Grit

能很投入地一直做一件事很久

- **What**：古英语中的原义是沙砾，即沙堆中坚硬耐磨的颗粒。**Grit**可译为“坚毅”，但其涵义远比毅力、勤勉、坚强都要丰富得多。**Grit**是对长期目标的持续激情及持久耐力，是不忘初衷、专注投入、坚持不懈，是一种包涵了自我激励、自我约束和自我调整的性格特征。
- **Why**：智商是与生俱来的，而坚毅是每个人都可以开发的。

- How：

1. 超越知识的学习（不能“死读书读死书”）

2. 原创性思维的培养（不能“人云亦云”）

3. 价值取向的重塑（不能“功利主义”）

4. 核心竞争力的建立（不能“身无绝技”）

5. 解决问题的模式（不能“片面肤浅短视”）

Information Science and Technology College of
Northeast Normal University

Compiling and Running of Program

Dr. Zheng Xiaojuan
Professor

Sep. 2019

Question

- **How much you know about a compiler?**
- **Which compilers have you used before?**
- **What kind of compiling errors have you find in your programs before?**
  - **Undefined identifier;**
  - **Missing ……;**

Outline

1. Introduction to Compiler

- 1.1 Programming Languages
- 1.2 Compiler and Interpreter
- 1.3 Programs related to Compiler
- 1.4 Design and Implementation of a Compiler
- 1.5 Functional Decomposition and Architecture of a Compiler
- 1.6 General Working Process of a Compiler for a C0 Language

Objectives

- **To know**
  - **Different programming languages and their features**
  - **Different ways to implement a programming language**
  - **The process to handling programming languages**
  - **The functional components of a compiler**
  - **The working process of a general simple compiler with an example**

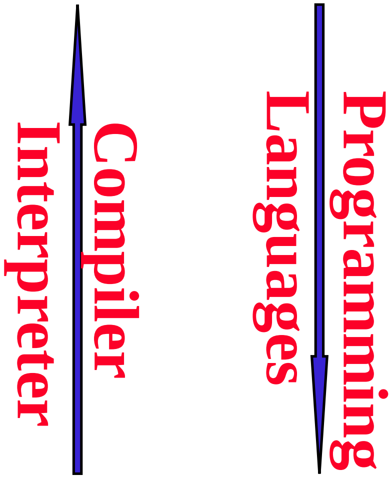
1.1 Programming Languages

1.1 Programming Languages

- **History**
  - **1800，First Programmer**
    （Jacquard loom; Analytical engine; Ada Augusta）
  - **1950，First Programming Language**
    (FORTRAN; COBOL; Algol60; LISP)
  - **1960，emerging hundreds of programming languages**
    (special-purpose languages; universal language)
  - **1970，simplifying, abstraction (PASCAL; C;)**
  - **1980， Object Oriented (Ada; Modular; Smalltalk; C++)**
  - **1990，Internet (Java)，Libraries，Script**
    (Scripting; Perl; Javascript)
  - **2000，new specification language**

1.1 Programming Languages

- **Classifications**
  - **Functions**
    - **Scientific computation; business; table handling; forms; strings; multi-functional;**
  - **Abstraction level**
    - **Low level**
      - **Machine language; assembly language**
    - **High Level (different paradigms范例)**
      - **Procedural programming languages: FORTRAN, PASCAL, C**
      - **Object-oriented programming languages: Smalltalk, Java, C++**
      - **Functional programming languages: LISP, HASKELL, ML**
      - **Logical programming languages: PROLOG**

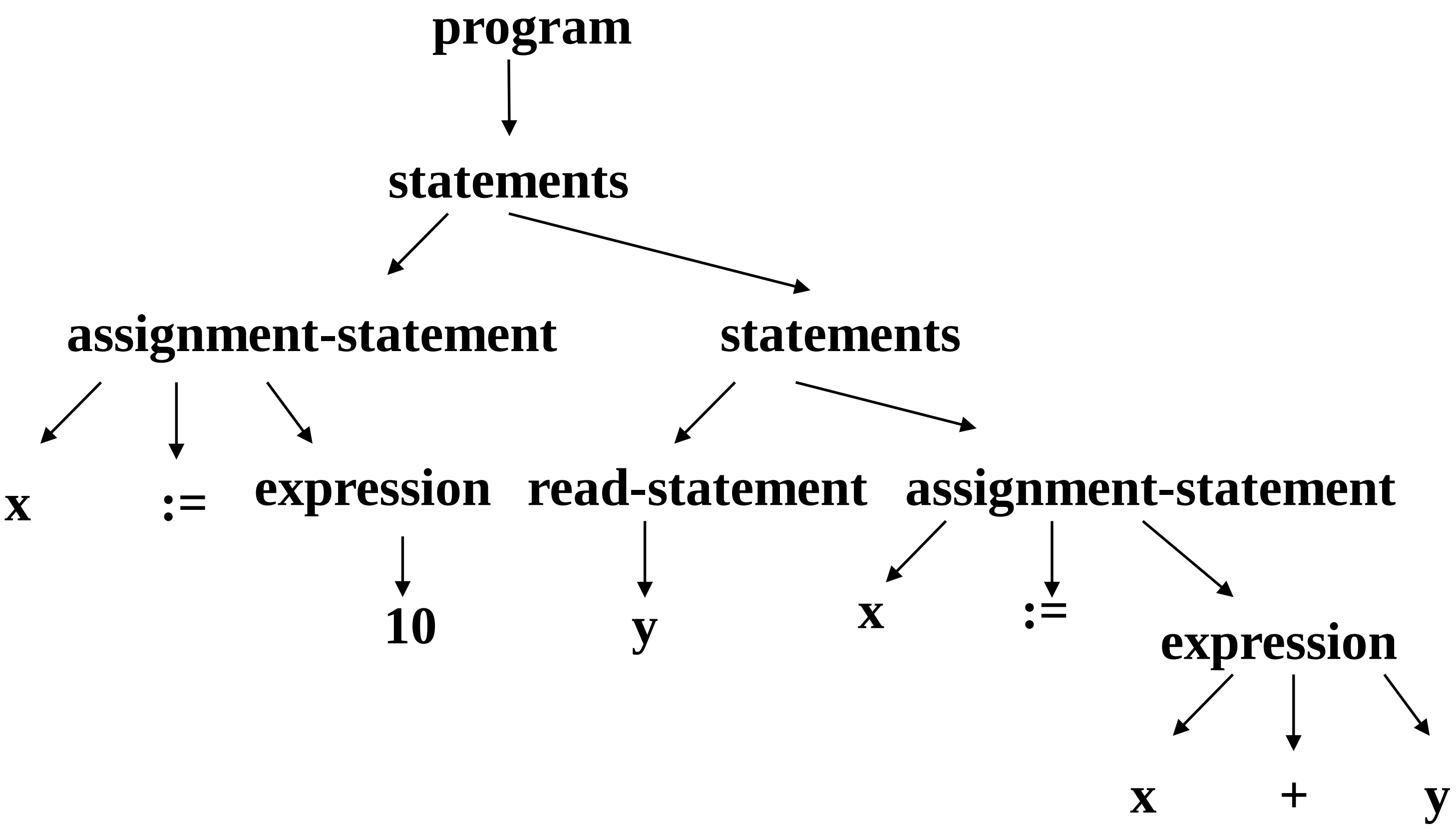


1.1 Programming Languages

- Definition of a programming language includes
  - Lexeme
    - Allowed set of characters
    - Lexical structure
  - Syntax
    - Program structure
  - Semantics
    - Meaning of different structures

```
{
      x:= 10;
      read(y);
      x := x + y
}
```

{	↵	Iden,”x”	:=	10	;	↵
read	(	Iden,”y”	)	;	↵	Iden,”x”
:=	Iden,”x”	+	Iden,”y”	;	↵	}



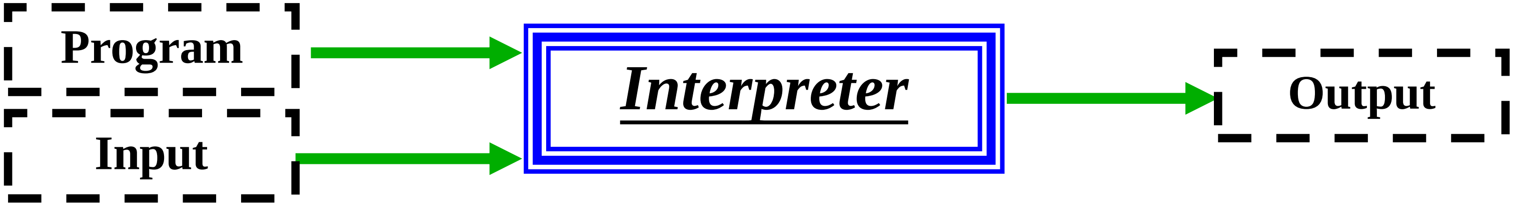
program
statements
assignment-statement
statements
x
:=
expression
read-statement
assignment-statement
10
y
x
:=
expression
x
+
y

x	**varKind**	**int**	**……**
y	**varKind**	**int**	**……**

1.2 Compiler and Interpreter

1.2 Compiler and Interpreter

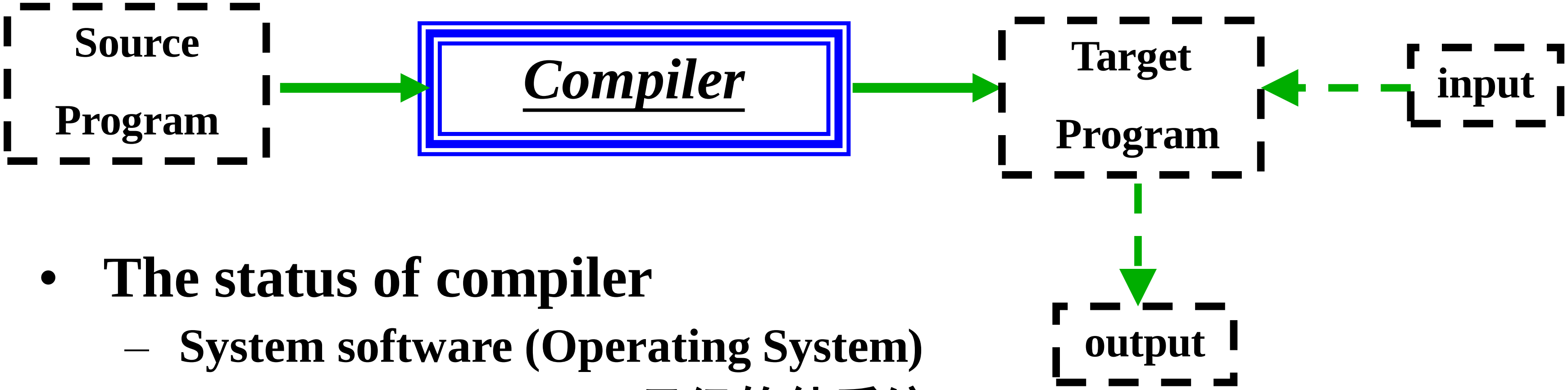
- **Implementation of Programming Languages**
  - ***Interpreter*** **:**



  - ***Translator*** **: Language1 → Language2**
    - ***Assembler*****: Assembly Languages → Machine Code**
    - ***Compiler*** **: High-level Languages → Low-level Languages**

1.2 Compiler and Interpreter

- **Compiler: a program that reads a program written in one language (source language) and translate it into an equivalent program in another language (target language).**



- **The status of compiler**
  - **System software (Operating System)**
  - **Meta software system(元级软件系统)**

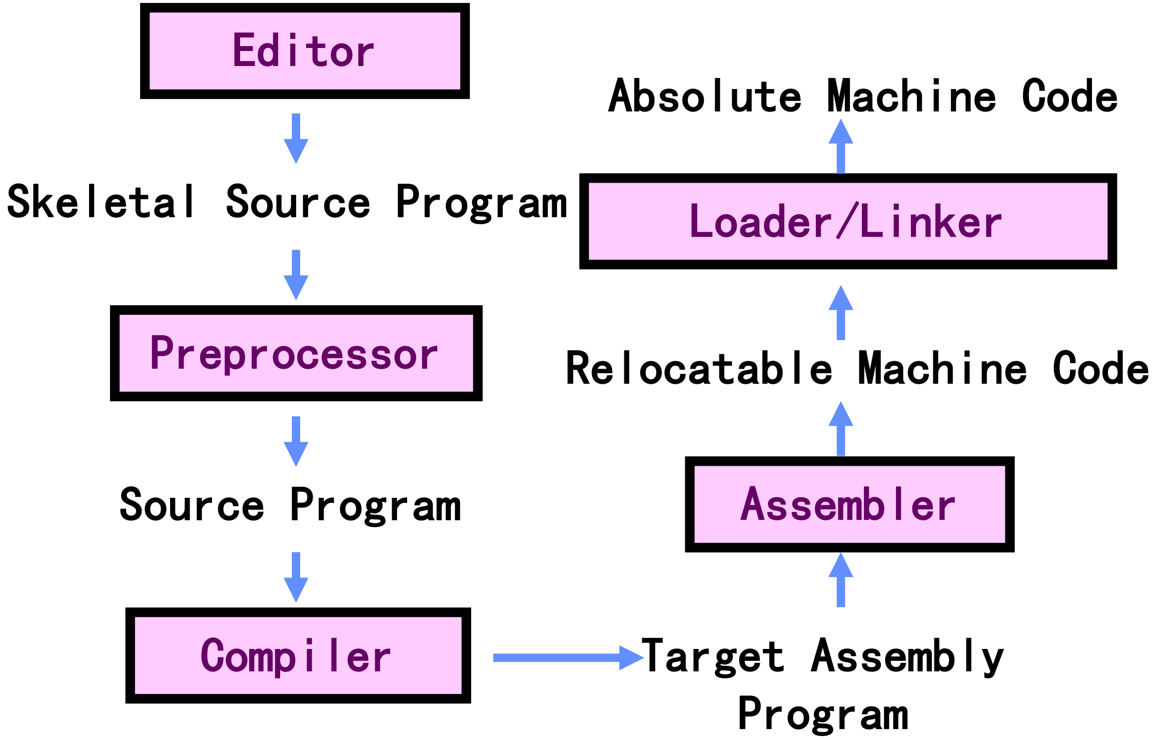
1.2 Compiler and Interpreter

- **Comparing Compiler with Interpreter**
  - **Similarity**
    - **Using same implementation techniques**
  - **Difference**
    - **Mechanism: Translation vs. Interpretation**
    - **Execution efficiency: high vs. low**
    - **Storage cost: less vs. more**
- **Interpreter has some advantages over Compiler**
  - **Portability: Java**
  - **General**
  - **Intermediate code generation is not necessary**

1.3 Programs related to Compiler

1.3 Programs Related to Compiler

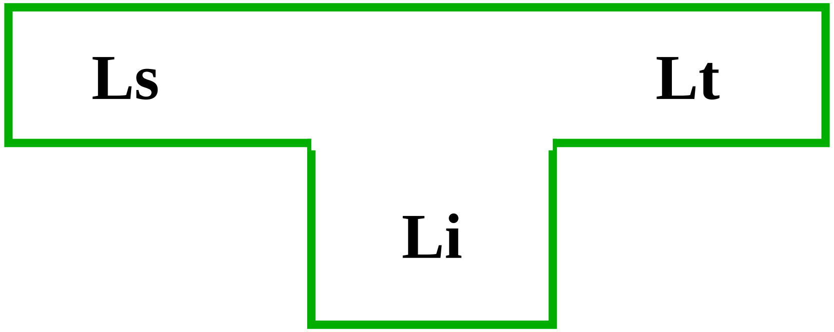
- Editor
- Preprocessor
- Compiler
- Assembler
- Loader
- Linker



1.4 Design and Implementation of Compiler

1.4 Design and Implementation of a Compiler

- **There are 3 languages associated with a compiler**
  - **Source language Ls: Input**
  - **Target language Lt: Output**
  - **Implementation language Li: the language for developing the compiler**
- **A *compiler* is a program written in Li, whose function is translating a program written in Ls into equivalent program in Lt.**



1.4 Design and Implementation of a Compiler

- **For different programming language paradigms, different techniques will be applied for developing their compilers;**
- **In this course, focus on general compiler construction principles and techniques on *procedural programming languages*;**

1.4 Design and Implementation of a Compiler

- **There are no existing compilers**
  - **Manually programming machine code (手工编写机器代码)**
    - **Inefficient, hard to maintain**
  - **Self extending (自展法)**
- **There are compilers available**
  - **Preprocessing (预处理方法)**
  - **Porting (移植法)**
  - **Tools (工具法)**
    - **Automatic generator (自动生成工具)**
  - **Writing codes**

1.4 Design and Implementation of a Compiler

- **Self extending**
  - ***Problem***: **if there is no any compiler available, we want to develop a compiler for a programming language *L*;**
  - ***Solution***:
    - **Define L0 as a sub-language of L;**
    - **Manually write a compiler for L0;**
    - **Make some extensions to L0, which is called L1,**
    - **Develop L1's compiler with L0;**
    - **……**
    - **Develop Ln(=L)'s compiler with Ln-1;**

1.4 Design and Implementation of a Compiler

- **Preprocessing**
  - **_Problem_: if we have a programming Language *L* and its compiler, we want to develop a compiler for a programming language *L1* which makes some *extensions* to L;**
  - **_Solution_:**
    - **Develop a preprocessor: Translating L1 into L**
    - **Use L's compiler: from L to Target code**
  - **For example: C++ → C**

1.4 Design and Implementation of a Compiler

- **Porting**
  - ***<u>Problems</u>*:**
    - **source language L**
    - **L’s compiler for machine M1**
    - **we want to develop another compiler of L for machine M2;**
  - **Same source language, Different target languages**
  - **Two ways**
    - **Develop a program for translating from machine code for M1 to machine code for M2;**
    - **Rebuild the *<u>back-end</u>* of the compiler**

1.5 **Functional Decomposition & Architecture of a Compiler**

1.5 Functional Decomposition & Architecture of a Compiler

- **Programming Problem**
  - **Develop a compiler for a programming language**

How?

- **Need to make clear**
  - **What we already know?**
  - **What we are going to do?**
    - **Input & Output**
    - **Data structure + algorithm**

1.5 Functional Decomposition & Architecture of a Compiler

- **What we already know?**
  - **Definition of the source language (notation, structure, semantics, rules)**
  - **Definition of the target language (notation, structure, semantics, rules)**
  - **The language that we are going to use to develop the compiler**

1.5 Functional Decomposition & Architecture of a Compiler

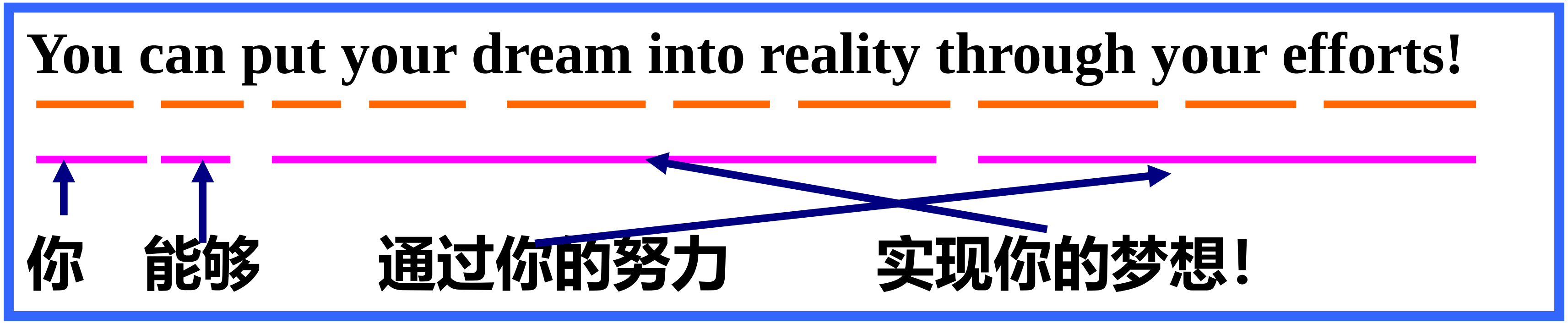
- Functional description of a compiler
  - Input: programs written in source language (source programs)

    = sequence of characters
  - Output: programs written in target language (target programs/code)

    = sequence of instructions
  - Algorithm?
    - A general process of translating each source program into corresponding target program;

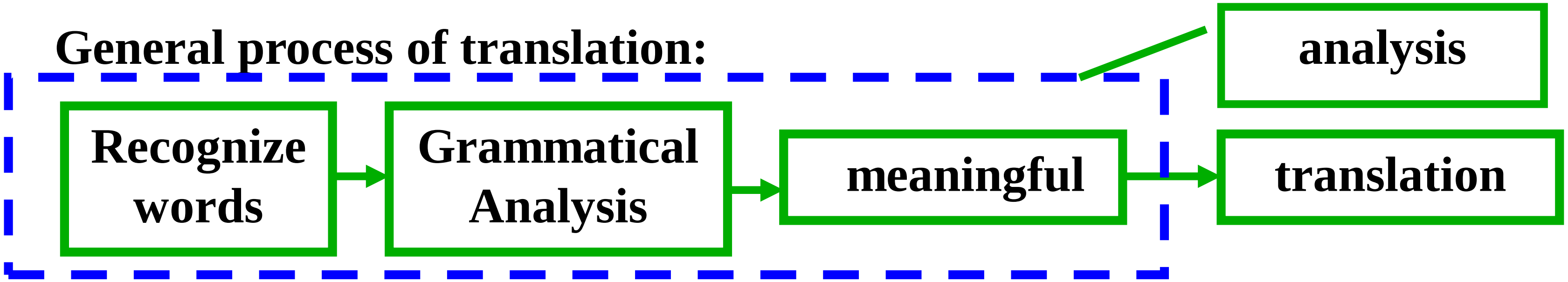
1.5 Functional Decomposition & Architecture of a Compiler

- Think about “natural language translation”
  - From English to Chinese



You can put your dream into reality through your efforts!

你 能够 通过你的努力 实现你的梦想！



General process of translation:

Recognize words → Grammatical Analysis → meaningful → translation

analysis

1.5 Functional Decomposition & Architecture of a Compiler

- **To summarize**
  - **Grasp source language and target language**
    - **Words, syntax, the meaning**
  - **The process of translation one sentence includes**
    - **Analyzing the sentence to make sure that it is correct**
      - **Spell, including recognizing words and their attributes**
      - **Build syntactic structure with respect to the grammar of source language;**
      - **Make sure it is meaningful;**

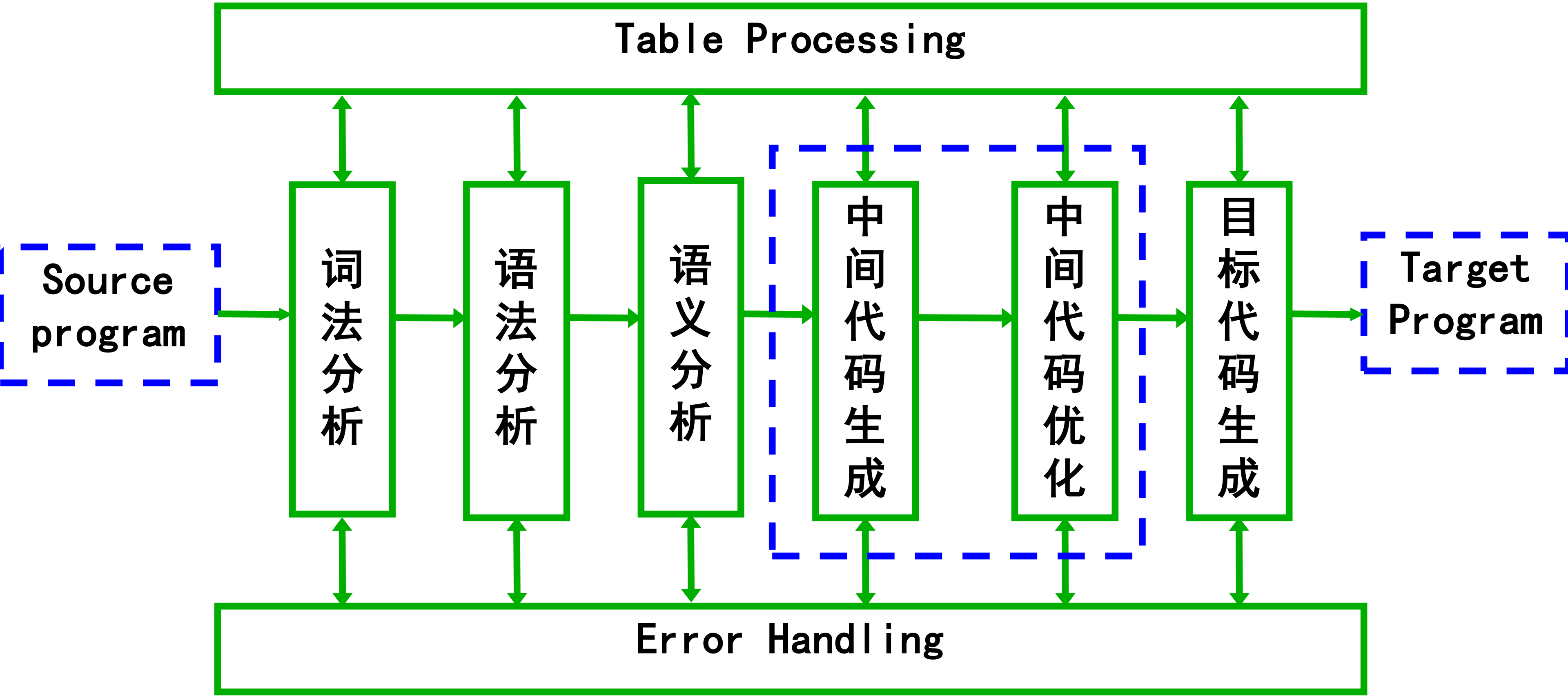
I eat sky in dog.

-
  -
    - **Translating the sentence into target language**
      - **Translating each syntactic parts**
      - **Composing them into a meaningful sentence in target language**

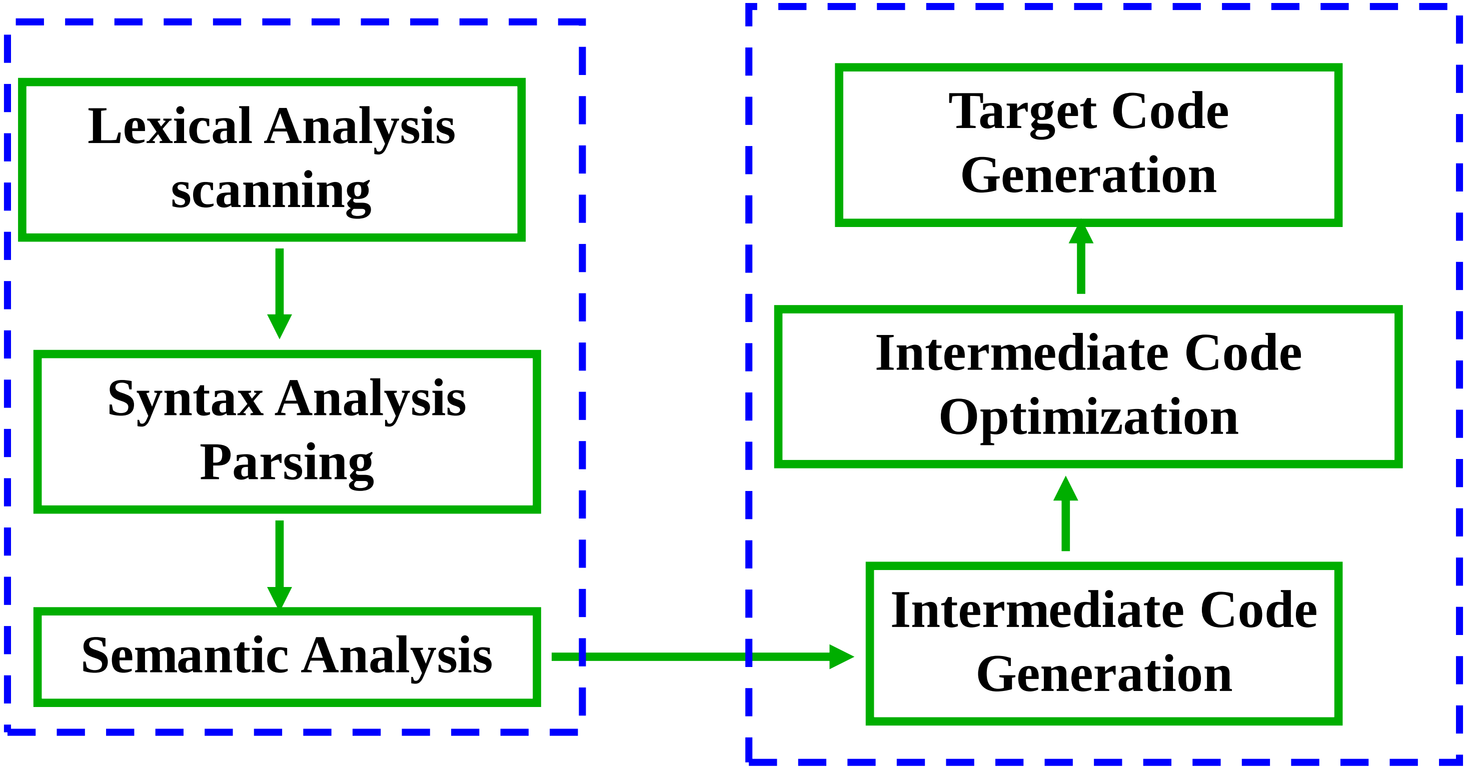
1.5 Functional Decomposition & Architecture of a Compiler

What about translating one programming language into anothter programming language?

1.5.1 Functional Decomposition & Architecture of a Compiler



1.5 Functional Decomposition & Architecture of a Compiler



analysis/front end

synthesis/back end

1.5 Functional Decomposition & Architecture of a Compiler

- **Lexical Analysis**
  - ***Reading*** **the source program, which is actually in the form of a stream of characters;**
  - ***Collects*** **sequences of characters into meaningful unit, called *tokens*s;**
- **Syntax Analysis**
  - **Reading the sequences of tokens;**
  - **Determining the syntactical structure of the program;**
  - **The results of parsing are represented as a parse tree or a syntax tree;**
- **Semantic Analysis**
  - **Static semantics checking, such as type checking**
  - **Symbol table (attributes of identifiers)**

1.5 Functional Decomposition & Architecture of a Compiler

- Code Generation
  - Intermediate Code generation
    - Intermediate representation
    - portability
  - Target Code generation
- Code Optimization
  - Efficiency of target program
    - Intermediate code optimization
    - Target code optimization

1.6 A General Working Process of a Compiler

1.6 A General Working Process of a Compiler

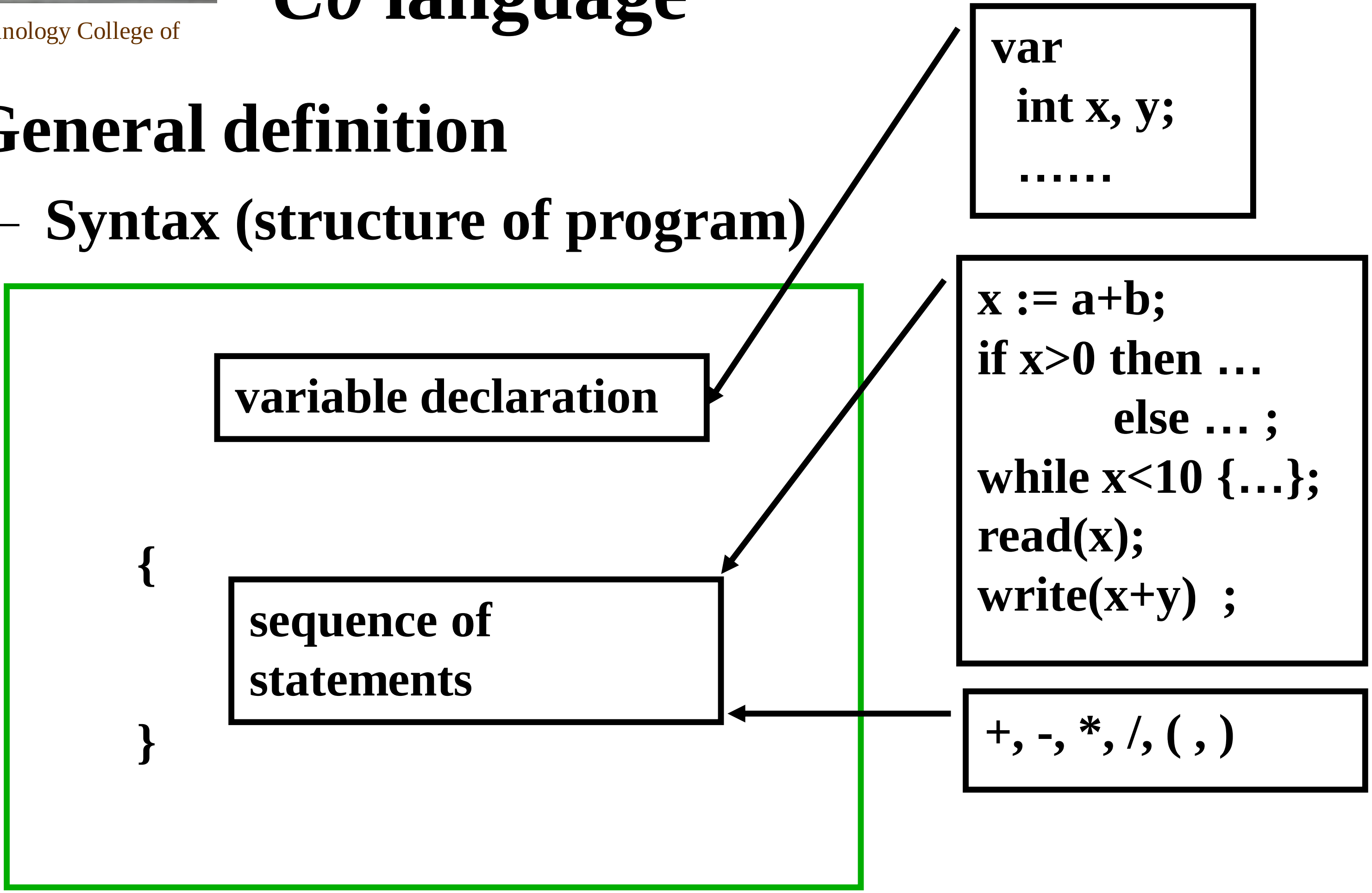
- **Source language : a toy programming language *<u>C0</u>***
- **Target language : assembly language *AL***
- **Demonstrate with an example on how a compiler is translating a program in *<u>C0</u>* into assembly codes;**

C0 language

- **General definition**
  - **Lexical structure**
    - **Allowed set of characters {a-z, A-Z,0-9}**
    - **Tokens:**
      - **keywords : {, read, write, }**
      - **Identifiers: sequence of limited number of characters starting with letters**
      - **Numbers: integer**
      - **Operators: +, *,:=**
      - **Delimiters: ; , (,)**

C0 language

- **General definition**
  - **Syntax (structure of program)**



variable declaration

{

sequence of statements

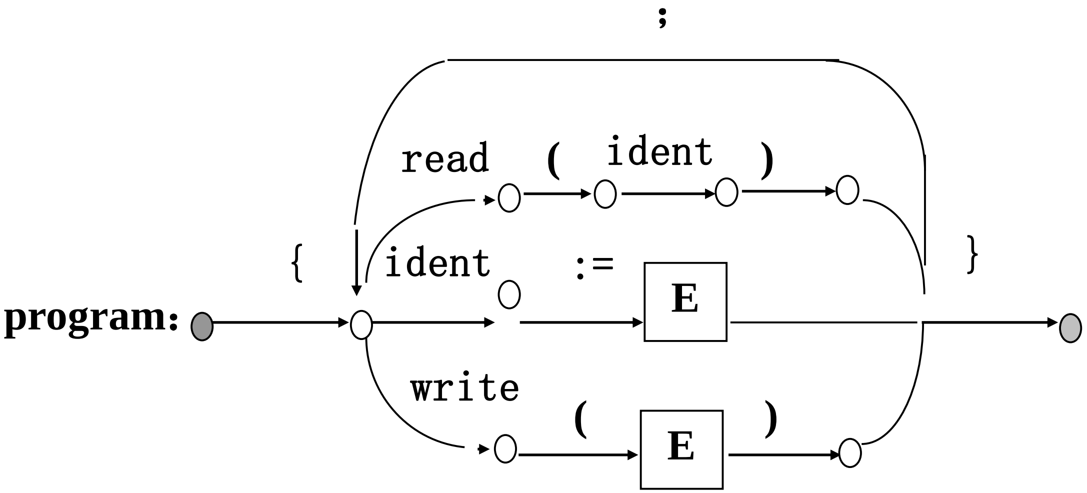
}

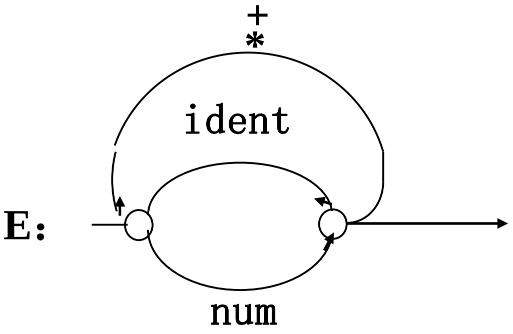
```
var
  int x, y;
  ……
```

```
x := a+b;
if x>0 then …
        else … ;
while x<10 {…};
read(x);
write(x+y)  ;
```

+, -, *, /, (,)

The structure of program of *C0*





C0 language

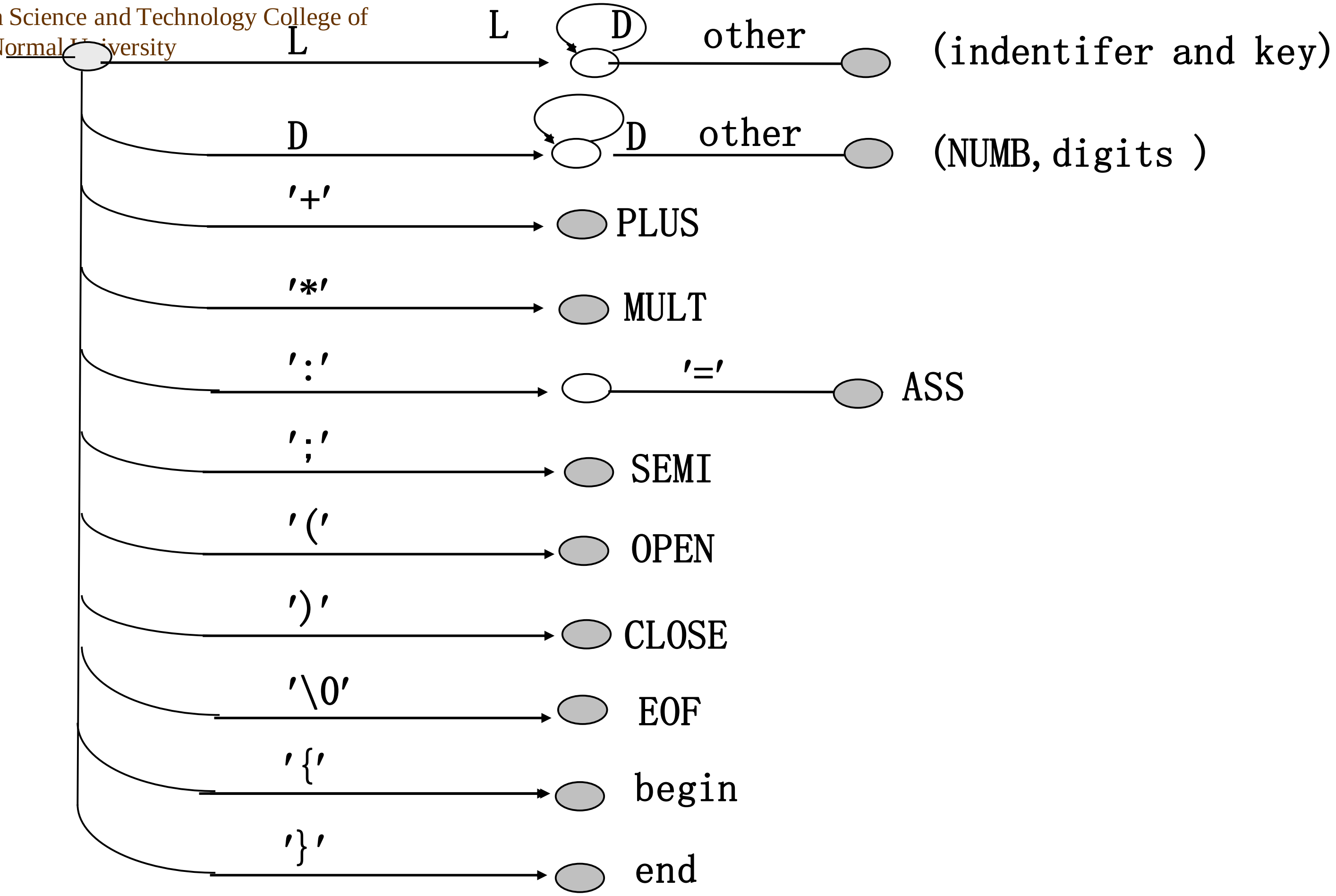
- **General definition**
  - **Semantics**
    - **Static semantics**
      - **One identifier should not be declared more than once;**
      - **Use identifiers after they are declared;**
      - **Type equivalence in assignment and expressions;**
      - **The result type of conditional expression in if or while statement should be boolean;**
    - **Dynamic semantics**

- **Sample program**

```
{
    x:= 10;
    read(y);
    x := x + y
}
```

{	↵	x	:	=	1	0	;	↵	r	e	a	d	(	y
)	;	↵	x	:	=	x	+	y	↵	}				

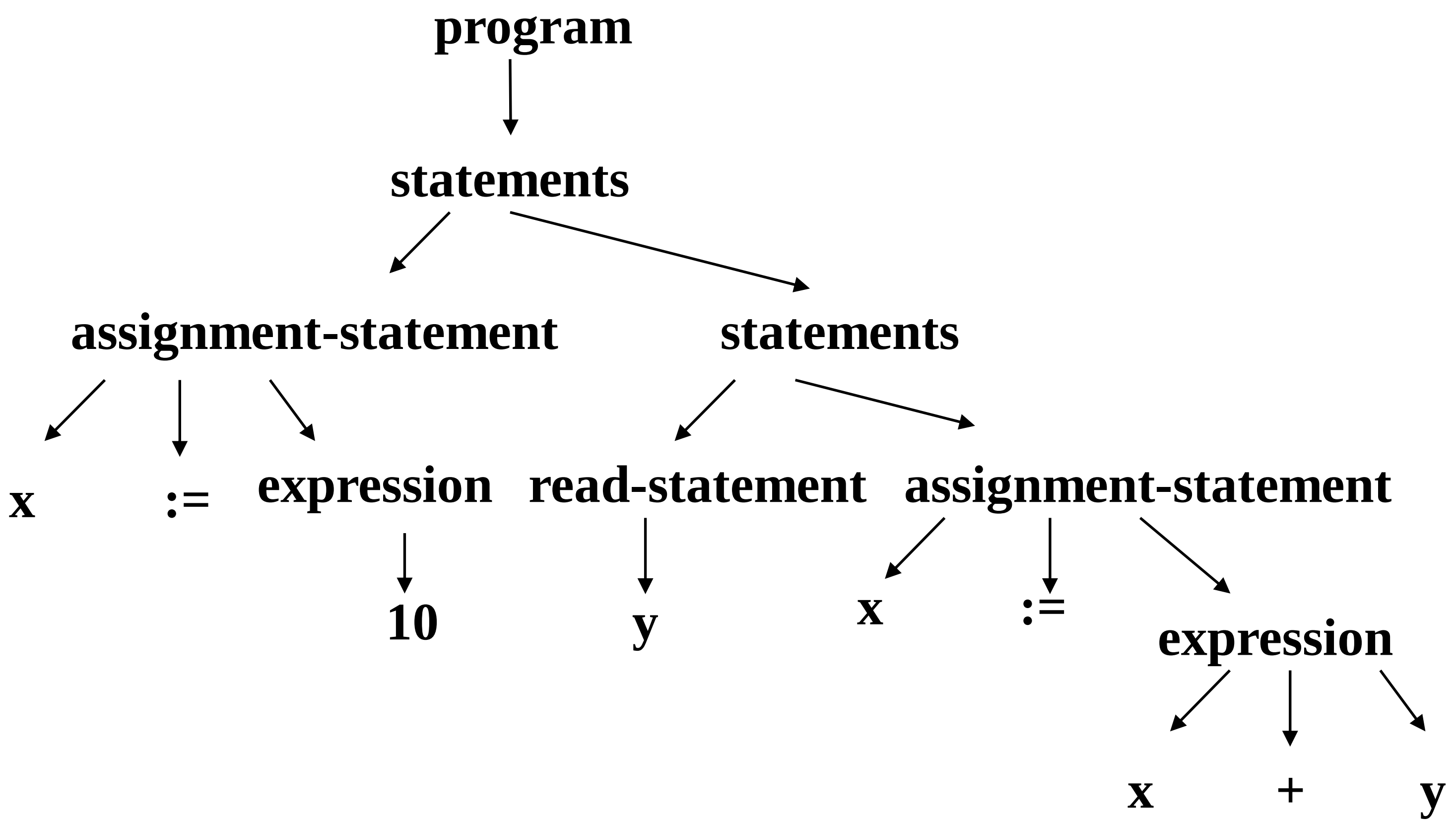
Lexical analysis



Lexical analysis

{	↵	Iden,”x”	:=	10	;	↵
read	(	Iden,”y”	)	;	↵	Iden,”x”
:=	Iden,”x”	+	Iden,”y”	;	↵	}

Syntax Analysis



Semantic Analysis

x	varKind	int	……
y	varKind	int	……

Code Generation

	STOR 10 <x>
	INP(y)
	PLUS <x> <y> <x>

Summary

- **Different classification of programming languages**
- **Definition of a programming language**
- **Definitions and differences of compiler and interpreter;**
- **Programs related to processing programming languages;**
- **Design and implementation of a compiler;**
- **Functional components of a compiler;**
- **General working process of a compiler;**

Summary

- The problem that we are going to solve in this course
  - How to develop a compiler for a programming language?
    - Source language: a high-level programming language
    - Target language: assembly language or machine language
    - Develop a program, whose function is to translating a program written in source language
- Principle
  - Divide and conquer (分而治之)
  - Problem → programming task → solution → general principles and methods

Any Questions?

Reading Assignment

- **Topic: How to develop a scanner(词法分析器)？**
- **Objectives**
  - **Get to know**
    - **What is a scanner? (input, output, functions)**
    - **Lexical rules for C & Java?**
    - **Originally how you want to develop a scanner?**
    - **From textbook, how a scanner can be built?**
- **References**
  - **Optional textbooks**
- **Tips:**
  - **Collect more information from textbooks and internet;**
  - **Establish your own opinion;**