CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

http://www.cse.buffalo.edu/faculty/alphonce/SP17/CSE443/index.php
https://piazza.com/class/iybn4ndqa1s3ei
BUILD
A
COMPILER!
# Learning outcomes

<table>
<thead>
<tr>
<th>Learning outcome</th>
<th>Instructional methods</th>
<th>Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify and describe the function of the major phases of a compiler.</td>
<td>Lecture-based instruction</td>
<td>PRE, HW, EX</td>
</tr>
<tr>
<td>Define formally the grammars used in the front end of a compiler, their application in the front end, and techniques for parsing such grammars.</td>
<td>Hands-on activities in lecture and recitation</td>
<td></td>
</tr>
<tr>
<td>Evaluate (compare and contrast) different intermediate representations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explain the compiler’s role in creating and managing run-time environments.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluate and evaluate (compare and contrast) different approaches to code generation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identify and explain the applicability and operation of code optimizations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build both the front and back ends of a compiler.</td>
<td></td>
<td>PROJ</td>
</tr>
</tbody>
</table>
Assessment

- Homework - 5-7 assignments
- Project - 5-7 phases
- Presentation - 1-2 per team
- Examination - 3 hour final, based on homework/project
<table>
<thead>
<tr>
<th></th>
<th>INDIVIDUAL</th>
<th>TEAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework</td>
<td>30%</td>
<td>Project</td>
</tr>
<tr>
<td>Exam</td>
<td>20%</td>
<td>Presentation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Teams & Recitations

- Form teams this week - I recommend teams of size 3-4.

- Recitations start this week.

- Recommend all team members attend same recitation, but not required (you can attend either recitation).

- For project, you may choose either C or SML - decide with your teammates (you must have a unanimous decision). We will discuss more in recitation this week.
Goal: build a compiler

source program

executable
Phases of a compiler

source program

executable

Figure 1.6, page 5 of text
Why?

- Deeper understanding of languages
- Become a better programmer
- Learn how to build tools
- Build special-purpose languages (DSLs)
- Theory meets practice
- High-level meets low-level
Deep understanding - ex 1

name

vs

identifier

vs

variable
void foo() {
    int x = 0;
    printf(x);
}

Deep understanding - ex 1
int func(int x) {
    if (x == 0) { return 1; }
    else { return x * func(x-1); }
}

Deep understanding - ex 1
Deep understanding – ex 1

```c
struct Pair {
    int x;
    int y;
};

void bar() {
    Pair r, s;
}
```
name
y, x
identifier

refers to

variable
location in memory
variables in distinct scopes,
variables in distinct records/objects, or
variables in distinct function invocations
Deep understanding - ex 2

order of evaluation
Deep understanding - ex 2

\[ a + b \times c; \]
Deep understanding - ex 2

\[ a + b \times c; \]

\[ f() + g() \times h(); \]
Deep understanding - ex 2

\[ a + b \times c; \]

\[ f() + g() \times h(); \]

\[ f() + f() \times f(); \]
Languages: the Chomsky hierarchy

"On Certain Formal Properties of Grammars"

recursively enumerable
context-sensitive
context-free
regular

https://upload.wikimedia.org/wikipedia/commons/8/86/Noam_chomsky.jpg
grammars (generators) and languages

automata (acceptors)

recursively-enumerable language

Turing machine

context-sensitive language

linear-bounded automaton

context-free language

push-down automaton

regular language

finite-state automaton

the traditional Chomsky hierarchy


license: http://creativecommons.org/licenses/by/3.0/
grammars (generators) and languages

automata (acceptors)

recursively-enumerable language

Turing machine

context-sensitive language

linear-boundary automaton

context-free language

push-down automaton

regular finite-state language

finite-state automaton

the traditional Chomsky hierarchy

Syntactic structure

Lexical structure
Phases of a compiler

Figure 1.6, page 5 of text
Lexical Structure

int main(){
Lexical Structure

```c
int main()
{
    character stream
}
```
Lexical Structure

character stream → token stream

```c
int main() {
    id("int") id("main") LPAR RPAR LBRACE
}```
Lexical Structure

tokens

- keywords (e.g. static, for, while, struct)
- operators (e.g. <, >, <=, =, ==, +, -, &, .)
- identifiers (e.g. foo, bar, sum, mystery)
- literals (e.g. -17, 34.52E-45, true, ‘e’, “Serenity”)
- punctuation (e.g. {, }, , (, ), , ; )
meta vs object language

- **object language**: the language we are describing
- **meta language**: the language we use to describe the object language
meta vs object language

- use quotes (meta vs ‘object’)
- punctuation (e.g. ‘{’, ‘}’, ‘(‘, ‘)’, ‘;’)

- use font or font property (meta vs object)
- punctuation (e.g. {, }, (, ), ;)
languages & grammars

Formally, a **language** is a set of strings over some alphabet

**Ex.** \{00, 01, 10, 11\} is the set of all strings of length 2 over the alphabet \{0, 1\}

**Ex.** \{00, 11\} is the set of all even parity strings of length 2 over the alphabet \{0, 1\}
Formally, a grammar is defined by 4 items:

1. $N$, a set of non-terminals
2. $\Sigma$, a set of terminals
3. $P$, a set of productions
4. $S$, a start symbol

$G = (N, \Sigma, P, S)$
languages & grammars

\( N \), a set of non-terminals
\( \Sigma \), a set of terminals (alphabet)

\( N \cap \Sigma = {} \)

\( P \), a set of productions of the form (right linear)

\( X \rightarrow a \)
\( X \rightarrow aY \)
\( X \rightarrow \varepsilon \)

\( X \in N, \ Y \in N, \ a \in \Sigma, \ \varepsilon \) denotes the empty string

\( S \), a start symbol

\( S \in N \)
languages & grammars

Given a string \( \alpha A \), where \( \alpha \in \Sigma^* \) and \( A \in N \), and a production
\[ A \rightarrow \beta \in \mathcal{P} \]
we write \( \alpha A \Rightarrow \alpha \beta \) to indicate that \( \alpha A \) derives \( \alpha \beta \) in one step.

\( \Rightarrow^k \) and \( \Rightarrow^* \) can be used to indicate \( k \) or arbitrarily many derivation steps, respectively.
Languages & grammars

$L(G)$ is the set of all strings derivable from $G$ starting with the start symbol; i.e. it denotes the language of $G$. 
Given a grammar $G$ the language it generates, $L(G)$, is unique.

Given a language $L$ there are many grammars $H$ such that $L(H) = L$. 

languages & grammars