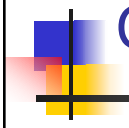


# CS 5641 Compiler Design

Rich Maclin
rmaclin@d.umn.edu
319 Heller Hall

## Acknowledgements

- Notes derived from:
  - Susan Horwitz (UW-Madison)
  - Ras Bodik (UW-Madison)
  - Alex Aiken (Berkeley)
  - George Necula (Berkeley)

# Readings

- Chapter 1
- Chapter 2 (optional) – may want to review this chapter periodically

# Levels of Programming Languages

- Machine language
- Assembly language
- High-level languages
  - C, C++, LISP, Pascal, Prolog, Scheme
- Natural language
  - English

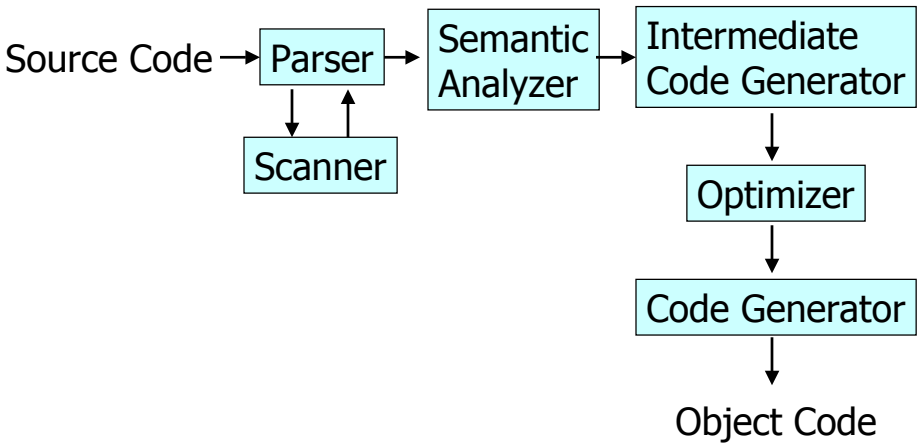
# Programming Paradigms

- Imperative languages
  - Computation as a series of actions
- Object-oriented programming
  - Computation organized around objects and functions that can be applied to objects
- Functional programming
  - Language as a set of (extendable) functions
- Logic programming
  - Programs as defining what a solution look like, letting the machine find a solution

# Tools for Programming

- Interpreter
  - Commands in a high level language are translated to machine terms as they are encountered
- Compiler
  - Program translated in its entirety at one time to a corresponding machine language program
- Hybrids

# Parts of a Compiler




# Scanner

- Translates an input sequence of characters into a sequence of tokens
- Tokens in English: word (junk), capitalized word (Program), period (.)
- Sample input: Dogs like chocolate.
  - Tokens: capitalized word (Dogs)
    word (like)
    word (chocolate)
    period
- Scanners can note illegal characters
- Some scanners also do limit checks on integers

# Program Tokens

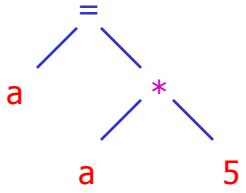
- Sample input:

```
int main () {
    int a = 0;
    cout << a << endl;
    return 1;
}
```

- Tokens:
  - Identifier (int)
  - Identifier (main)
  - Left parenthesis
  - Right parenthesis
  - Left curly brace
  - Identifier (int)
  - Equals
  - Integer (0)
  - Semi-colon
  - Identifier (cout)
  - Double left angle bracket
  - Identifier (a)
  - Double left angle bracket
  - Identifier (endl)
  - Semi-colon
  - Reserved word (return)
  - Integer (1)
  - Semi-colon
  - Right curly brace

# Parser

- Groups tokens together to form grammatical phrases
  - Builds a structure to capture the program (abstract syntax tree)
    - Interior nodes – operators
    - Children - operands
  - Example: a = a * 5;



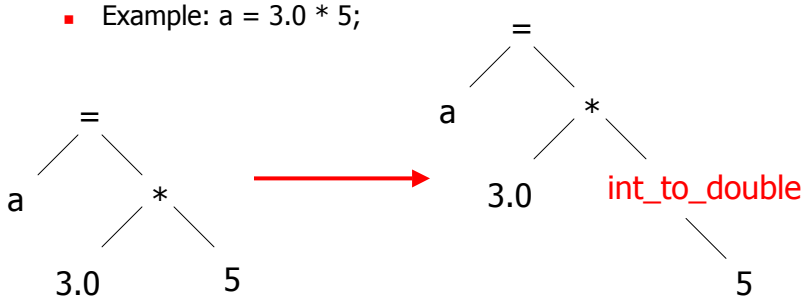
# Parser Errors

- Parsers generally understand programs as a series of statements (think sentences)
- Errors generated when it cannot understand your sentence
- Example: a = * 5;

Something missing!

# Semantic Analyzer

- Checks for non-syntactic errors
  - Example: type errors
- May change or annotate the abstract syntax tree
  - For example, many arithmetic operators apply only to operands of one type, if two compatible types are mixed semantic analyzer may convert
  - Example: a = 3.0 * 5;



# Intermediate Code Generator

- Translates from syntax tree to some intermediate code
  - One possibility – 3-address code, statements with at most 3 operands
  - Example: a = initial + rate * 60;
  - Translation:

```
Temp1 = int_to_double(60)
Temp2 = rate * Temp1
Temp3 = initial + Temp2
a = Temp3
```

# Optimizer

- Improves code generated by intermediate code generator
  - Usually for speed, sometimes for size
- Example (from previous)
  - Initial

```
Temp1 = int_to_double(60)
Temp2 = rate * Temp1
Temp3 = initial + Temp2
a = Temp3
```

  - Improved

```
Temp2 = rate * 60.0
a = initial + Temp2
```

Convert at compile time

No need to store, copy Temp3

# Code Generator

- Generates the object code
- Intermediate instructions are translated into a sequence of target code instructions
- Example:

```
LOADF    rate, R1
MULF     #60.0, R1
LOADF    initial, R2
ADDF     R2, R1
STOREF   R1, a
```