Lecture 1

Compiler Design

Awanish Pandey

Department of Computer Science and Engineering
Indian Institute of Technology
Roorkee

January 21, 2025

Introduction

Instructor

- Awanish Pandey

Introduction

Instructor

- Awanish Pandey
- Office: MFS building, 1st floor

Introduction

Instructor

- Awanish Pandey
- Office: MFS building, 1st floor
- Please use teams for any communication related to the course.

Introduction

Instructor

- Awanish Pandey
- Office: MFS building, 1st floor
- Please use teams for any communication related to the course.
- E-Mail: ap@cs.iitr.ac.in

Introduction

Instructor

- Awanish Pandey
- Office: MFS building, 1st floor
- Please use teams for any communication related to the course.
- E-Mail: ap@cs.iitr.ac.in
- **Subject must start from [CSN-352] Email not complying with this rule will NOT be entertained.**

Introduction

Instructor

- Awanish Pandey
- Office: MFS building, 1st floor
- Please use teams for any communication related to the course.
- E-Mail: ap@cs.iitr.ac.in
- **Subject must start from [CSN-352] Email not complying with this rule will NOT be entertained.**

TA

To be added

Acknowledgments

The course structure is based on Compilers: Principles, Techniques, and Tools by Aho, Sethi, Ullman and Lam (2nd Edition).

Acknowledgments

The course structure is based on Compilers: Principles, Techniques, and Tools by Aho, Sethi, Ullman and Lam (2nd Edition).

Slides are based on the slides of the late Prof Sanjeev K Aggarwal.

Learning from this course

- Course has theoretical and practical components. Both are needed in implementing programming languages. The focus will be on the practical application of the theory.

Learning from this course

- Course has theoretical and practical components. Both are needed in implementing programming languages. The focus will be on the practical application of the theory.
- Emphasis will be on algorithms and data structures rather than proofs of correctness of algorithms.

Learning from this course

- Course has theoretical and practical components. Both are needed in implementing programming languages. The focus will be on the practical application of the theory.
- Emphasis will be on algorithms and data structures rather than proofs of correctness of algorithms.
- Theory of lexical analysis, parsing, type checking, runtime system, code generation, optimization

Learning from this course

- Course has theoretical and practical components. Both are needed in implementing programming languages. The focus will be on the practical application of the theory.
- Emphasis will be on algorithms and data structures rather than proofs of correctness of algorithms.
- Theory of lexical analysis, parsing, type checking, runtime system, code generation, optimization
- Techniques for developing lexical analyzers, parsers, type checkers, run-time systems, code generators, and optimization.

Learning from this course

- Course has theoretical and practical components. Both are needed in implementing programming languages. The focus will be on the practical application of the theory.
- Emphasis will be on algorithms and data structures rather than proofs of correctness of algorithms.
- Theory of lexical analysis, parsing, type checking, runtime system, code generation, optimization
- Techniques for developing lexical analyzers, parsers, type checkers, run-time systems, code generators, and optimization.
- Students will be confident that they can design, develop, understand, modify, enhance, and maintain compilers for (even complex!) programming languages

Course Structure

Relative Weight

Project	25
Mid-Term Evaluation	25
End-Term Evaluation	50

Course Structure

Relative Weight

Project	25
Mid-Term Evaluation	25
End-Term Evaluation	50

General Instructions

- Please give feedback throughout the course, either personally or over the mail.

Course Structure

Relative Weight

Project	25
Mid-Term Evaluation	25
End-Term Evaluation	50

General Instructions

- Please give feedback throughout the course, either personally or over the mail.
- Feel free to discuss anything with me.

Course Structure

Relative Weight

Project	25
Mid-Term Evaluation	25
End-Term Evaluation	50

General Instructions

- Please give feedback throughout the course, either personally or over the mail.
- Feel free to discuss anything with me.
- Please be on time. Respect your own time and my/TAs time.

Course Structure

Relative Weight

Project	25
Mid-Term Evaluation	25
End-Term Evaluation	50

General Instructions

- Please give feedback throughout the course, either personally or over the mail.
- Feel free to discuss anything with me.
- Please be on time. Respect your own time and my/TAs time.
- Keep your electronic devices in silent mode.

Course Structure

Relative Weight

Project	25
Mid-Term Evaluation	25
End-Term Evaluation	50

General Instructions

- Please give feedback throughout the course, either personally or over the mail.
- Feel free to discuss anything with me.
- Please be on time. Respect your own time and my/TAs time.
- Keep your electronic devices in silent mode.
- **DO NOT CHEAT**

Project Detail

- Group of 4

Project Detail

- Group of 4
- Implement end-to-end toy compiler

Project Detail

- Group of 4
- Implement end-to-end toy compiler
- Each group has to choose a source, intermediate, target, and implementation language.

Project Detail

- Group of 4
- Implement end-to-end toy compiler
- Each group has to choose a source, intermediate, target, and implementation language.
- Project form

Project Detail

- Group of 4
- Implement end-to-end toy compiler
- Each group has to choose a source, intermediate, target, and implementation language.
- Project form

Assignment 1 (Lexical Analysis)	10%
Assignment 2 (Syntax Analysis)	20%
Assignment 3 (Intermediate Code generation)	30%
Assignment 4 (Optimization and Target code generation)	40%

History

Two major ways of implementing Programming Languages

History

Two major ways of implementing Programming Languages

- Interpreter

History

Two major ways of implementing Programming Languages

- Interpreter
- Compiler

History

Two major ways of implementing Programming Languages

- Interpreter
- Compiler

Early Machines

- IBM developed 704 in 1954. All programming was done in assembly language. The cost of software development far exceeded the cost of hardware. Low productivity.

History

Two major ways of implementing Programming Languages

- Interpreter
- Compiler

Early Machines

- IBM developed 704 in 1954. All programming was done in assembly language. The cost of software development far exceeded the cost of hardware. Low productivity.
- Speedcoding interpreter: programs ran about 10 times slower than hand written assembly code

History

Two major ways of implementing Programming Languages

- Interpreter
- Compiler

Early Machines

- IBM developed 704 in 1954. All programming was done in assembly language. The cost of software development far exceeded the cost of hardware. Low productivity.
- Speedcoding interpreter: programs ran about 10 times slower than hand written assembly code
- John Backus (in 1954): Proposed a program that translated high level expressions into native machine code. Skeptism all around. Most people thought it was impossible

History

Two major ways of implementing Programming Languages

- Interpreter
- Compiler

Early Machines

- IBM developed 704 in 1954. All programming was done in assembly language. The cost of software development far exceeded the cost of hardware. Low productivity.
- Speedcoding interpreter: programs ran about 10 times slower than hand written assembly code
- John Backus (in 1954): Proposed a program that translated high level expressions into native machine code. Skeptism all around. Most people thought it was impossible
- Fortran I project (1954-1957): The first compiler was released

Fortran 1

- The whole new field of compiler design was started

Fortran 1

- The whole new field of compiler design was started
- More than half the programmers were using `Fortran` by 1958

Fortran 1

- The whole new field of compiler design was started
- More than half the programmers were using `Fortran` by 1958
- The development time was cut down to half

Fortran 1

- The whole new field of compiler design was started
- More than half the programmers were using `Fortran` by 1958
- The development time was cut down to half
- Led to the enormous amount of theoretical work (lexical analysis, parsing, optimization, structured programming, code generation, error recovery, etc.)

Fortran 1

- The whole new field of compiler design was started
- More than half the programmers were using `Fortran` by 1958
- The development time was cut down to half
- Led to the enormous amount of theoretical work (lexical analysis, parsing, optimization, structured programming, code generation, error recovery, etc.)
- **Modern compilers preserve the basic structure of the `Fortran I` compiler !!!**

What are Compilers

What are Compilers

Basics

What are Compilers

Basics

- What is High-Level Languages
  - Expressive: matches our notion of languages (and application?!)

What are Compilers

Basics

- What is High-Level Languages
  - Expressive: matches our notion of languages (and application?!)
  - Redundant to help avoid programming errors

What are Compilers

Basics

- What is High-Level Languages
  - Expressive: matches our notion of languages (and application?!)
  - Redundant to help avoid programming errors
- Can the computer understand this?

What are Compilers

Basics

- What is High-Level Languages
  - Expressive: matches our notion of languages (and application?!)
  - Redundant to help avoid programming errors
- Can the computer understand this?
- Who will convert it?

What are Compilers

Basics

- What is High-Level Languages
  - Expressive: matches our notion of languages (and application?!)
  - Redundant to help avoid programming errors
- Can the computer understand this?
- Who will convert it?
- Goals of translation

What are Compilers

Basics

- What is High-Level Languages
  - Expressive: matches our notion of languages (and application?!)
  - Redundant to help avoid programming errors
- Can the computer understand this?
- Who will convert it?
- Goals of translation
  - Good performance for the generated code

What are Compilers

Basics

- What is High-Level Languages
  - Expressive: matches our notion of languages (and application?!)
  - Redundant to help avoid programming errors
- Can the computer understand this?
- Who will convert it?
- Goals of translation
  - Good performance for the generated code
  - Good compile-time performance

What are Compilers

Basics

- What is High-Level Languages
  - Expressive: matches our notion of languages (and application?!)
  - Redundant to help avoid programming errors
- Can the computer understand this?
- Who will convert it?
- Goals of translation
  - Good performance for the generated code
  - Good compile-time performance
  - Maintainable code

What are Compilers

Basics

- What is High-Level Languages
  - Expressive: matches our notion of languages (and application?!)
  - Redundant to help avoid programming errors
- Can the computer understand this?
- Who will convert it?
- Goals of translation
  - Good performance for the generated code
  - Good compile-time performance
  - Maintainable code
  - What about correctness?