CSE443 Compilers

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

- Academic Integrity (esp wrt teamwork)
- Values Statement
Textbook

Classic text.

You should hang on to this one.
Team formation

- If you have a team, please list members in Piazza post.
What determines program semantics?

```c
#include <stdio.h>

int main() {
    int i = 0;
    int sum = 0;
    while (i <= 10) {
        sum = sum + i;
        printf("sum of integers from 0 to %d is %d.\n",i,sum);
        i = i + 1;
    }
}
```
What is this?

#include <stdio.h>

int main() {
    int i = 0;
    int sum = 0;
    while (i <= 10) {
        sum = sum + i;
        printf("sum of integers from 0 to %d is %d.\n", i, sum);
        i = i + 1;
    }
}
What is this?

```c
#include <stdio.h>

int main() {
  int i = 0;
  int sum = 0;
  while (i <= 10) {
    sum = sum + i;
    printf("sum of integers from 0 to %d is %d.\n",i,sum);
    i = i + 1;
  }
}
```
/* La suite de Syracuse est définie ainsi :
- on part d'un entier ;
- s'il est pair, on le divise par 2 ;
- sinon, on le multiplie par 3 et on ajoute 1 ;
- on recommence la même opération sur l'entier obtenu, et ainsi de suite ;
- la suite s'arrête si on arrive à 1. */

syracuse :
durée est un nombre
e est un nombre
début
e prend 14
tant que e != 1 lis
durée prend durée + 1
si (e mod 2) = 0, e prend e / 2
sinon e prend e * 3 + 1
affiche e
ferme
affiche "durée = {durée}"
The Syracuse sequence is defined as follows:
- it starts with any natural number > 0
- if it is even, we divide by 2
- else we multiply by 3 and add 1
- the process is repeated on the result
- the process ends when the result is 1

```c
void syracuse() {
    int iterations;
    int e;

    iterations = 0;
    e = 14;
    while (e != 1) {
        iterations = iterations + 1;
        if ( (e % 2) == 0 ) e = e / 2;
        else e = e * 3 + 1;
        printf("%d\n",e);
    }
    printf("iterations = %d\n",iterations);
}
```
void syracuse() {
    int iterations = 0;
    int e = 14;
    while (e != 1) {
        iterations = iterations + 1;
        if ( (e % 2) == 0 ) e = e / 2;
        else e = e * 3 + 1;
        printf("%d
",e);
    }
    printf("iterations = %d\n",iterations);
}

int main() {
    syracuse();
    return 0;
}
void syracuse() {
    int iterations = 0;
    int e;
    e = 14;
    while (e != 1) {
        iterations = iterations + 1;
        if ( (e % 2) == 0 ) e = e / 2;
        else e = e * 3 + 1;
        printf("%d\n",e);
    }
    printf("iterations = %d\n",iterations);
}
Syntax and semantics

- Syntax: program structure
- Semantics: program meaning
- Semantics are determined (in part) by program structure.
Languages: the Chomsky hierarchy
"On Certain Formal Properties of Grammars" published 1959

- 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/
Phases of a compiler

Figure 1.6, page 5 of text
Phases of a compiler

Figure 1.6, page 5 of text
Lexical Structure

int main(){
Lexical Structure

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

`int main()`

character stream $\rightarrow$ token stream

`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. {, }, , (, ), , ; )
Describing lexical structure

- We need some formal way of describing the lexical structure of a language.
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

- How do we distinguish between the two?
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

\[ G = (N, \Sigma, P, S) \]
Lexical analysis: a bird's eye view

Language: a set of strings

Grammar: rules for generating language

Regular expression: a form of grammar

Finite automaton: a machine for language

C program: generated by FLEX

{ for, while, x, factorial, ... }

\[ G = (N, \Sigma, P, S) \]
Formally, a grammar $G = (N, \Sigma, P, S)$ is defined by 4 items:

1. $N$, a set of non-terminals
   \[ N = \{ X, Y \} \]

2. $\Sigma$, a set of terminals (alphabet)
   \[ \Sigma = \{ a \} \quad \text{<- for example} \]
   \[ N \cap \Sigma = \{\} \quad \text{<- general grammar constraints} \]

3. $P$, a set of productions of the form (right linear)
   \[ X \rightarrow aY \]
   \[ Y \rightarrow bX \]
   \[ Y \rightarrow a \quad \text{<- a right linear grammar describing a regular language} \]
   \[ X \rightarrow \varepsilon \]
   \[ X \in N, Y \in N, a \in \Sigma, \varepsilon \text{ denotes the empty string} \]

4. $S$, a start symbol
   \[ S = Y \]
   \[ S \in N \]
In computer science, a linear grammar is a context-free grammar that has at most one nonterminal in the right hand side of each of its productions.

Two special types of linear grammars are the following:
- the left-linear or left regular grammars, in which all nonterminals in right hand sides are at the left ends;
- the right-linear or right regular grammars, in which all nonterminals in right hand sides are at the right ends.

https://en.wikipedia.org/wiki/Linear_grammar
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.