Compilers

CMSC 430

This Lecture

What is a compiler?

Before we start...

Before we start...

1. Who am I?

Before we start...

1. Who am I?

Before we start...

1. Who am I?
2. Who are the TAs?

Before we start...

1. Who am I?
2. Who are the TAs?
3. Some admin

Who am I?

Who am I?

José Manuel Calderón Trilla
PhD in Compilers

Who am I?

José Manuel Calderón Trilla
PhD in Compilers

Who are the TAs?

Who are the TAs?

- Benjamin Glover Quiring
- William Chung
- Drhuv Maniktala

Admin stuff

- Website: The “source of truth”
- ELMS: Announcements and logistics
- Discord: Questions, Interaction, Office Hours
- Gradescope: Homework

What is a compiler?

What is a compiler?

If I've done my job right: Everything.

What is a compiler?

Compilers are everwhere, as compiler-writers this is a blessing and a curse.

Compilers are *everywhere*

Compilers are *everywhere*

I have seen compilers on:

Compilers are *everywhere*

I have seen compilers on:

- Phones

Compilers are *everywhere*

I have seen compilers on:

- Phones
- Printers

Compilers are *everywhere*

I have seen compilers on:

- Phones
- Printers
- SmartCards

Compilers are *everywhere*

I have seen compilers on:

- Phones
- Printers
- SmartCards
- Space Stuff (NASA Deep Space 1)

Compilers are *everywhere*

I have seen compilers on:

- Phones
- Printers
- SmartCards
- Space Stuff (NASA Deep Space 1)
- Scariest of all:

Compilers are *everywhere*

I have seen compilers on:

- Phones
- Printers
- SmartCards
- Space Stuff (NASA Deep Space 1)
- Scariest of all:
    - The Linux Kernel itself

What is a *compiler*?

- Easy:

What is a *compiler*?

- Easy:
    - `compiler : SourceProgram -> TargetProgram`

Source Programs

Source Programs

- Concrete Syntax

Source Programs

- Concrete Syntax
- Parsers

Source Programs

- Concrete Syntax
- Parsers
- Grammars (LR, LALR, GLR, etc.)

Source Programs

- Concrete Syntax
- Parsers
- Grammars (LR, LALR, GLR, etc.)
- i.e. ”what the programmer writes”

Target Programs

Target Programs

Could be anything, but...

Target Programs

Often:

- Machine Code

Target Programs

Often:

- Machine Code
- Byte Code

Target Programs

Often:

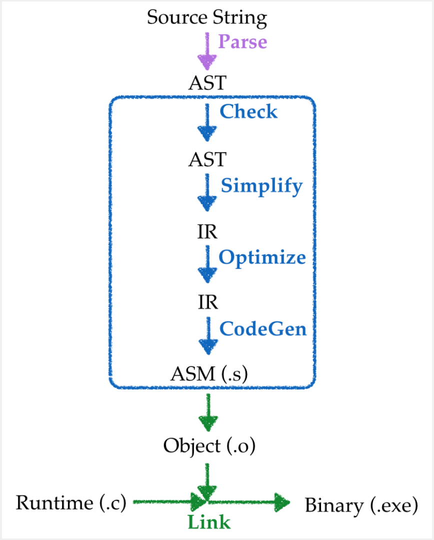
- Machine Code
- Byte Code
- Another Programming Language (i.e. C to Javescript)

Target Programs

Often:

- Machine Code
- Byte Code
- Another Programming Language (i.e. C to Javescript)
- Information/Data for tooling

Show me


Source String
Parse
AST
Check
AST
Simplify
IR
Optimize
IR
CodeGen
ASM (.s)
Object (.o)
Runtime (.c)
Link
Binary (.exe)

It’s a convenient lie

It’s a convenient lie

Compilers don’t always have that form.

It’s a convenient lie

Compilers don’t always have that form. In fact, it is increasingly rare that they do!

It’s a convenient lie

Compilers don’t always have that form. In fact, it is increasingly rare that they do!

Regardless, it’s a useful *intuition* and allows us to cover all the important concepts.

It’s a convenient lie

Can anyone think of when a compiler *does not* have that form?

What’s important about the Souce-Target relationship?

What's important about the Souce-Target relationship?

Put another way: When you *write* a program, what do you expect the implementation of the programming language to do (or not do)?

This class

This class

CMSC 430 is about the study of that relationship. In particular:

This class

CMSC 430 is about the study of that relationship. In particular:

- How do we bridge the gap between ‘high-level’ and ‘low-level’ languages?

This class

CMSC 430 is about the study of that relationship. In particular:

- How do we bridge the gap between ‘high-level’ and ‘low-level’ languages?
  - CMSC 330: ‘high-level’

This class

CMSC 430 is about the study of that relationship. In particular:

- How do we bridge the gap between ‘high-level’ and ‘low-level’ languages?
    - CMSC 330: ‘high-level’
    - CMSC 216: ‘low-level’

This class

CMSC 430 is about the study of that relationship. In particular:

- How do we bridge the gap between ‘high-level’ and ‘low-level’ languages?
- What does it *mean* to bridge that gap?

This class

CMSC 430 is about the study of that relationship. In particular:

- How do we bridge the gap between ‘high-level’ and ‘low-level’ languages?
- What does it *mean* to bridge that gap?
  - Do *all* the features of one language need to be present in the other?

This class

CMSC 430 is about the study of that relationship. In particular:

- How do we bridge the gap between ‘high-level’ and ‘low-level’ languages?
- What does it *mean* to bridge that gap?
- In other words: This class is about *abstraction*.

The result

The result

- Goal is to write a compiler for `NanoML -> X86`

The result

- Goal is to write a compiler for `NanoML -> X86`
- This includes looking at

The result

- Goal is to write a compiler for `NanoML -> X86`
- This includes looking at
    - Parsing

The result

- Goal is to write a compiler for `NanoML -> X86`
- This includes looking at
    - Parsing
    - Checking and Validation

The result

- Goal is to write a compiler for `NanoML -> X86`
- This includes looking at
  - Parsing
  - Checking and Validation
  - Simplification and Normalization (sometimes known as *Elaboration*)

The result

- Goal is to write a compiler for `NanoML -> X86`
- This includes looking at
  - Parsing
  - Checking and Validation
  - Simplification and Normalization (sometimes known as *Elaboration*)
  - Optimization

The result

- Goal is to write a compiler for `NanoML -> X86`
- This includes looking at
  - Parsing
  - Checking and Validation
  - Simplification and Normalization (sometimes known as *Elaboration*)
  - Optimization
  - Code Generation

Why is this interesting?

Why is this interesting?

I find compilers/interpreters/PLs interesting in their own right, but not everyone shares the same aesthetic values.

Why is this interesting?

I find compilers/interpreters/PLs interesting in their own right, but not everyone shares the same aesthetic values.

- Compilers combine several important aspects of CS

Why is this interesting?

I find compilers/interpreters/PLs interesting in their own right, but not everyone shares the same aesthetic values.

- Compilers combine several important aspects of CS
    - Theory (parsing, types, ASTs, semantics, etc.)

Why is this interesting?

I find compilers/interpreters/PLs interesting in their own right, but not everyone shares the same aesthetic values.

- Compilers combine several important aspects of CS
  - Theory (parsing, types, ASTs, semantics, etc.)
  - Systems (computer architectures, performance, sys-calls, etc.)

Why is this interesting?

I find compilers/interpreters/PLs interesting in their own right, but not everyone shares the same aesthetic values.

- Compilers combine several important aspects of CS
- Unlike some other major software artifacts, compilers *can be* well specified!

Why is this interesting?

I find compilers/interpreters/PLs interesting in their own right, but not everyone shares the same aesthetic values.

- Compilers combine several important aspects of CS
- Unlike some other major software artifacts, compilers *can be* well specified!
    - This lets us reason about *correctness*

Why is this interesting?

I find compilers/interpreters/PLs interesting in their own right, but not everyone shares the same aesthetic values.

- Compilers combine several important aspects of CS
- Unlike some other major software artifacts, compilers *can be* well specified!
- Develop good habits on non-trivial software projects.

Most important point of the day

Most important point of the day

The ‘dragon book’ lied.

Most important point of the day

Compilers are not scary.

How we’ll do it

How we’ll do it

We will write *several* compilers.

How we'll do it

We will write *several* compilers.
Each will be simple and implement a specific feature or set of related features.

How we'll do it

We will write *several* compilers.
Each will be simple and implement a specific feature or set of related features.
Combining these compilers is what gives us powerful abstractions.

Any Questions?

Closing thoughts

Closing thoughts

Please read over the syllabus on the website and reach out to me if there are any issues.

Thanks for your time!