UCF | College of Engineering and Computer Science
UNIVERSITY OF CENTRAL FLORIDA

Compiler Overview

COP-3402 Systems Software

Paul Gazzillo

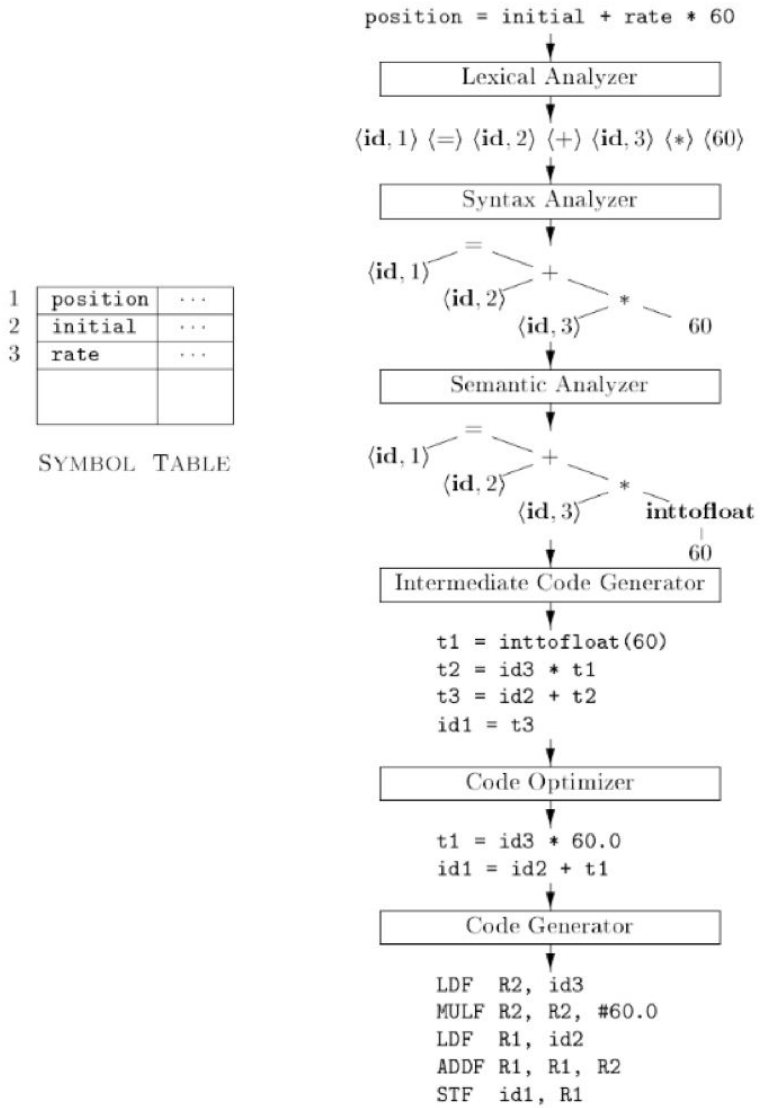
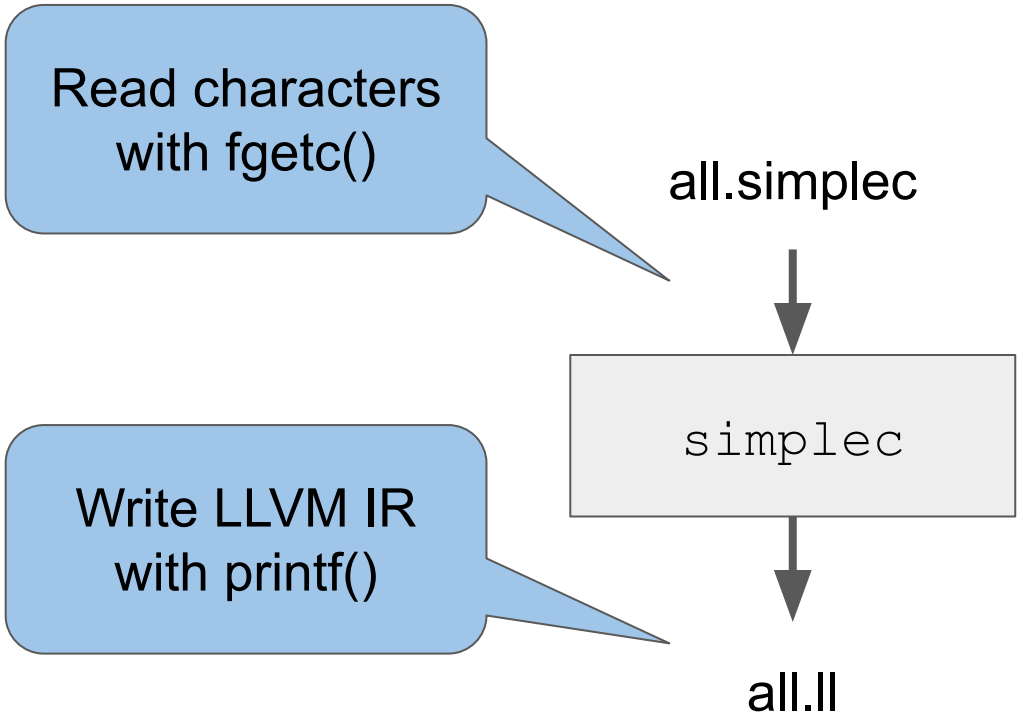


Figure 1.7: Translation of an assignment statement

Your Compiler's Input and Output



Processing the Input File

- Open the input file by name
- Read one character at-a-time (`stdio.h`)
  - `c = fgetc(file);`
- Loop until you reach the end of the input file
  - `if (EOF == c) break;`
- Helper functions for checking characters (`ctype.h`)
  - `isalpha`
  - `isdigit`
  - `ispunct`
- Use a char buffer to save numbers and keywords

Generating the Output File

- Read characters until seeing a full statement
- Print the corresponding LLVM IR with `printf()`
- Suggestion: use format strings as templates
  - `"\%t%d = add nsw i32 %d, %d"`
  - Use printf to fill in the right values from the source file
  - Use your imagination to refine your templates for each project

The Projects Are Each a Complete Compiler

- Each project adds new language constructs
  - arithmetic operations
  - arithmetic expressions
  - variables
  - control-flow structures
  - functions
- For each project we will go over
  - The target language, i.e., the requisite LLVM IR instructions
  - The source language, its syntax and semantics
  - Relevant compiler theory and algorithms

Project 0 Overview

https://github.com/cop3402fall19/syllabus/blob/master/projects/project0.md

Language Specification

- Grammar specification provides the syntax
  - “a program is a list of zero or more statement”
  - “a print statement contains the print keyword, then an expression followed by a semicolon”
- Lexical specification defines the symbols
  - “an identifier is a letter followed by zero or more letters or digits”
  - “the PRINT keyword is recognized by matching the ‘print’ string”

There Are Two Statement Patterns

- Printing a constant

```
print 5;
```

- Printing the result of one operation

```
print -5 - 7;
```

- The other four statements just use different operators

The Minus Sign Does Double Duty

- The minus sign is both
  - the subtraction operation, and
  - part of a negative number.
- How can your compiler tell the difference?

Keywords and Identifiers Look the Same

- Identifiers are any sequence of letters (digits as well)
- Keywords, e.g., “print”, are also sequences of letters
- How can our compiler tell the difference?

Architecture Suggestions

- Use one function for each
  - *token*, or symbol, of the language, e.g., IDENTIFIER or SEMI
  - *syntactic unit*, e.g., expression or statement
- Can (should) we merge any functions?
- How can we compose syntactic units?
  - E.g., a statement depends on result an expression.
  - If each is a function, how do we pass info between them?
- Emit LLVM IR after recognizing a syntactic unit

Use the Given template.ll as Boilerplate

- Your compiler should emit this boilerplate
  - Hardcoding it as strings (one for before, one for after) is fine
- Emit your generated LLVM IR inside the main method
  - Indicated with a comment in template.ll
- We will learn more about theboilerplate in project 4

LLVM Intermediate Representation (IR)

Constants

- Integer constants always give the type

```
i32 -43
```

- `i32` means a 32-bit (signed) integer
- `-43` is the decimal
- We will only use i32 values for our compiler

Variables

- Variable names are prefixed with %, e.g., `%t1`
  - Global values are prefixed with `@`
- Assignments uses an equals sign

```
%t1 = add nsw i32 2, 3
```

- *Important:* **variables can only be assigned once**
  - This is static single-assignment (SSA) form
  - This makes program analysis and optimization easier
- Generate variable names by incrementing a counter

Arithmetic Instructions

- Instructions can happen in assignments

```
%t1 = add nsw i32 2, 3
```

- `nsw` is for tracking undefined behavior from overflows
  - Just treat “add nsw” as the addition operation
- The operation then takes the type and two operands
- Operands may also be variables

```
%t2 = add nsw i32 %1, 3
```

- Other instructions: `sub nsw`, `mul nsw`, `sdiv`, `srem`

Printing

- The given template.ll file has a `print_integer` function
  - It calls printf and clang links with the C standard library
- Use a function with the call instruction

```
call void @print_integer(i32 %t1)
```

- Each argument needs its type
- Arguments can be constants or variables

Demo: LLVM IR for all.simplec

Conclusion

- The compiler reads each character of the source file
- The compiler prints (emits) equivalent target language
  - We will use the LLVM IR
- Project 0 is a complete but very simple compiler
  - Only single arithmetic operations
- Each project adds new language constructs
  - Expressions, variables, control-flow, and functions
- LLVM IR provides instructions for arithmetic