFA into DFA**
- **2.2.4 Minimizing DFA**

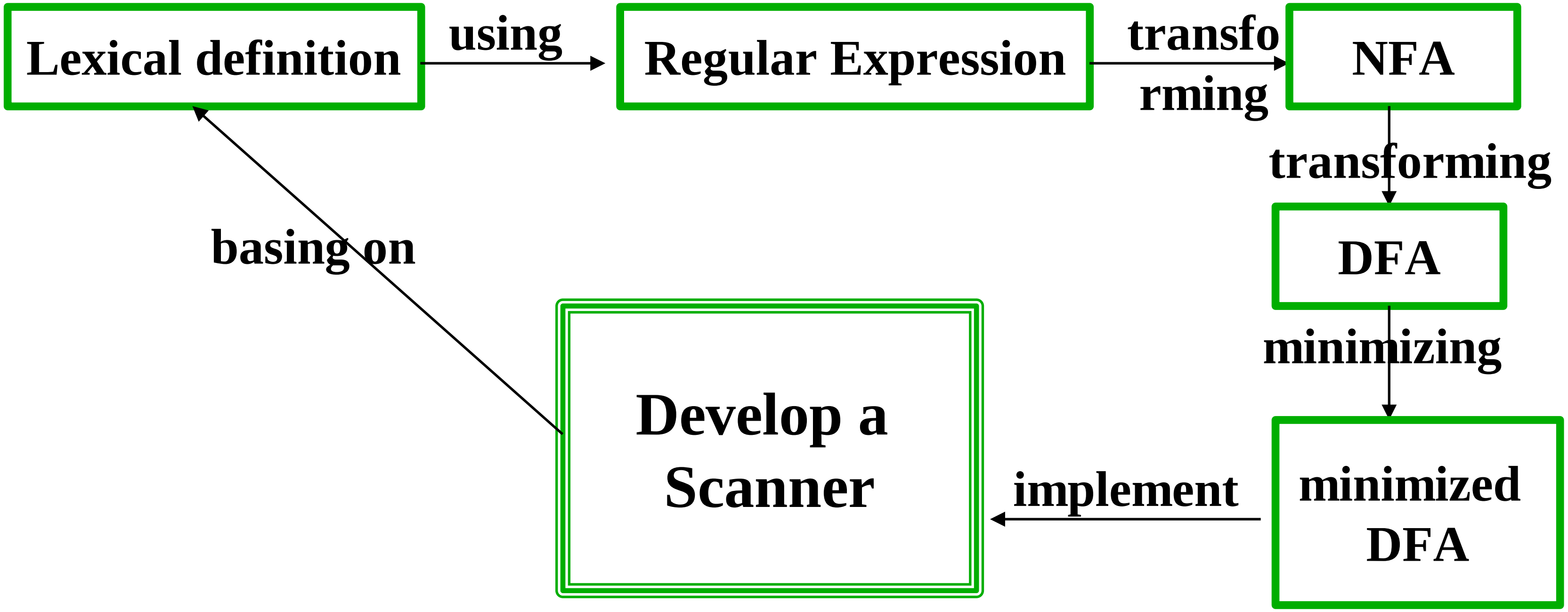
2.3 Regular Expressions

- **2.3.1 Definition of Regular Expressions**
- **2.3.2 Regular Definition**
- **2.3.4 From Regular Expression to DFA**

2.4 Design and Implementation of a Scanner

- **2.4.1 Developing a Scanner from DFA**
- **2.4.2 A Scanner Generator – Lex**

Knowledge Relation Graph



Outline

2.1 Overview

2.1.1 General Function of a Scanner

2.1.2 Some Issues about Scanning

2.2 Finite Automata

2.2.1 Definition and Implementation of DFA

2.2.2 Non-Determinate Finite Automata

2.2.3 Transforming NFA into DFA

2.2.4 Minimizing DFA

2.3 Regular Expressions

2.3.1 Definition of Regular Expressions

2.3.2 Regular Definition

2.3.4 From Regular Expression to DFA

2.4 Design and Implementation of a Scanner

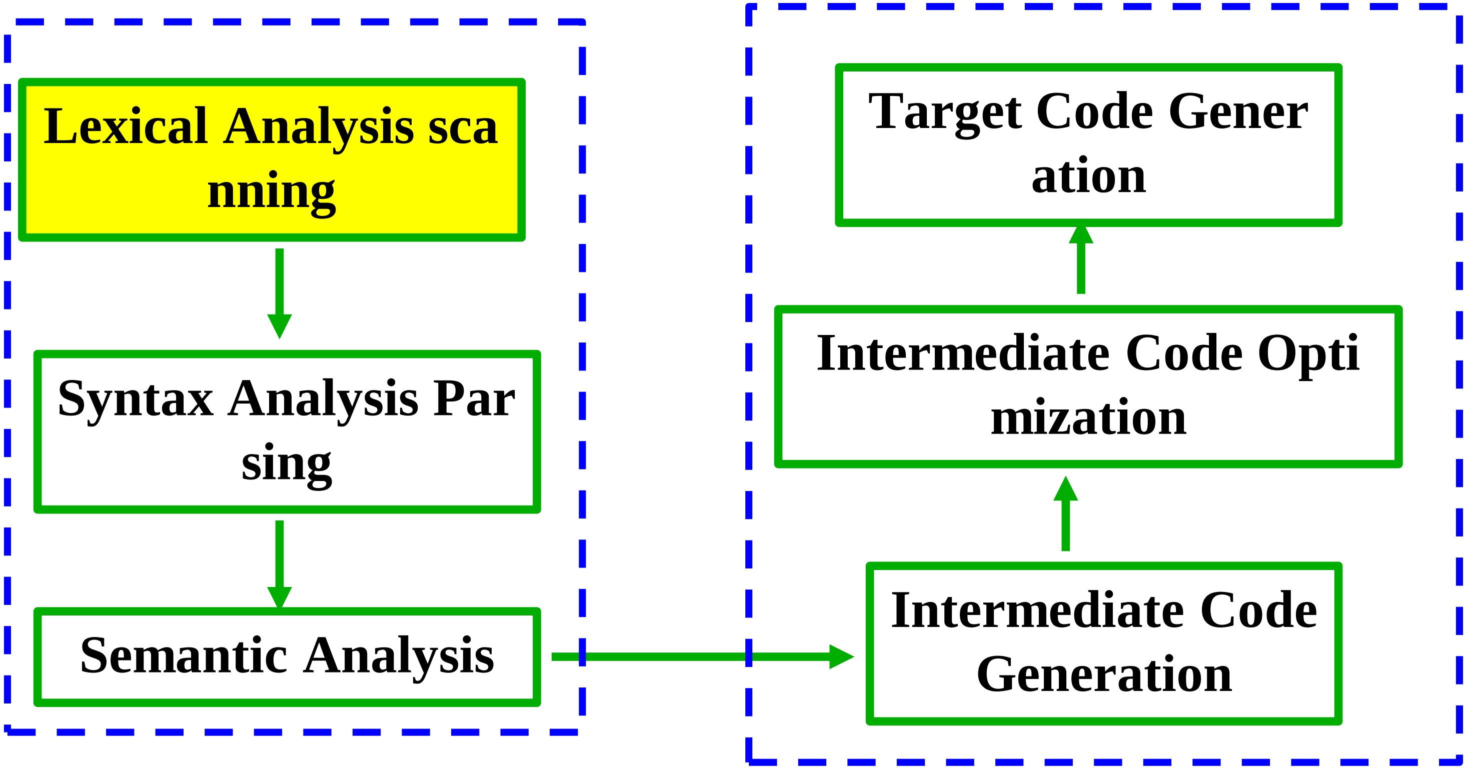
2.4.1 Developing a Scanner from DFA

2.4.2 A Scanner Generator – Lex

2.1 Overview

- **Status of Scanning in a Compiler**
- **General function of scanning**
  - **Input**
  - **Output**
  - **Functional description**
- **Some issues about scanning**
  - **Data structure（Token）**
  - **Blank/tab, return, newline, comments**
  - **Lexical errors**

Status of Scanning in a Compiler


Lexical Analysis sca nning
Syntax Analysis Par sing
Semantic Analysis
Intermediate Code Generation
Intermediate Code Opti mization
Target Code Gener ation

analysis/front end

synthesis/back end

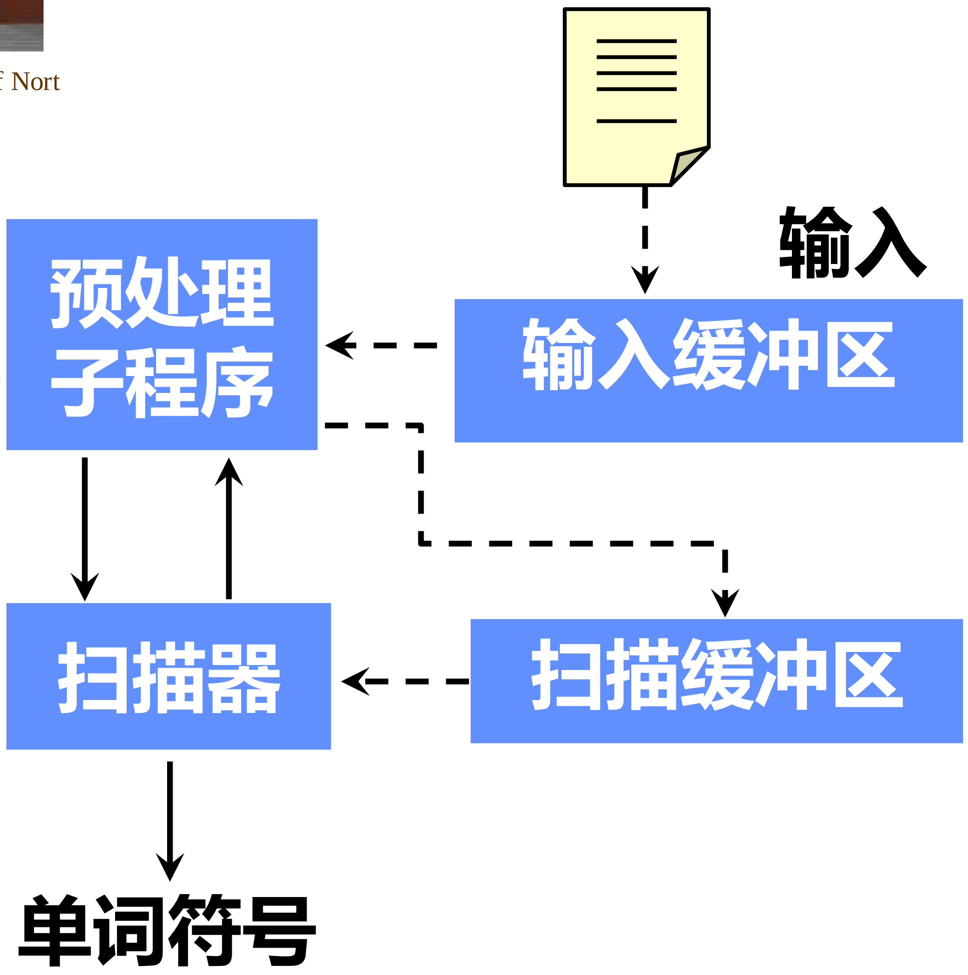
General Function of Scanning

- **Input**
  - **Source program**
- **Output**
  - **Sequence of tokens/ token**
- **Functional description (similar to spelling)**
  - ***Reads*** **source program；**
  - ***Recognizes*** **words one by one according to the lexical definition of the source language；**
  - ***Builds*** **internal representation of words – tokens;**
  - ***Checks*** **lexical errors;**
  - ***Returns*** **the sequence of tokens/token；**

词法分析器的结构

- 计算思维
  - □ 分解

- 剔除无用的空白、跳格、回车和换行等编辑性字符
- 区分标号区、续行和给出句末符等



Two Forms of a Scanner

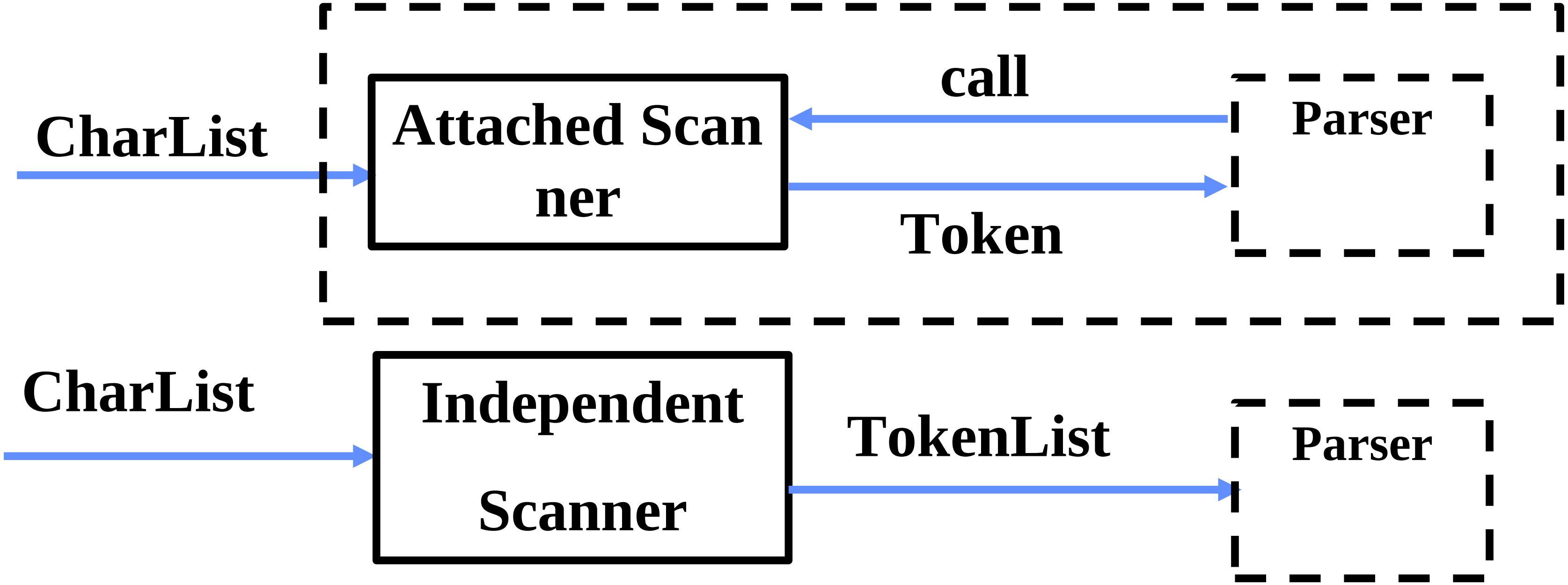
- **Independent**
  - **A scanner is independent of other parts of a compiler**
  - **Output: a sequence of tokens;**

- **Attached**
  - **As an auxiliary function to the parser;**
  - **Returns a token once called by the parser;**

Scanner

- 计算思维
  - □ 分解
  - □ 权衡

- Two forms



Some Issues about Scanning

Token

- 单词的内部表示
- Is the ***minimal semantic unit*** in a programming language;
- The content of a token

token-type	Semantic information

- A ***token*** includes two parts
  - Token Type（词性）
  - Attributes (Semantic information)（词义）
- Example:
  - < id, “ x ” >
  - < intNum, 10 >

Tokens

-Token types

- Identifier: x, y1, ……
- Number: 12, 12.3,
- Keywords (reserved words): int, real, main,
- Operator: +, -, *, / , >, < , ……
- Delimiter: ;, {, }, ……

Each can be treated as one type!
(一符一种)

-Semantic information (attributes):

- Identifier: the string
- Number: the value
- Keywords (reserved words):
  the number in the keyword table
- Operator: itself
- Delimiter: itself

单词的一般分类

1. 保留字（也称关键字，基本字等）：由语言系统自身定义，通常是由字母组成的字符串。如C语言中的main，break，switch，char等；
2. 标识符：标识程序中各个对象的名称。如变量名，常量名，过程名，数组名等；
3. 常数：表示各类常数。如整型常数，实型常数，布尔常数，字符常数等；
4. 运算符：程序中算术运算、逻辑运算、字符运算的确定的字符或字符串。如各类语言通用的＋，－，×，／，＜，＞，C语言中的++，？：，&等；
5. 界限符：如逗号，分号，括号，单引号等。

Token

- **Data Structure**
  - **Token type**

    enum
  - **One token**

    struct
  - **Sequence of tokens**

    list

Please define data structure with C !!!

Keywords

- **Keywords**
  - **Words that have special meaning**
  - **Can not be used for other meanings**
- **Reserved words**
  - **Words that are reserved by a programming language for special meaning;**
  - **Can be used for other meaning, overload previous meaning;**
- **Keyword table**
  - **to record all keywords defined by the source programming language**

Sample Source Program

v	a	r	↵	i	n	t	△	x	,	y	;	↵	{	↵
r	e	a	d	(	x	)	;	↵	r	e	a	d	(	y
)	;	↵	i	f	△	x	>	y	△	t	h	e	n	△
w	r	i	t	e	(	1	)	△	e	l	s	e	△	w
r	i	t	e	(	0	)	;	↵	}					

Sequence of Tokens

var， k	↵	int, k	x, ide	,	y, ide	;	↵
{	read, k	(	x, ide	)	;	↵	read, k
(	y, ide	)	;	↵	if, k	x, ide	>
y , ide	then, k	write,k	(	1, num	)	;	↵
else, k	write,k	(	0, num	)	;	↵	}

Blank, tab, newline, comments

- No semantic meaning
- Only for readability
- Can be removed
- Line number should be calculated;

The End of Scanning

- Two optional situations
  - Once read the character representing the end of a program;
    - PASCAL: ‘.’
  - The end of the source program file

Lexical Errors

- **Limited types of errors can be found during scanning**
  - **illegal character ;**
    - § , ←
  - **the first character is wrong;**
    - **"/abc"**
- **Lexical Error Recovery**
  - **Once a lexical error has been found, scanner will not stop, it will take some measures to continue the process of scanning**
    - **Ignore current character, start from next character**

if § a then x = 12.else ……

To Develop a Scanner

- Now we know ***what*** is the function of a Scanner；
- How to implement a scanner?
  - Basis : lexical rules of the source programming language
    - Set of allowed characters;
    - What kinds of tokens it has?
    - The structure of each token-type
  - Scanner will be developed according to the ***lexical rules***;

How to define the lexical structure of a programming language?

1. **Natural language (English, Chinese ……)**

   - easy to write, ambiguous and hard to implement

2. **Formal languages**

   - need some special background knowledge;

   - concise, easy to implement;

   - automation;

- **Two formal languages for defining lexical structure of a programming language**
  - ***<u>Finite automata</u>***
    - **Non-deterministic finite automata (NFA)**
    - **Deterministic finite automata (DFA)**
  - ***<u>Regular expressions</u>*** **(RE)**
  - **Both of them can formally describing lexical structure**
    - **The set of allowed words**
  - **They are equivalent;**
  - **FA is easy to implement; RE is easy to define;**

Summary

- What is the main function of scanning (a scanner)?
- Two forms of Scanner?
- What is a token? The content of a token?
- Semantic information for different token types?
- Lexical error?
- Two key problems in design and implementation of a scanner?

Reading Assignment (2)

- **Topic: Different methods for dealing with lexical errors?**
- **Objectives**
  - **Try to find out different ways to cope with lexical errors;**
- **Tips**
  - **There are different schema (error correction; error repair; error recovery;……)**
  - **Google some research papers;**
  - **Treat it as a survey;**
  - **challenging!**