UNIVERSITY OF NICOSIA
ΠΑΝΕΠΙΣΤΗΜΙΟ ΛΕΥΚΩΣΙΑΣ

COMP-421 Compiler Design

Presented by
Dr Ioanna Dionysiou

Administrative

- Next time reading assignment
  - [ALSU07] Chapters 1,2
    - [ALSU07] Sections 1.1 - 1.5 (cover in class)
    - [ALSU07] Section 1.6 (read on your own)
      - Programming language basics that you should be familiar
      - Email me questions (if any) before next class
    - [ALSU07] We will cover selected parts from all sections

Copyright (c) 2012 Ioanna Dionysiou

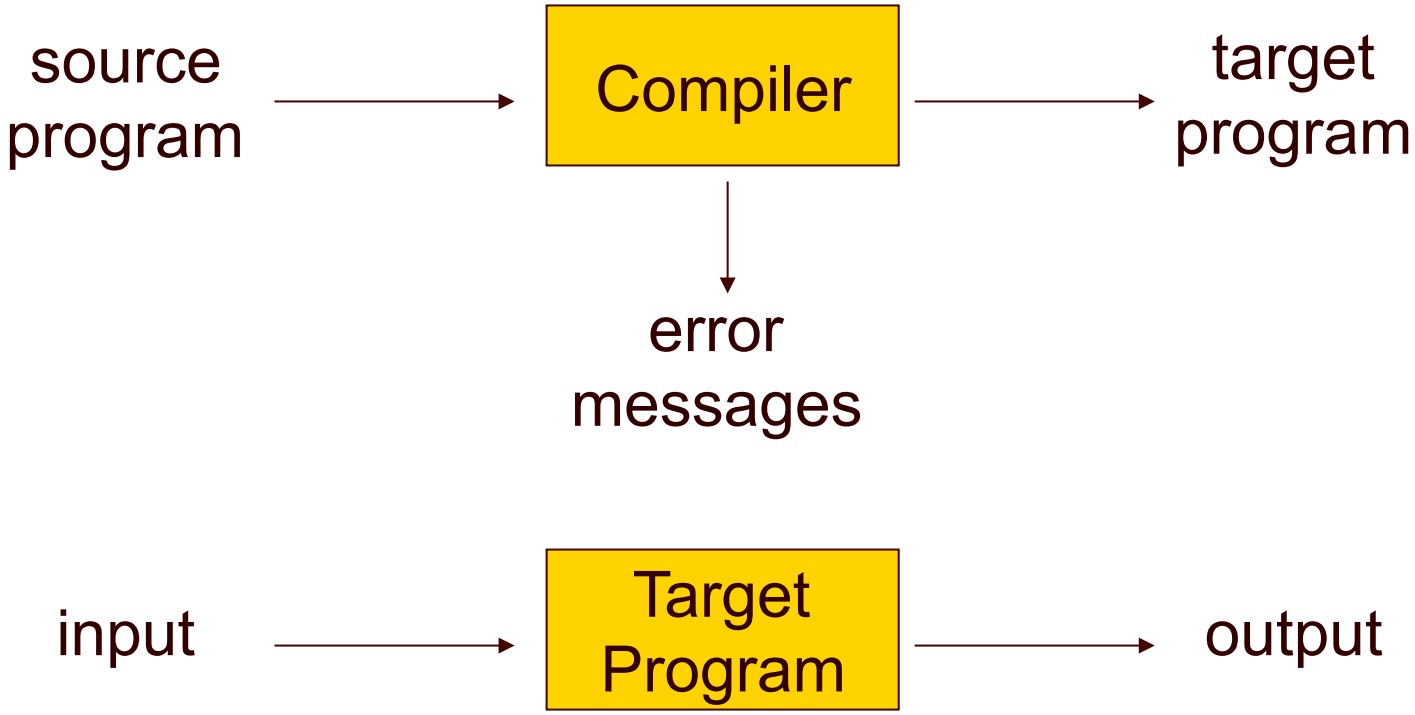
Lecture Outline

- Chapter 1 - Introduction
  - Definition
  - Structure of Compiler
    - Compilation parts
      - Analysis, Synthesis
    - Phases of a compiler
  - An Example: Translation of a statement
  - Evolution of Programming Languages
  - Building a Compiler and Applications of Compiler Technology

Copyright (c) 2012 Ioanna Dionysiou

Compiler: Simple Definition

- A compiler is a program that reads a program written in one language and translates it into an equivalent program in another language



Copyright (c) 2012 Ioanna Dionysiou

The context of a compiler

- In addition to a compiler, several other programs may be required to create an executable target program
  - Preprocessor
  - Assembler
  - Linker/Loader

These are the cousins of the compilers!

Copyright (c) 2012 Ioanna Dionysiou

Cousins of the compiler

- Preprocessors
  - Macro processing
  - File inclusion
- Assemblers
  - Some compilers produce assembly code that is passed to an assembler for further processing
  - The assembler then produces machine code
- Linker/loader
  - Applies to large programs in multiple files
  - Linker resolves the address (location) of variables
  - Loader combines all the pieces into an executable

Copyright (c) 2012 Ioanna Dionysiou

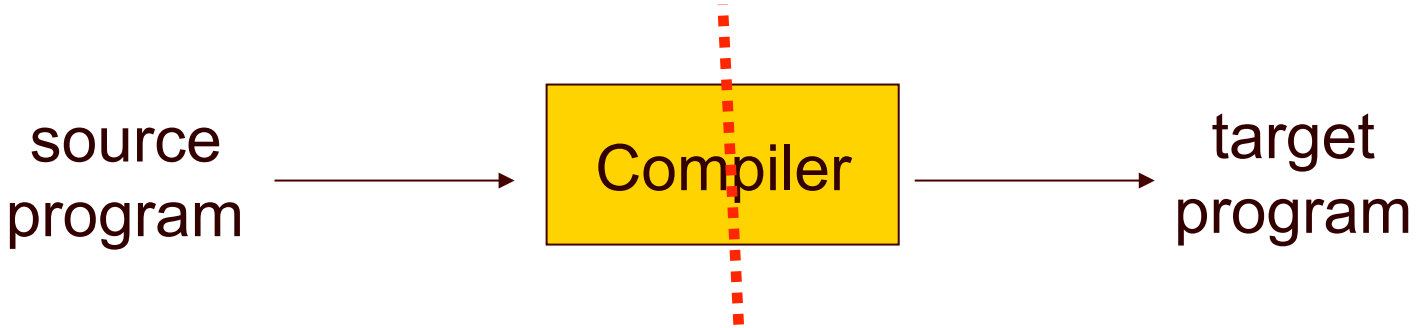
Lecture Outline

- Chapter 1 - Introduction
  - Definition
  - Structure of Compiler
    - Compilation parts
      - Analysis, Synthesis
    - Phases of a compiler
  - An Example: Translation of a statement
  - Evolution of Programming Languages
  - Building a Compiler and Applications of Compiler Technology

Copyright (c) 2012 Ioanna Dionysiou

Compilation Process

- There are two parts to compilation
  - Analysis
    - Break up the source program into pieces
    - Create an intermediate representation of the source program
  - Synthesis
    - Construct the desired target program from the intermediate representation



Copyright (c) 2012 Ioanna Dionysiou

Analysis of Source Program

- It consists of three phases (front-end)
  - Lexical analysis
    - Linear analysis, scanning
    - Stream of characters are read and grouped into tokens (sequence of characters with a collective meaning)
  - Syntax analysis
    - Hierarchical analysis, parsing
    - Tokens are grouped hierarchically into nested collections with collective meaning
  - Semantic analysis
    - Check to ensure that the components of a program have a meaning

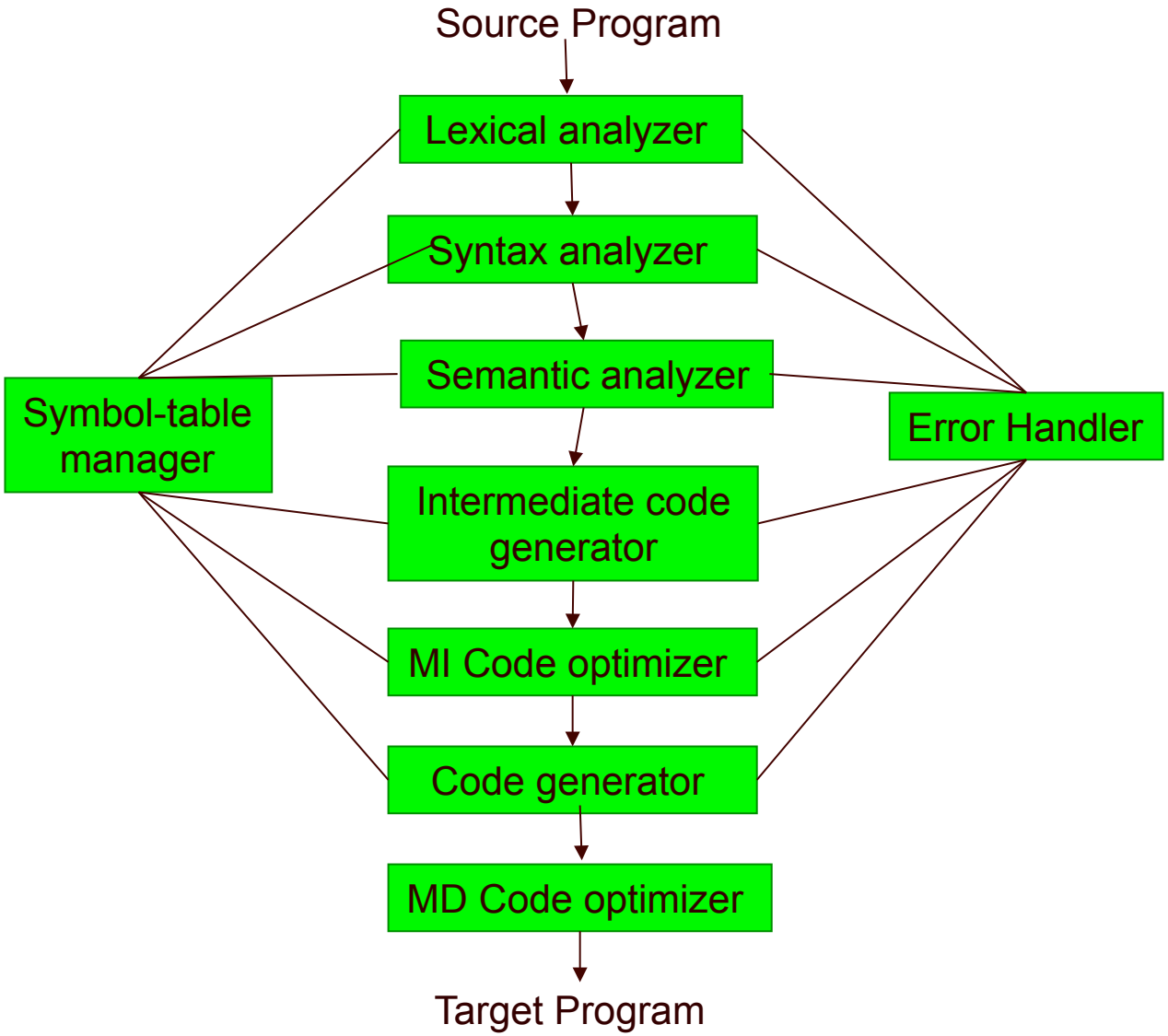
Copyright (c) 2012 Ioanna Dionysiou

Synthesis of Target Program

- It consists of four phases (back-end)
  - Intermediate Code Generator
    - Explicit low-level or machine-like intermediate representation
  - Machine-Independent Code Optimizer
    - Improve the intermediate code (faster, shorter code, etc)
  - Code Generator
    - Translate intermediate instructions into sequence of machine instructions
  - Machine-Dependent Code Optimizer
    - Improve the generated code

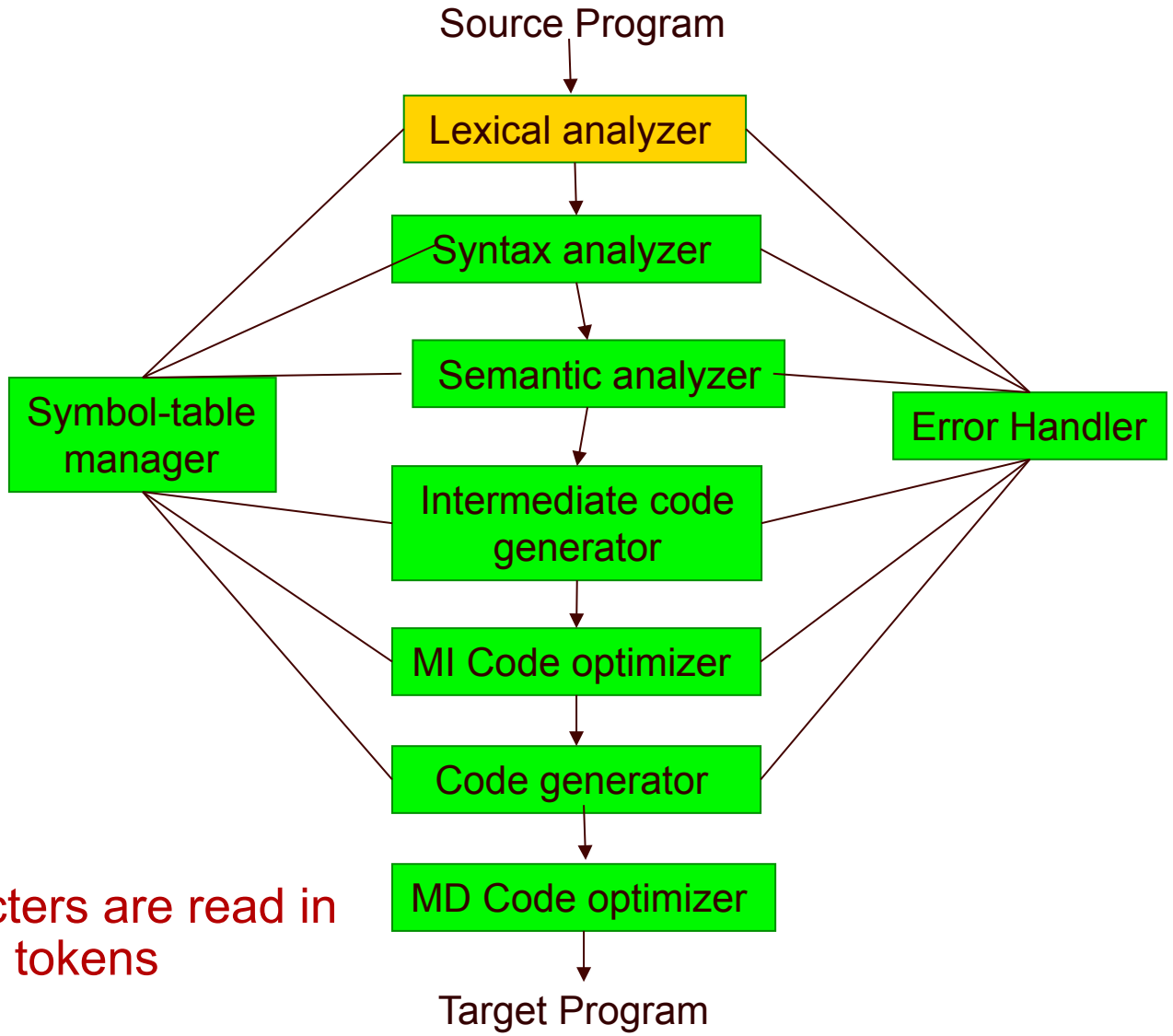
Copyright (c) 2012 Ioanna Dionysiou

Phases of a compiler



Copyright (c) 2012 Ioanna Dionysiou

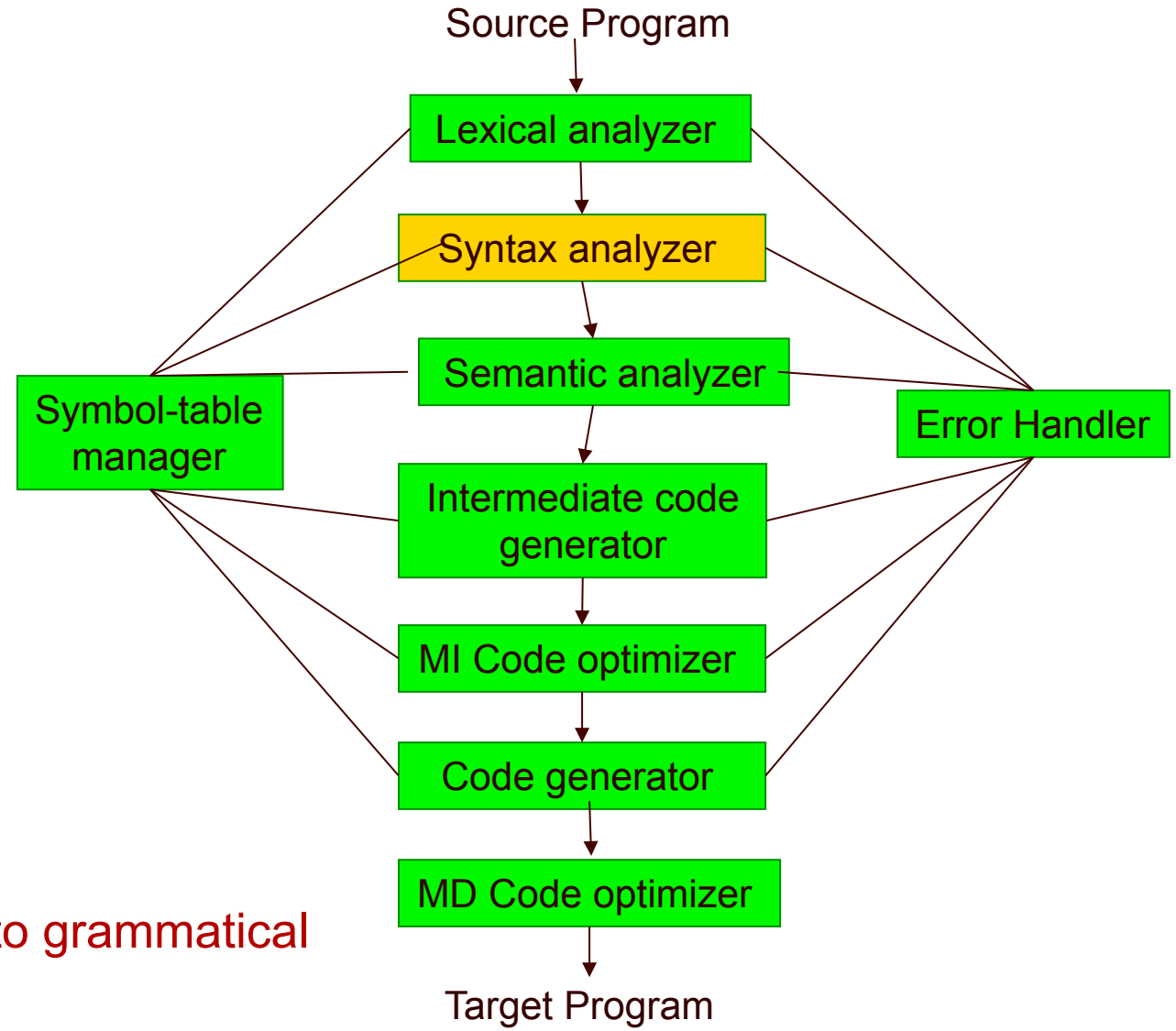
The phases of a compiler



- Lexical analyzer
  - Stream of characters are read in and grouped into tokens

Copyright (c) 2012 Ioanna Dionysiou

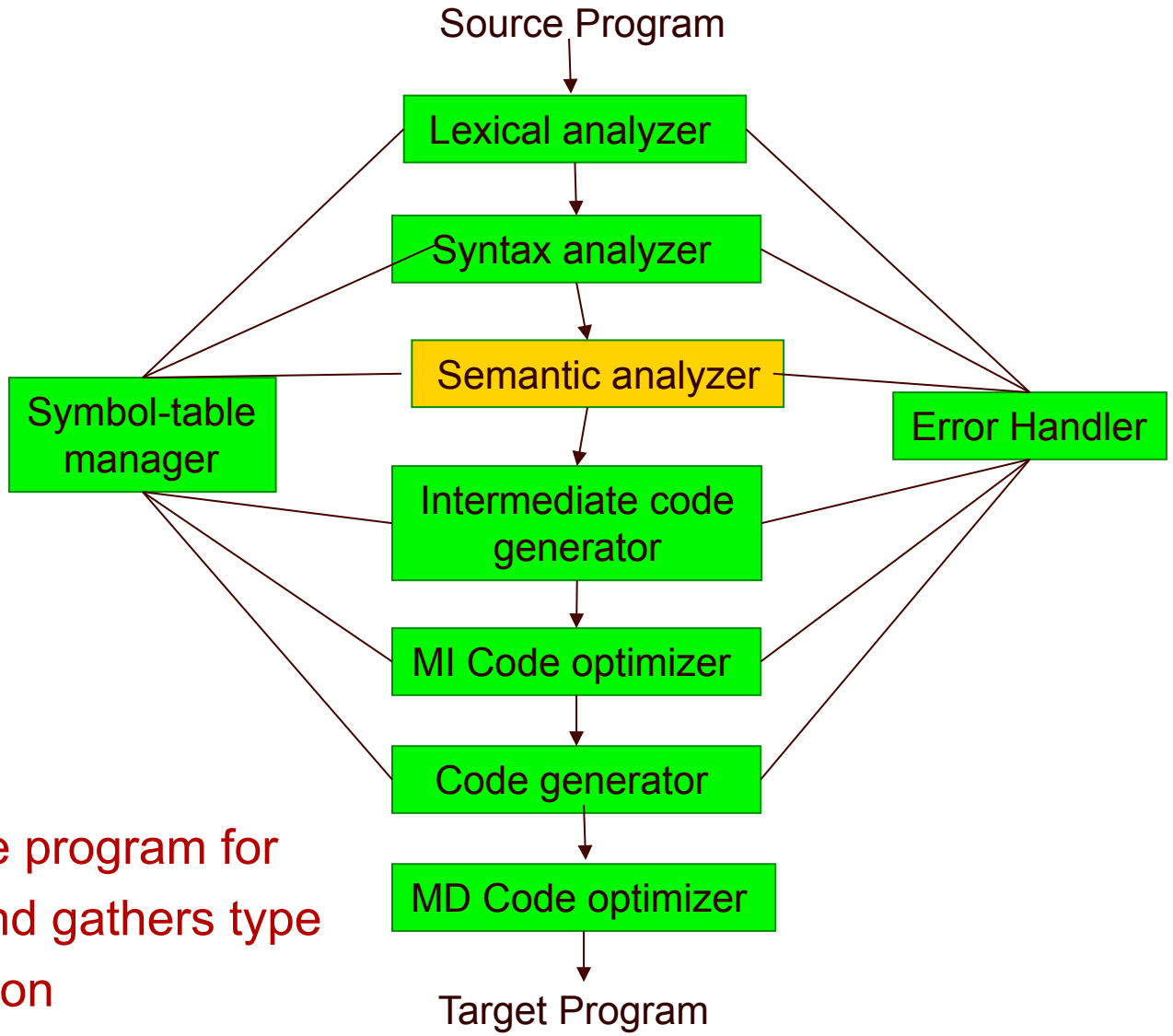
The phases of a compiler



- Syntax analyzer
  - Group tokens into grammatical phrases

Copyright (c) 2012 Ioanna Dionysiou

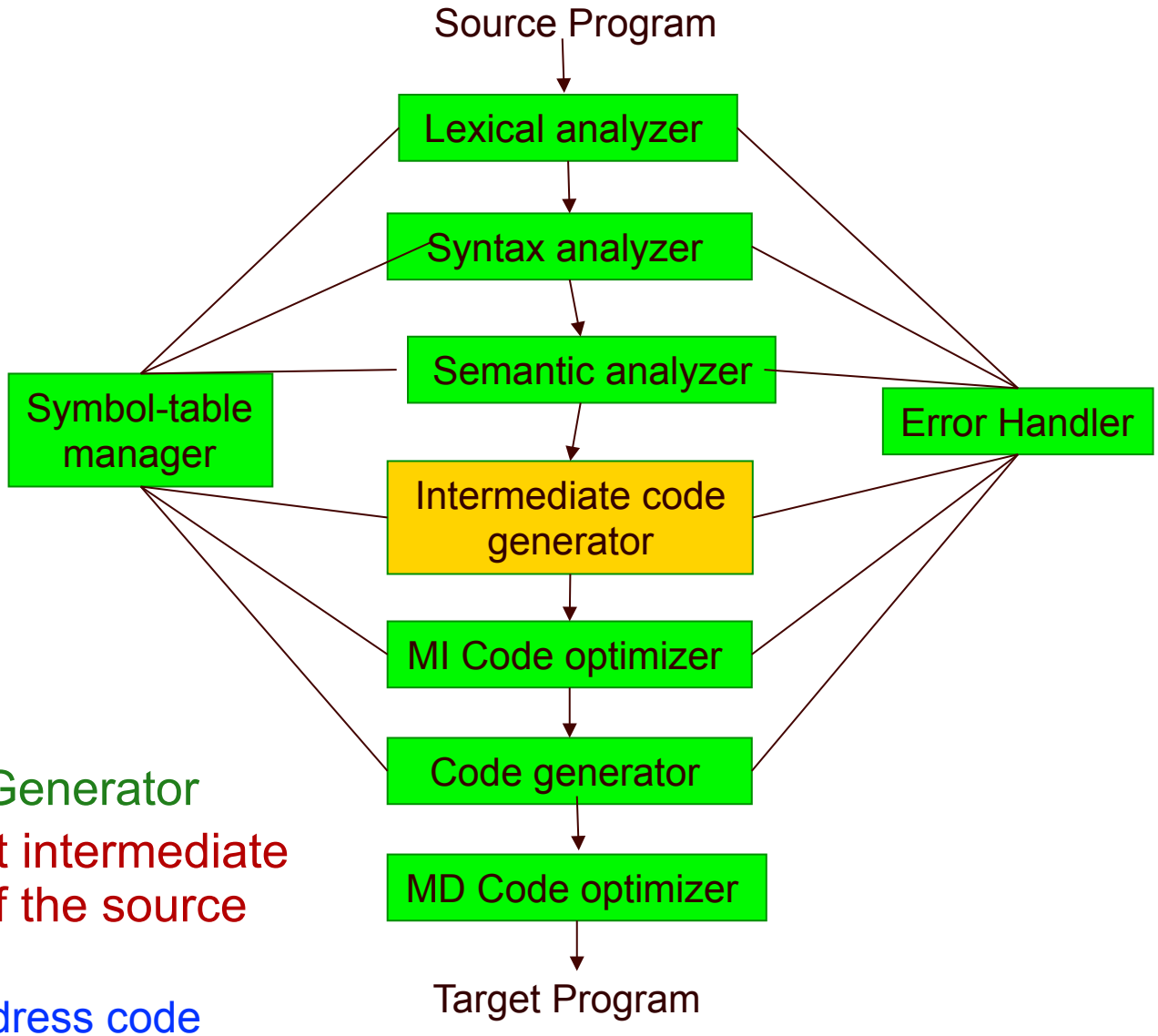
The phases of a compiler



- Semantic analyzer
  - Checks the source program for semantic errors and gathers type checking information
    - E.g.use a real number as index to array elements

Copyright (c) 2012 Ioanna Dionysiou

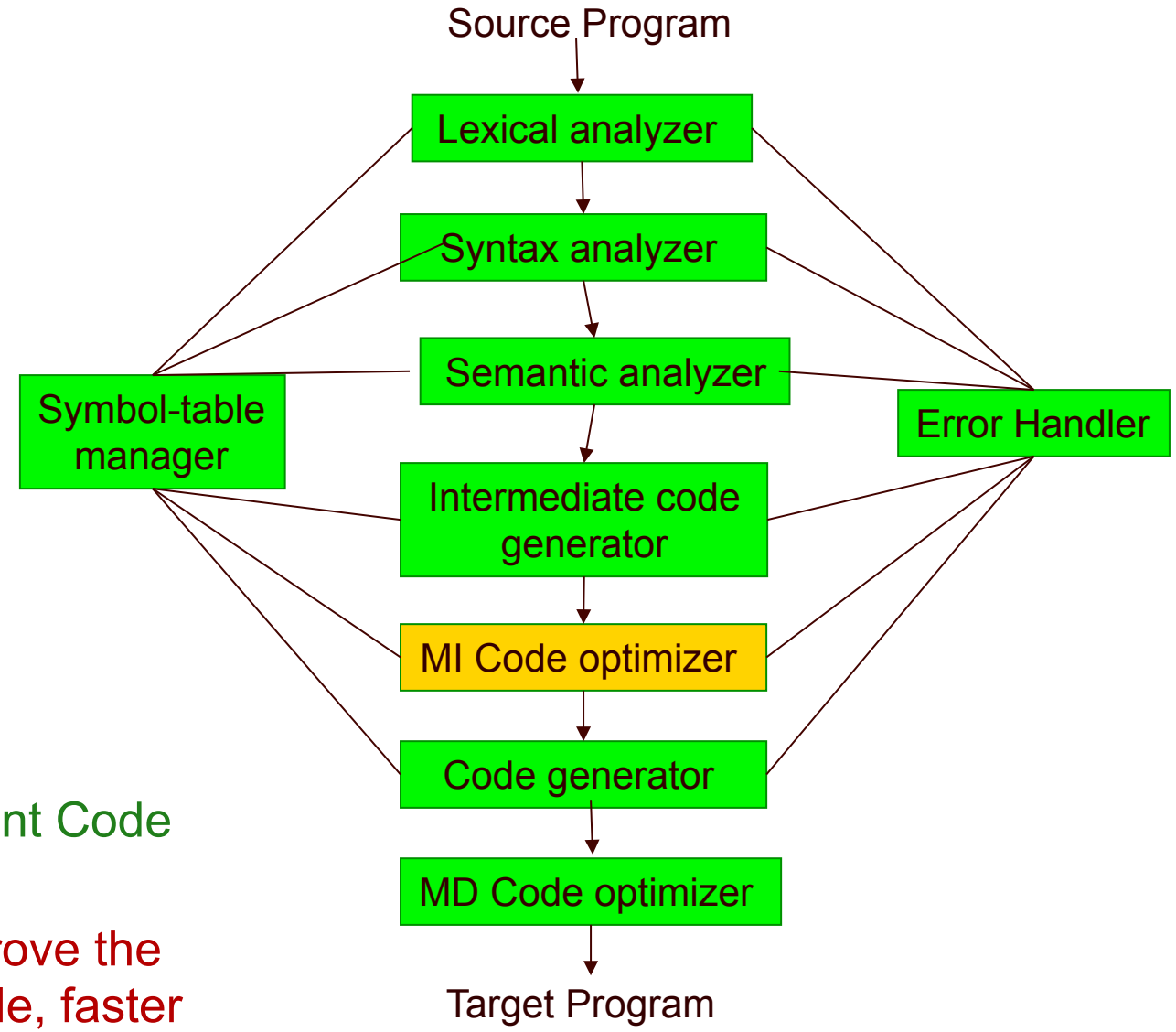
The phases of a compiler



- Intermediate Code Generator
  - Generate explicit intermediate representation of the source program
    - E.g. Three-address code

Copyright (c) 2012 Ioanna Dionysiou

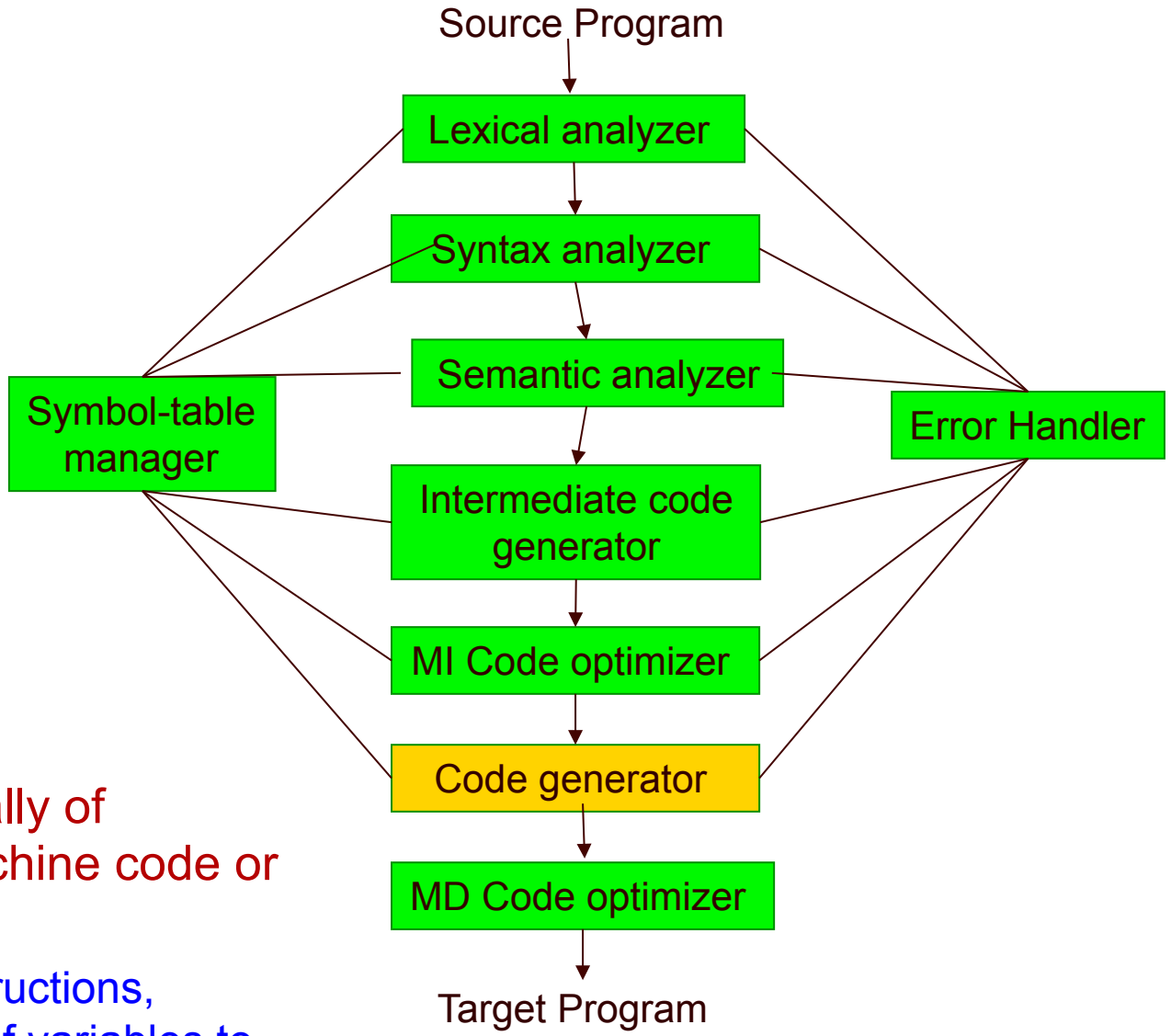
The phases of a compiler



- Machine-Independent Code Optimizer
  - Attempts to improve the intermediate code, faster performance results

Copyright (c) 2012 Ioanna Dionysiou

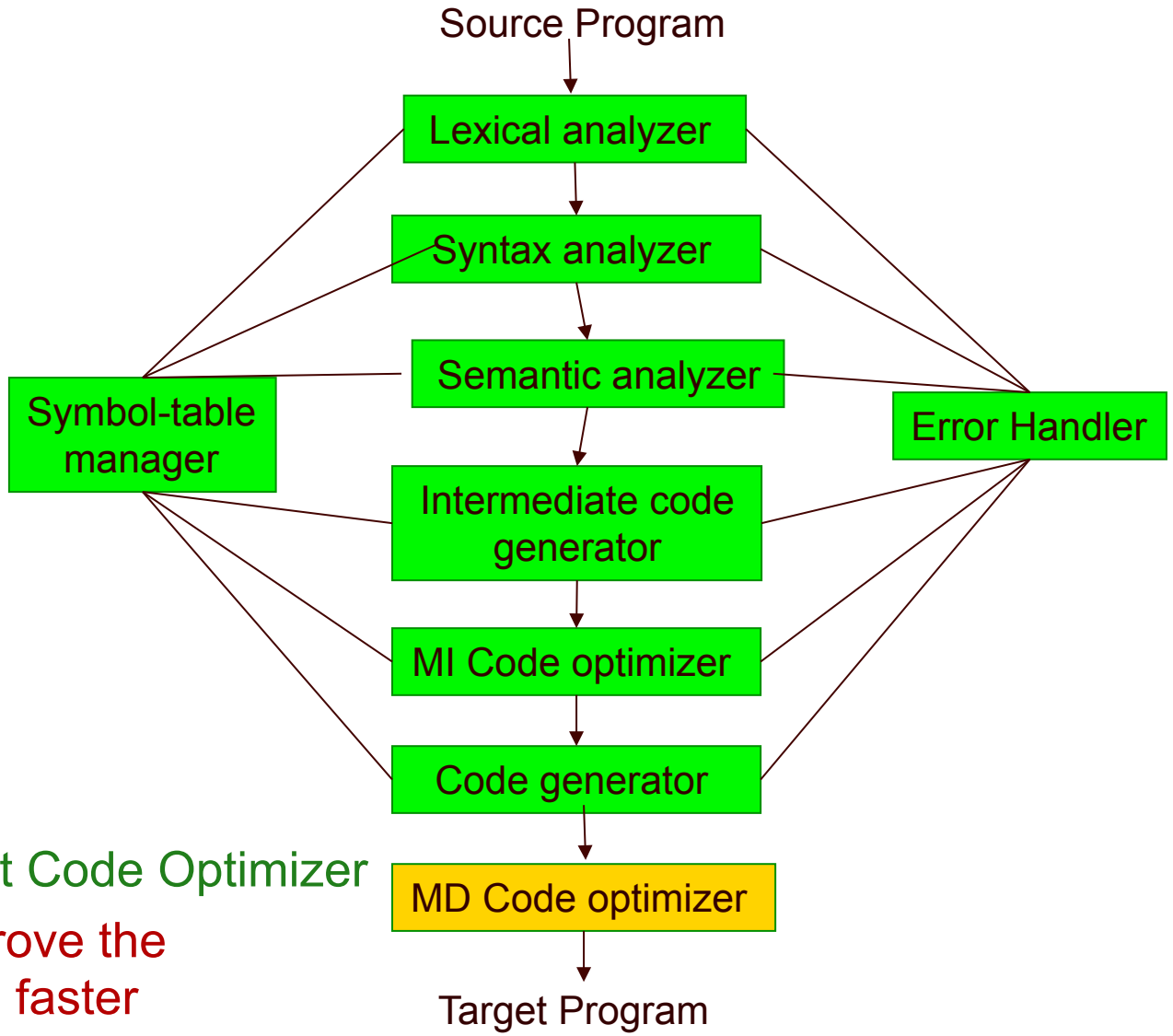
The phases of a compiler



- Code Generator
  - Consists normally of relocatable machine code or assembly code
    - Machine instructions, assignment of variables to registers

Copyright (c) 2012 Ioanna Dionysiou

The phases of a compiler



- Machine-Dependent Code Optimizer
  - Attempts to improve the generated code, faster performance results

Copyright (c) 2012 Ioanna Dionysiou

Lecture Outline

- Chapter 1 - Introduction
  - Definition
  - Structure of Compiler
    - Compilation parts
      - Analysis, Synthesis
    - Phases of a compiler
  - An Example: Translation of a statement
  - Evolution of Programming Languages
  - Building a Compiler and Applications of Compiler Technology

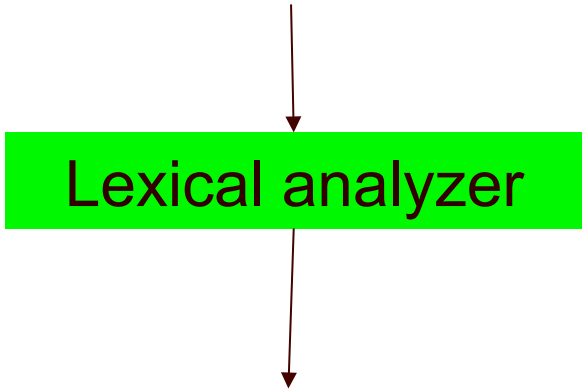
Copyright (c) 2012 Ioanna Dionysiou

Example: Translation of a statement

SYMBOL TABLE

position	
initial	
rate	
…	

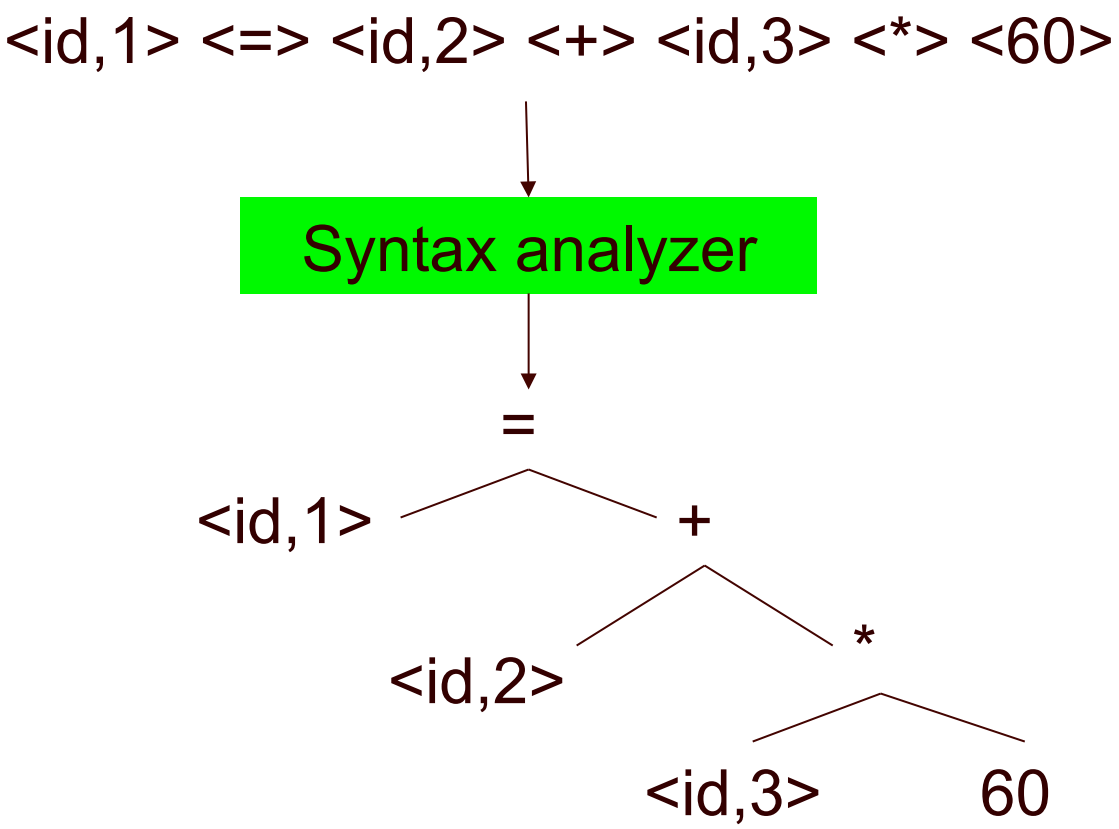
position = initial + rate * 60



<id,1> <=> <id,2> <+> <id,3> <*> <60>

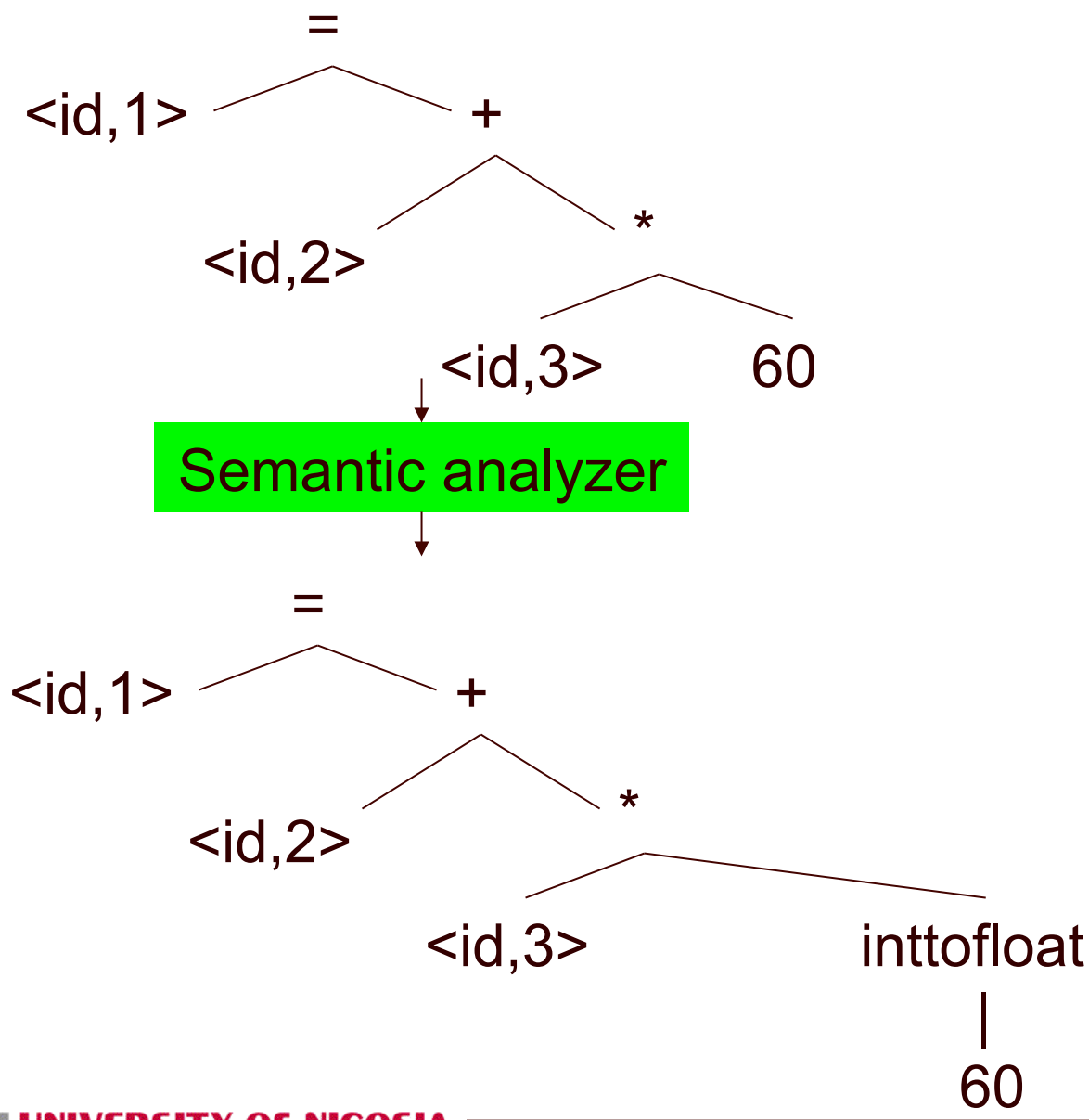
Copyright (c) 2012 Ioanna Dionysiou

Example: Translation of a statement



Copyright (c) 2012 Ioanna Dionysiou

Example: Translation of a statement

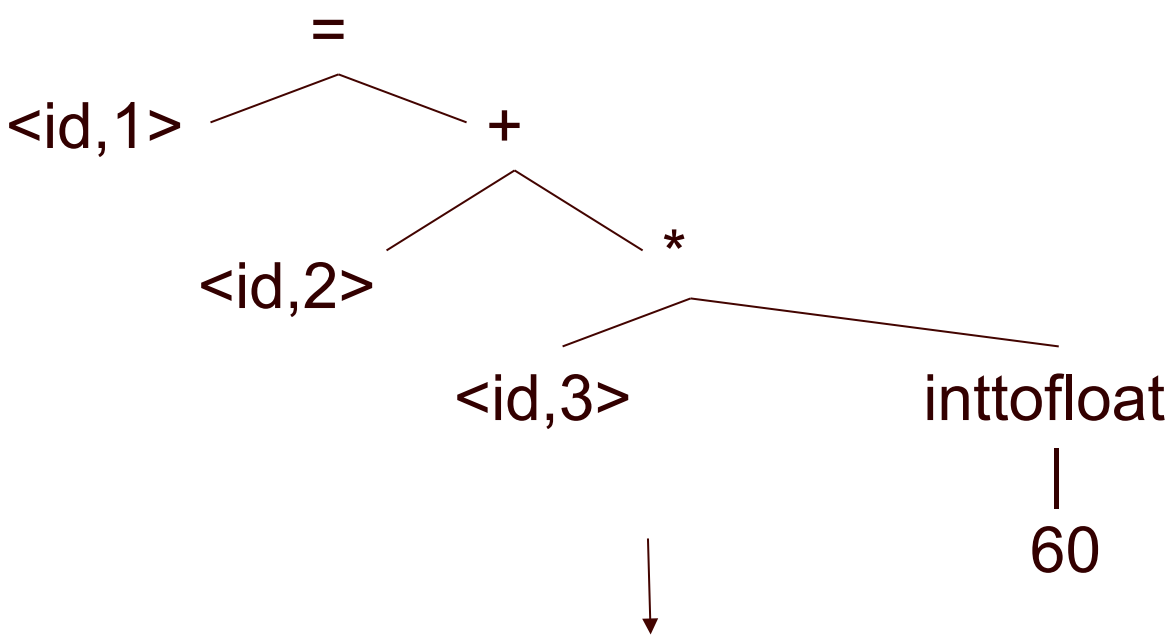


SYMBOL TABLE

position	FLOAT
initial	FLOAT
rate	FLOAT
…	

Copyright (c) 2012 Ioanna Dionysiou

Example: Translation of a statement



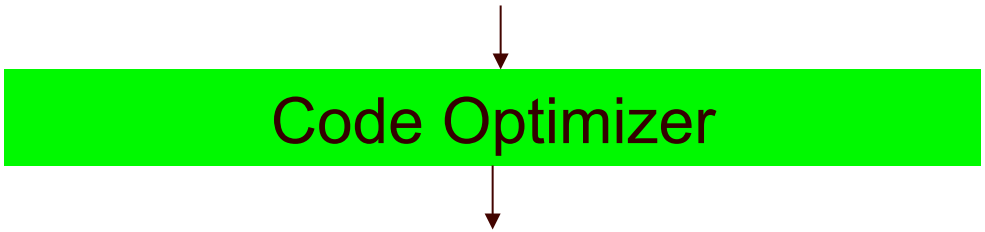
Intermediate Code Generator

```
temp1 = inttofloat(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```


Copyright (c) 2012 Ioanna Dionysiou

Example: Translation of a statement

```
temp1 = inttoreal(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```

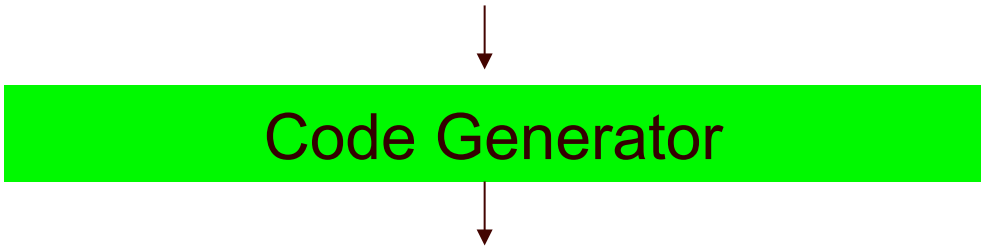



```
temp1 = id3 * 60.0
id1 = id2 + temp1
```


Copyright (c) 2012 Ioanna Dionysiou

Example: Translation of a statement

temp1 = id3 * 60.0
id1 = id2 + temp1



MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1

Copyright (c) 2012 Ioanna Dionysiou

Lecture Outline

- Chapter 1 - Introduction
  - Definition
  - Structure of Compiler
    - Compilation parts
      - Analysis, Synthesis
    - Phases of a compiler
  - An Example: Translation of a statement
  - Evolution of Programming Languages
  - Building a Compiler and Applications of Compiler Technology

Copyright (c) 2012 Ioanna Dionysiou

First Computers

- Appeared in 1940's
- Programmed in machine language
  - 0's and 1's
- Low-level operations
  - Move data, add contents of registers
- Programming problems
  - Slow, tedious, error prone
  - Very hard to understand and modify

Copyright (c) 2012 Ioanna Dionysiou

Move to Higher-level languages

- Early 50's
  - Assembly languages
- Mid-late 50's
  - Fortran, Cobol, Lisp
- Today
  - Thousands of programming languages
  - Various classifications
    - Generation
    - Imperative vs. declarative
    - Object oriented
    - Scripting

Copyright (c) 2012 Ioanna Dionysiou

More on Higher-Level Languages

- Generation
  - 1st : machine languages
  - 2nd : assembly languages
  - 3rd : higher-level languages (Cobol, C, C++, Java)
  - 4th : application-specific languages (SQL)
  - 5th : logic-based languages (Prolog)
- Imperative vs. Declarative
  - How a computation is done: C, C++, Java
  - What computation is to be done: Prolog
- Object-Oriented
  - Java, C#
- Scripting
  - JavaScript, PHP

Copyright (c) 2012 Ioanna Dionysiou

Lecture Outline

- Chapter 1 - Introduction
  - Definition
  - Structure of Compiler
    - Compilation parts
      - Analysis, Synthesis
    - Phases of a compiler
  - An Example: Translation of a statement
  - Evolution of Programming Languages
  - Building a Compiler and Applications of Compiler Technology

Copyright (c) 2012 Ioanna Dionysiou

Building a Compiler

- Compiler development is challenging
  - A compiler must accept ALL source programs that conform to the language specification
    - Set of source programs -> Infinite!
    - Millions of lines of code
  - An transformation performed by the compiler must preserve the meaning of the source program

Copyright (c) 2012 Ioanna Dionysiou

Modeling in Compiler Design

- Study of compilers
  - How do we design the right mathematical models?
  - How do we choose the right algorithms?
- Fundamental models
  - Finite-state machines
  - Regular expressions
  - Context-free grammars
  - Trees

Copyright (c) 2012 Ioanna Dionysiou

Compiler-Construction Tools

- Luckily, there are tools available
  - Parser generators
  - Scanner generators
  - Syntax-directed translation engines
  - Code-generator generators
  - Data-flow analysis engines
  - Compiler-construction toolkits

Copyright (c) 2012 Ioanna Dionysiou

Science of Code Optimization

- Optimization
  - Attempts that a compiler makes to produce more efficient code
- Compiler Optimizations must meet the following design objectives
  - Optimization must be correct
  - Optimization must improve the performance of many programs
  - Compilation time must be kept reasonable
  - Engineering effort required must be manageable

Copyright (c) 2012 Ioanna Dionysiou

More on Optimization

- Correct optimization
  - Generate fast code that is correct!!!
- Improve performance
  - Normally means reducing speed of program execution
  - Minimizing size of generated code (embedded applications)
  - Minimizing power consumption (mobile devices)
- Short compilation time
  - To support rapid development and debugging cycle
- Keep it simple
  - Compiler is a complex system - Keep system simple so that engineering and maintenance costs of the compiler are manageable
    - Prioritize optimizations, implement those that lead to greatest benefits

Copyright (c) 2012 Ioanna Dionysiou

Compiler Technology Apps.

- Optimizations for Computer Architectures
  - Parallelism at the instruction level and processor level
    - Compiler techniques are developed to generate code automatically for such machines from sequential programs

Copyright (c) 2012 Ioanna Dionysiou