CSC 488S/CSC 2107S Lecture Notes

These lecture notes are provided for the personal use of students taking CSC488H1S or CSC2107HS in the
Winter 2015/2016 term at the University of Toronto

Copying for purposes other than this use and all forms of distribution are expressly prohibited.

CSC488S/CSC2107S - Compilers and Interpreters

Instructor	Prof. Dave Wortman		
Email	dw@cdf.toronto.edu		
Office	Bahen Centre, Room 3520		
Office Hours	immediately after lecture and by appointment		
Lectures	Tuesday	14:00	SS 2106
	Thursday	14:00	SS 2106
Tutorial	Thursday	13:00	SS 2106
Text	Charles Fischer, Ron Cytron and Richard LeBlanc Jr. , Crafting a Compiler , Addison-Wesley 2009		
Marking	Mid term test, Final Exam, Course Project		
Web Page	`http://www.cdf.toronto.edu/~csc488h/winter/`		
Bulletin Board	**Read Often!!** `https://csc.cdf.toronto.edu/csc488h1s`		
Slides	on the Bulletin Board		
Handouts	on the Bulletin Board		

Course Project

- Design and Implementation of a small compiler system for a toy language.
- Work in teams of 5 (- 1 , + 0)
- Five Phase Project

Assignment 1	Write programs in project language
Assignment 2	Revise grammar and build parser
Assignment 3	Implement symbol table and semantic checking.
Assignment 4	Design code generation
Assignment 5	Implement code generator

- Selecting a hard-working, compatible team is important for success in the course project. **All team members are expected to contribute a significant effort to the course project.**
- Teams are expected to use good software engineering practices in all phases of the project. Good quality documentation and thorough testing will be expected.

Course Outline

Topic	Chapters
Compiler structure	Ch. 1, 2
Lexical Analysis	Ch. 3
Syntax Analysis	Ch. 4, 5, 6
Tables & Dictionaries	Ch. 8
Semantic Analysis	Ch. 7, 9
Run-time Environments	Ch. 12
Code generation	Ch. 11, 13
Optimization	Ch. 14

Course Schedule

Event	Marks Weight	Date	Topic
First Lecture		January 12	
Assignment 0 due	0%	January 19	Team formation
Assignment 1 due	2%	January 26	Language Understanding
Assignment 2 due	6%	February 9	Syntax Analysis
Assignment 3 due	12%	March 1	Semantic Analysis Symbol Tables
Midterm Test	20%	March 3	
Assignment 4 due	8%	March 17	Language Implementation
Last Lecture		April 7	
Assignment 5 due	12%	April 8	Code Generation
Final Exam	40%	April 12 – 29	

Reading Assignment

Fischer, Cytron, LeBlanc

Chapter 1

Compiler Technology is Everywhere

- Compiler techniques are used in many places besides compilers
- Anywhere that complicated structured text needs to be processed
  - Command script interpreters, e.g. bash, Perl, Python
  - Document description languages
    e.g. Adobe Postscript, Microsoft Word
  - HTML processing, e.g web browsers, servers
  - Interpreters for JavaScript, Flash
  - User interfaces
  - Query processing
    Twitter uses the ANTLR parser for query processing
    billions of queries per day.
  - Program analysis, e.g. verification, validation
  - Software testing, e.g. test case coverage analysis
  - Program transformation, e.g. the Year 2000 problem

What Do Compilers Do?

Check source program for correctness

Well formed lexically i.e. spell check

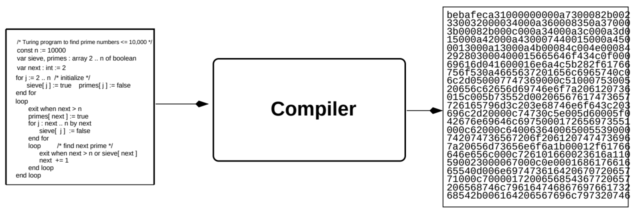
Well formed syntactically. i.e. grammar check

Passes static semantic checks sensibility check

Type correctness

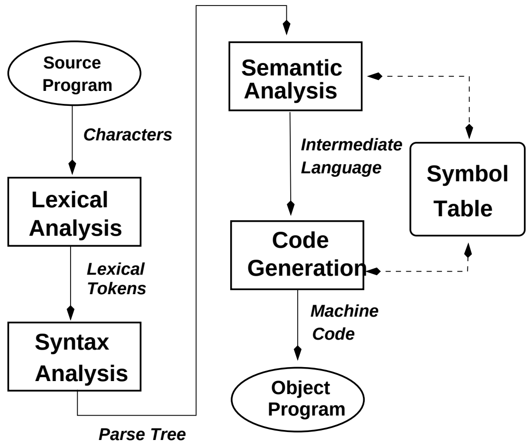
Usage correctness

Transform *source program* into an executable *object program*



Source Program **Executable Program**

Simple Generic Compiler



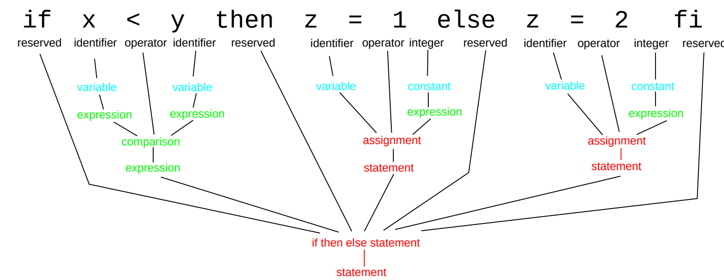
Source statement

`if x < y then z = 1 else z = 2 fi`

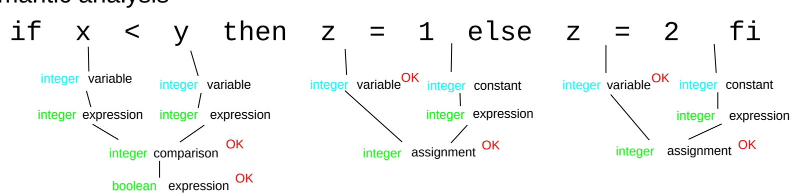
Lexical analysis

if	x	<	y	then	z	=	1	else	z	=	2	fi
reserved	identifier	operator	identifier	reserved	identifier	operator	integer	reserved	identifier	operator	integer	reserved

Syntax analysis



Semantic analysis



Code Generation

`if x < y then z = 1 else z = 2 fi`

```
     load     r1,x        load      r1,=1      L23:  load      r1,=2
     load     r2,y        loadaddr  r2,z             loadaddr  r2,z
     less     r1,r2       store     r2,r1            store     r2,r1
     brfalse  L23         branch    L24        L24:
```


Useful Background for Compiler Implementors

- Computer organization (CSC 258H)
- Software engineering (CSC 207H, CSC 301H, CSC 302H, CSC 410H)
- Software Tools (CSC 209H)
- File and Data structures (CSC 263H/CSC 265H)
- Communication Skills (CSC 290H)
- A large *variety* of programming languages (CSC 324H)
- Some operating systems (CSC 369H)
- Compiler implementation techniques (CSC 488H, ECE 489H).

Compiler Writing Requires Analytic Skills

- The compiler implementor(s) design the mapping from the source language to the target machine.
- Must be able to analyze a programming language for potential problems. Determine if language can be processed during lexical analysis, syntax analysis, semantic analysis and code generation.
- Must be able to analyze target machine and determine best way to implement each construct in the programming language.

Programming Language Designers are (usually) the Enemy

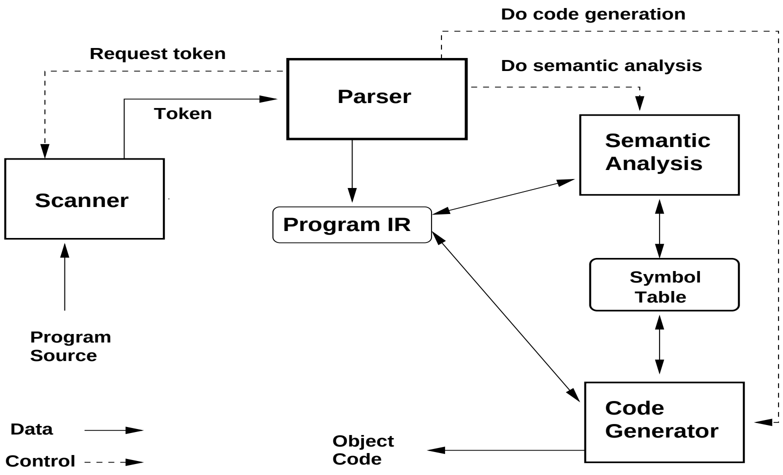
- Most programming language definitions are incomplete, imprecise and sometimes inconsistent. Real programs are written in language dialects.[a]
- Language designers often don't think deeply about the details of the implementation of a language, leaving lots of problems for the compiler writer.
- Typical problems
  - Poor lexical structure. May require extensive buffering or lookahead during lexical analysis
  - Difficulty syntax. Ambiguous, not suitable for normal parsing methods. May require hand written parser, backtracking or lookahead.
  - Incompletely defined or inconsistent semantics.
    User friendly options that are hard to implement.
  - Constructs that are difficult to generate good code for, make optimization difficult, require large run time support

[a] For a discussion of the difficulties of scanning and parsing real programs see `http://cacm.acm.org/magazines/2010/2/69354-a-few-billion-lines-of-code-later/fulltext`

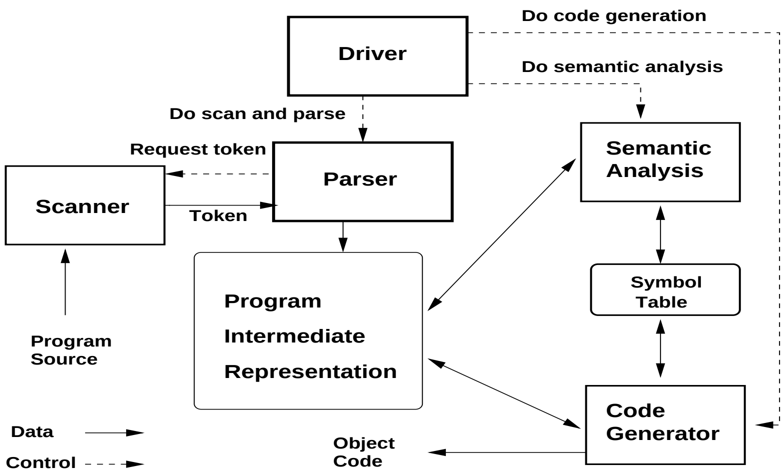
Characteristics of an Ideal Compiler

- User Interface
  - Precise and clear diagnostic messages
  - Easy to use processing options.
- Correctly implements the entire language
- Detects all *statically* detectable errors.
- Generates highly optimal code.
- Compiles quickly using modest system resources.
- Compiler software Engineering
  - Well modularized. Low coupling between modules.
  - Well documented and maintainable.
  - High level of internal consistency checking.
  - Thoroughly tested.

Single Pass Compiler Architecture



Multi Pass Compiler Architecture



Interpretive Systems

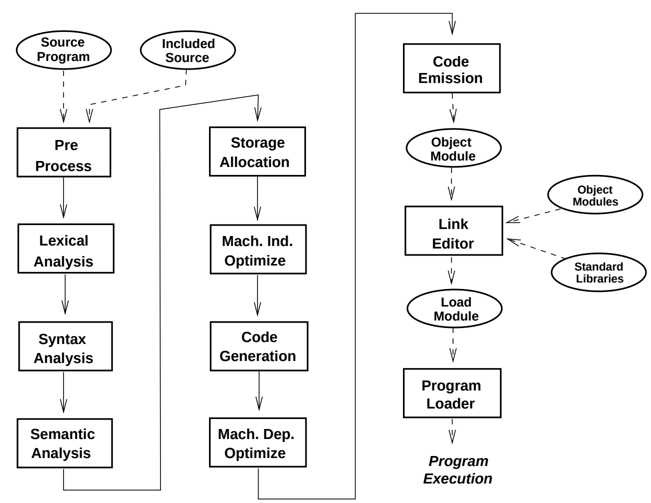
- Compiler generates a pseudo machine code that is a simple encoding of the program.
- The pseudo machine code is executed by another program (an *interpreter*)
- Interpreters are used for
  - Debugging newly written programs.
  - Student compilers that require good run-time error messages.
  - Languages that allow dynamic program modification.
  - Typeless languages that can't be semantically analyzed statically.
  - Cases where run-time size must be minimized.
  - Implementing ugly language features.
  - Quick and dirty compilers.
  - As a way to port programs between environments.
- Interpreters lose on
  - Execution speed, usually significantly slower than machine code.
  - May limit user data space or size of programs.
  - May require recompilation for each run.

Examples of Interpreters

- Pascal P Machine
  - First compiler for Pascal compiled to a pseudo code (P-code) for a language-oriented stack machine.
  - Compiler for Pascal was provided in P-code and source.
  - Porting Pascal to new hardware only required writing a P-code interpreter for the new machine. 1..2 months work.
  - P-code influenced many later pseudo codes including U-code (optimization intermediate language) and Turing internal T-code.
- Java Virtual Machine[a]
  - Java programs are compiled to a *byte-code* for the *Java Virtual Machine* (JVM).
  - JVM designed to make Java portable to many platforms.
  - JVM slow execution speed has lead to the development of *Just In Time* (JIT) native code compilers for Java.

[a] See *Fischer, Cytron, LeBlanc* Section 10.2

The Complete Compilation Process



Project Preview