Introducti

Dr. Liam O'Connor
University of Edinburgh LFCS
UNSW, Term 3 2020

Who are we?

I am Liam O'Connor, a lecturer at the University of Edinburgh, and former convenor of this course. I am pre-recording the first 5 weeks of lectures for this iteration, to ensure a smooth hand-ov

Dr. Christine Rizk
who works on, amo
with data61.

Vivian Dang and Matthew Di Meglio are the tutors for this year. Vi
Christine on security type systems, and Matthew will be immin
at Edinburgh.

Who are we?

I am Liam O'Connor, a lecturer at the University of Edinburgh, and former convenor of this course. I am pre-recording the first 5 weeks of lectures for this iteration, to ensure a smooth hand-ov

Dr. Christine Rizk
who works on, amo
with data61.

Vivian Dang and Matthew Di Meglio are the tutors for this year. Vi
Christine on security type systems, and Matthew will be immin
at Edinburgh.

Who are we?

I am Liam O'Connor, a lecturer at the University of Edinburgh, and former convenor of this course. I am pre-recording the first 5 weeks of lectures for this iteration, to ensure a smooth hand-ov

Dr. Christine Rizk
who works on, amo
with data61.

Vivian Dang and Matthew di Meglio are the tutors for this year. Vi
Christine on security type systems, and Matthew will be immin
at Edinburgh.

Contacting Us

http://www.cse.unsw.edu.au/~cs3161

Forum

There is a Piazza for

should typically be made there. You can ask us private question

solutions to other students.

I highly recommend disabling the Piazza Careers rubbish.

Administrative questions should be sent to cs3161@cse.unsw.edu.au.

What do we expect?

Maths

This course uses a si eed to be reasonably comf 81 is neither necessary nor su

Programming

We expect you to be familiar with C and at least one other progra

Course assignments 1 and 2 are in Haskell. Only very simple Hask some self-study may be needed.

What do we expect?

Maths

This course uses a si eed to be reasonably comf 81 is neither necessary nor su

Programming

We expect you to be familiar with C and at least one other program Course assignments 1 and 2 are in Haskell. Only very simple Hask some self-study may be needed.

Assessment

Final Exam	

Tutorials

- Start this we
- You may ch
- Please attempt some of the questions beforehand.

Assignment 0

- Focuses on t
- It will be relea
- Aim to have m
- 10% penalty for one day late, 25% for two, 50% for three and

Assignments

- Given a form
- Released ar
- Approxim
- 10% penalty for one day late, 25% for two, 50% for three and

Lectures

- My lectures (
  Blackboar

- We may use t
- All board-work will be done digitally and made available t
- Separate lecture notes are also published.

Books

There is no textbook for this course. Regular written lecture notes are made available throughout the semester, along with challenge exercises.

Much of the course
explanations ma
these books exact

- *Types and Programming Languages* by Benja
  https://www.cis.upenn.edu/~bcpierce
- *Practical Foundations for Programming Language* ridge University Press. http://www.cs.cmu.edu/~rwh/pfpl.html

Course Content

This is a programm n the three R's of compu

Read a

Write your own programming languages; and

Reason about programming languages in a rigorou

Why Read?

The choice of programming language affects nearly every aspect of a system:

- Design
- Developm
- Safety and S
- Performance

The Obvious

Learning to read and understand new programming langua computing discipline.

Why Write?

You may not implement a general-purpose programming language like C or Haskell in your career.

However..

Every company [illegible] accomplishing some task, often [illegible]

Example

XSLT, Perl scripts for processing text files, CSE's [illegible]

Learn how to make a PL properly and save yourself and your colleagues from headaches.

Why Write?

You may not implement a general-purpose programming language like C or Haskell in your career.

However..

Every company h [illegible] complishing some task, often e

Example

XSLT, Perl scripts for processing text files, CSE's

Learn how to make a PL properly and save yourself and your colleagues from headaches.

Why Write?

You may not implement a general-purpose programming language like C or Haskell in your career.

However..

Every company h [illegible] complishing some task, often e [illegible]

Example

XSLT, Perl scripts for processing text files, CSE's

Learn how to make a PL properly and save yourself and your colleagues from headaches.

Why Reason?

Programming languages are formal languages. Formal specification and proof allows us to:

- Design lang oblins.
- Make lang
- Give a mathe rification of programs. COMP4161, COMP2111, COMP6721
- Develop algorithms to find bugs automatically. COM
- Rigorously analyse optimisations and other program

These tools are also very important for the pursuit of research in programming languages.

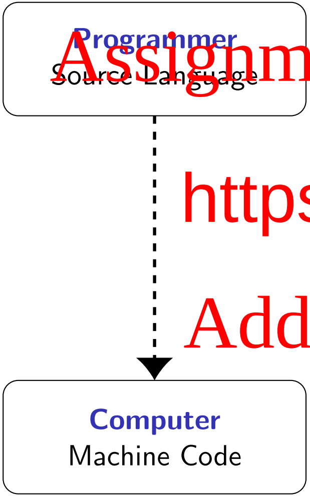
Why Reason?

Programming languages are formal languages. Formal specification and proof allows us to:

- Design lang oblins.
- Make lang
- Give a mathe *rification* of programs. COMP4161, COMP2111, COMP6721
- Develop algorithms to find bugs automatically. COM
- Rigorously analyse optimisations and other program

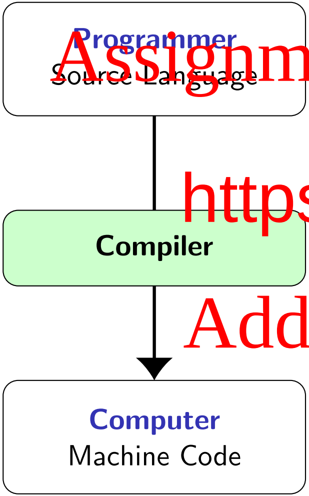
These tools are also very important for the pursuit of research in programming languages.

Bridging the Gap



Computers can’t typically execute source code directly.

Bridging the Gap



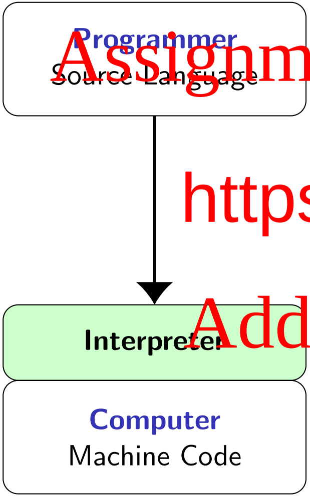
Programmer
Source Language

Compiler

Computer
Machine Code

A compiler translates from source code to a target language, typically machine code.

Bridging the Gap



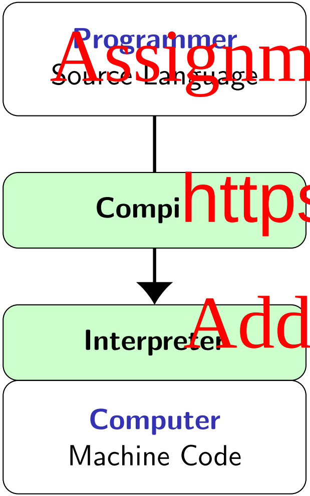
Programmer
Source Language

Interpreter

Computer
Machine Code

An interpreter executes a program as it reads the source code.

Bridging the Gap



Programmer
Source Language

Compiler

Interpreter

Computer
Machine Code

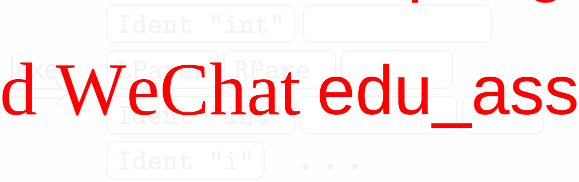
Some languages make use of a hybrid approach. First translating the source language to an intermediate language (abstract or virtual machine), then interpreting that.

Stages of a Compiler

The first stage of a compiler is called a *lexer*, which, given an input string of source code, produces a stream of *tokens* or *lexemes*, discarding irrelevant information like whitespace or comments.

Example (C)

```
int foo () {
  int i;
  i = 11;
  if (i > 5) {
    i = i - 1;
  }
  return i;
}
```



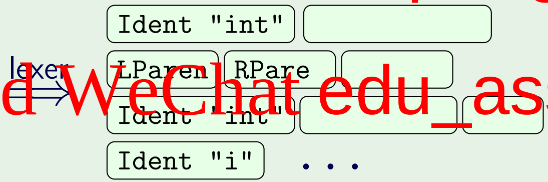
```
Ident "int"
LParen  RParen
Ident "int"
Ident "i"   . . .
```

Stages of a Compiler

The first stage of a compiler is called a *lexer*, which, given an input string of source code, produces a stream of *tokens* or *lexemes*, discarding irrelevant information like whitespace or comments.

Example (C)

```
int foo () {
  int i;
  i = 11;
  if (i > 5) {
    i = i - 1;
  }
  return i;
}
```



lexer

Ident "int"

LParen

RPare

Ident "int"

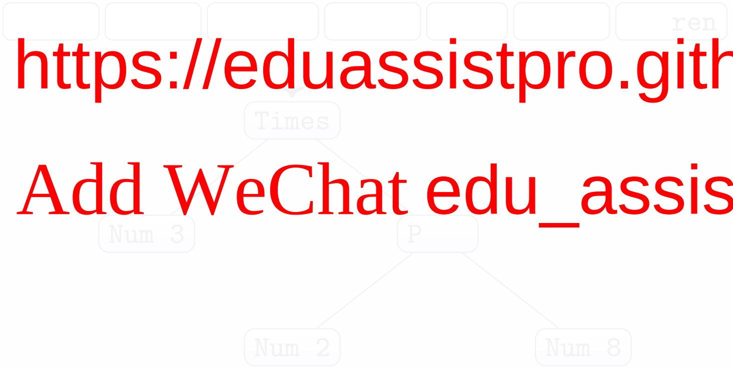
Ident "i"

. . .

Stages of a Compiler

A *parser* converts the stream of tokens from the lexer into a *parse tree* or *abstract syntax tree*:

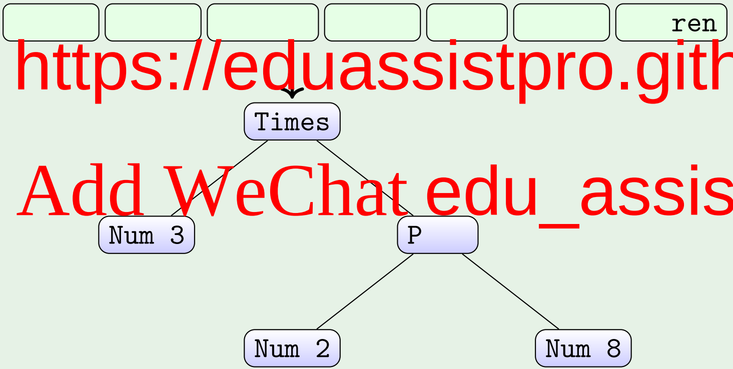
Example (Arithmetic)



Stages of a Compiler

A *parser* converts the stream of tokens from the lexer into a *parse tree* or *abstract syntax tree*:

Example (Arithmetic)



Grammars

The structure of lexemes expected to produce certain parse trees is called a *grammar*.

Example (Informal grammar for C)

C function definiti

- an identifi
- an identifi
- a possibly empty sequence of arguments, enclosed in pa
- a statement (function body)

Conclusions

This kind of definition is way too verbose and too imprecise to specify an implementation.

Grammars

The structure of lexemes expected to produce certain parse trees is called a *grammar*.

Example (Informal grammar for C)

C function definiti

- an identifie
- an identifie
- a possibly empty sequence of arguments, enclosed in pa
- a statement (function body)

Conclusions

This kind of definition is way too verbose and too imprecise to specify an implementation.

Grammars

The structure of lexemes expected to produce certain parse trees is called a *grammar*.

Example (Informal grammar for C)

C function definiti

- an identifie
- an identifie
- a possibly empty sequence of arguments, enclosed in pa
- a statement (function body)

Conclusions

This kind of definition is way too verbose and too imprecise to specify an implementation.

Backus-Naur Form

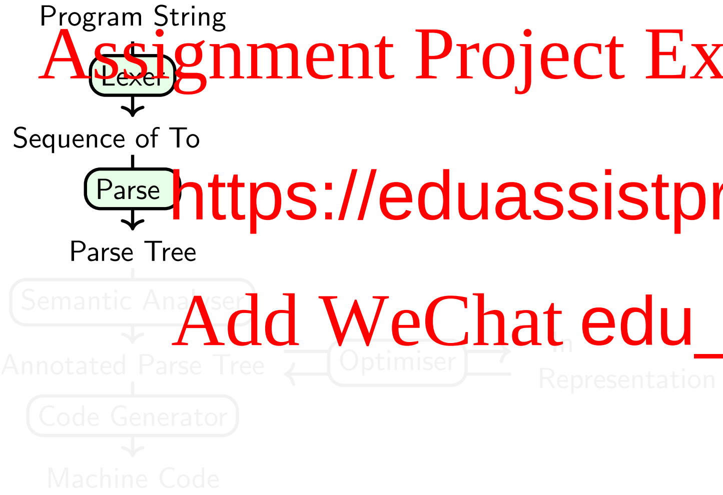
Specify grammatical productions by using a bare-bones recursive notation.

Non-terminals are in *italics* whereas *terminals* are in **this typeface**.

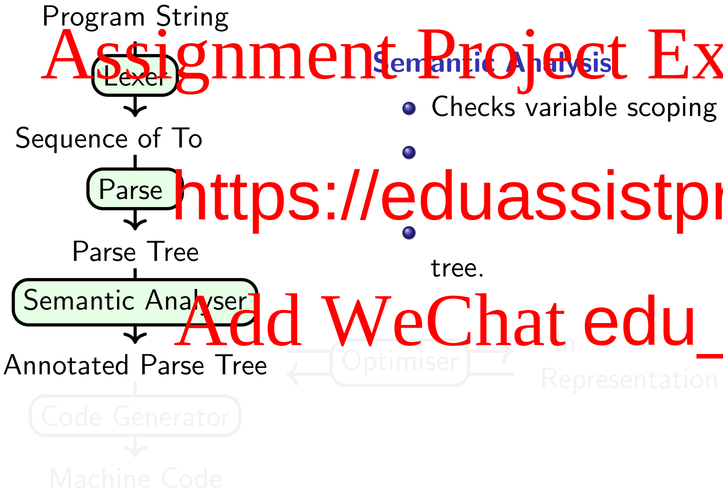
Example (C subset)

$$\begin{array}{lcl}
 & \mid & \textbf{while}\ \textbf{(}\ expr\ \textbf{)} \\
stmts & ::= & \varepsilon \mid stmt\ stmts \\
expr & ::= & Number \mid Ident \\
 & \mid & Ident\ \textbf{=}\ expr \mid \\
locDec & ::= & Ident_1\ Ident_2\ \textbf{;} \\
args & ::= & \varepsilon \mid \cdots
\end{array}$$

Stages of a Compiler



Stages of a Compiler



Program String

Lexer

Sequence of To

Parse

Parse Tree

Semantic Analyser

Annotated Parse Tree

Optimiser

In

Representation

Code Generator

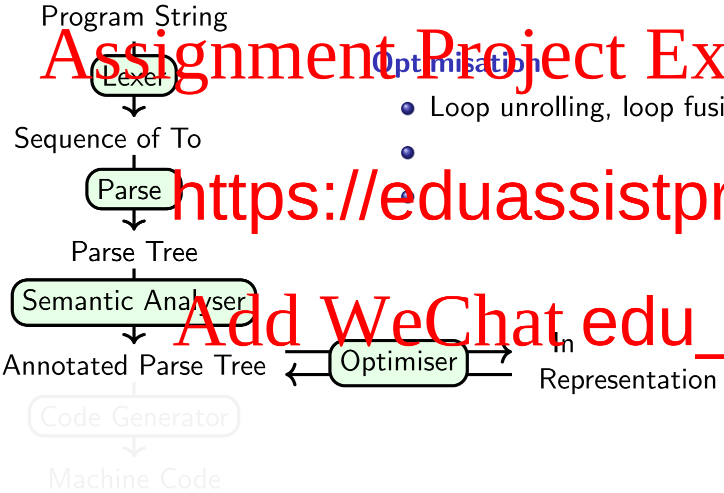
Machine Code

Semantic Analysis

- Checks variable scoping
-
-

tree.

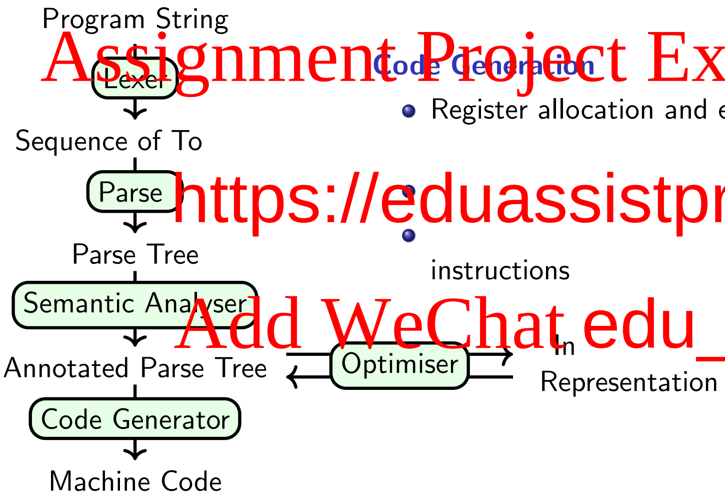
Stages of a Compiler



Optimisation

- Loop unrolling, loop fusion
-
-

Stages of a Compiler



Code Generation

- Register allocation and explicit
-
- instructions