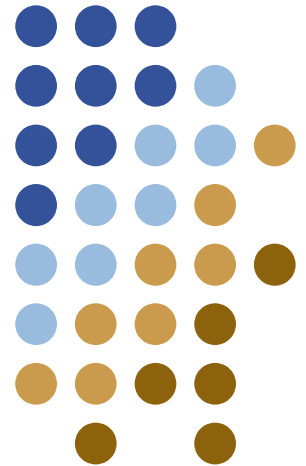


Compilers (and Program Analysis)

Yannis Smaragdakis, U. Athens





**I'd rather
write programs that write programs
than write programs**





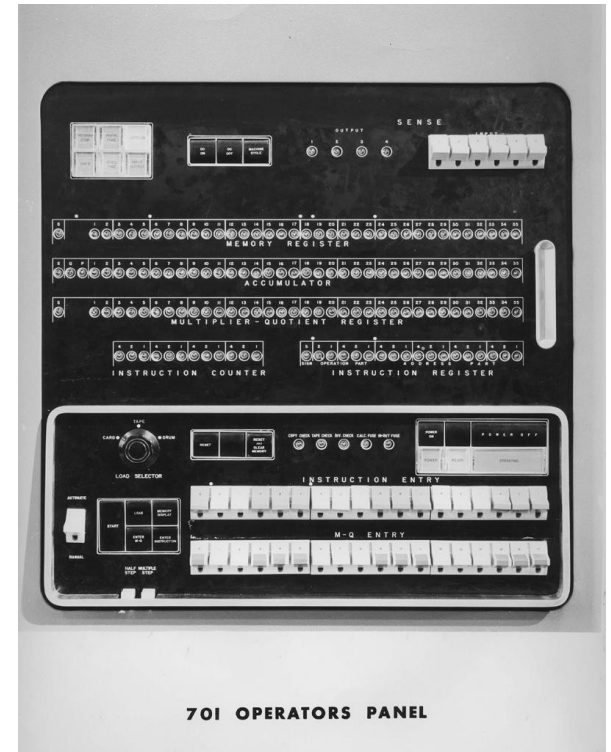
Discussion

- What does a compiler do?
- Why do you need that?
- Name some compilers you have used



A Brief History of High-Level Languages

- 1953 IBM develops the 701
 - Memory: 4096 words of 36 bits
 - Speed: 60 msec for addition
 - All programming done in assembly code





Programming

- **What's the problem?**
 - Assembly programming very slow and error-prone
 - Software costs exceeded hardware costs!
- John Backus: “*Speedcoding*”
 - Simulate a more convenient machine
 - But, ran 10-20 times slower than hand-written assembly
- Backus
 - **Idea**: translate high-level code to assembly
 - Many thought this impossible
 - Had already failed in other projects*
- 1954-7 FORTRAN I project
 - By 1958, >50% of all code is in FORTRAN
 - Cut development time dramatically – *from weeks to hours*





FORTRAN I

- The first compiler
 - Huge impact on computer science
 - Produced code almost as good as hand-written
- Led to an enormous body of work
 - Theoretical work on languages, compilers
 - Program semantics
 - Thousands of new languages
- Modern compilers preserve the outlines of FORTRAN I





Language implementations

- Two major strategies:
 - Interpretation
 - Compilation
- What are the main differences?
 - “**Online**”: read program, execute immediately
 - “**Offline**”: convert high-level program into assembly code
- Compilation is a language translation problem
 - What are the languages?

Can you think of another strategy – a “hybrid”?

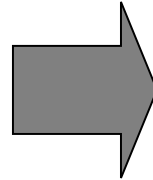


Languages involved



```
int i = 10;
while (i > 0) {
    x = x * 2;
    i = i - 1;
}
```

Source



```
movl    %esp, %ebp
subl    $4, %esp
movl    $10, -4(%ebp)
.L2:
cmpl    $0, -4(%ebp)
jle     .L3
movl    8(%ebp), %eax
sall    %eax
movl    %eax, 8(%ebp)
leal    -4(%ebp), %eax
decl    (%eax)
jmp     .L2
.L3:
movl    8(%ebp), %eax
```

Target





The compilation problem

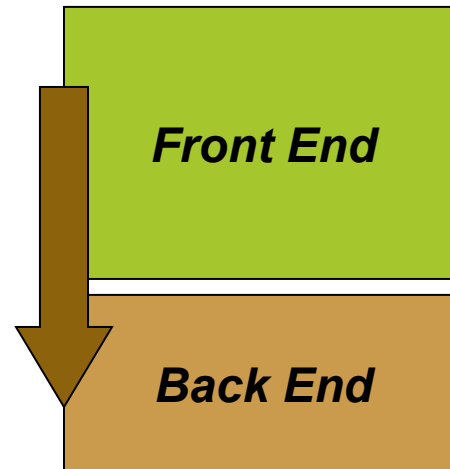
- Assembly language
 - Converts trivially into machine code
 - No abstraction: load, store, add, jump, etc.
 - Extremely painful to program
 - What are other problems with assembly programming?
- High-level language
 - Easy to understand and maintain
 - Abstractions: control (loops, branches); data (variables, records, arrays); procedures
 - **Problem**: how do we get from one to the other?
(*systematically*)





Structure of a compiler

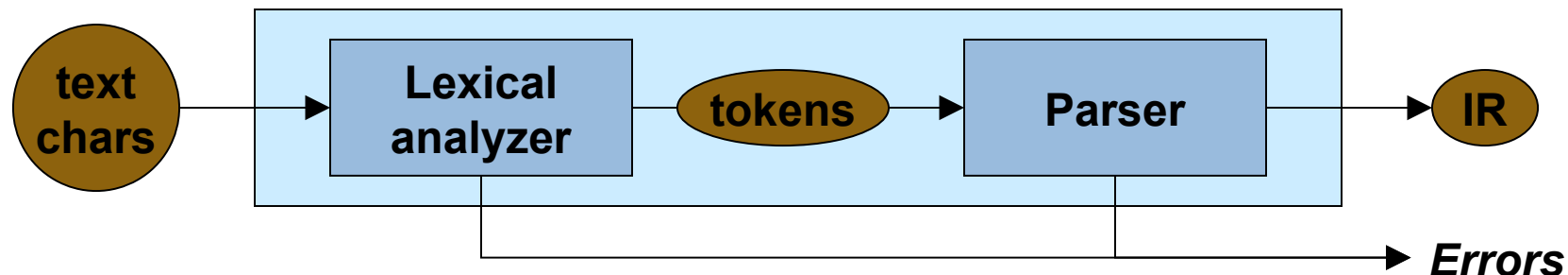
- Organized as a series of passes
 - Lexical Analysis
 - Parsing
 - Semantic Analysis
 - Optimization
 - Code Generation



- We will follow this outline in the class



The front end



Responsibilities?

Recognize legal (and illegal programs)

Report errors in a useful way

Generate internal representation

How it works

Good news: linear time, mostly generated automatically

By analogy to natural languages...





Lexical Analysis

First step: recognize words.

Smallest unit above letters

This is a sentence.

Some lexical rules

Capital “T” (start of sentence symbol)

Blank “ ” (word separator)

Period “.” (end of sentence symbol)





Lexical analysis

Another example:

```
void func(float * ptr, float val)
{
    float result;
    result = val/*ptr;
}
```

Why is this case interesting?

“/*” is the comment delimiter



Parsing



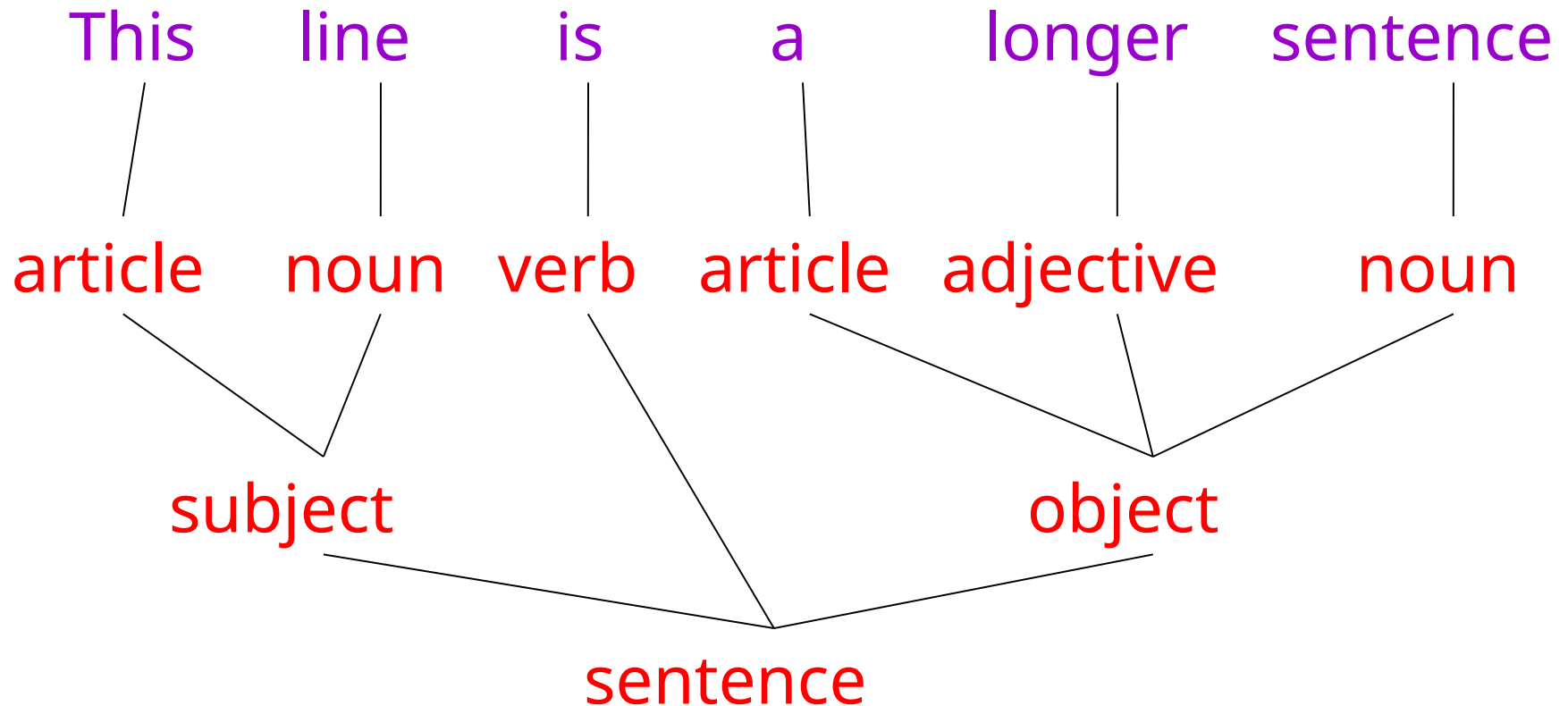
Once words are understood, the next step is to understand sentence structure

Parsing = Diagramming Sentences

The diagram is a tree...



Diagramming a Sentence





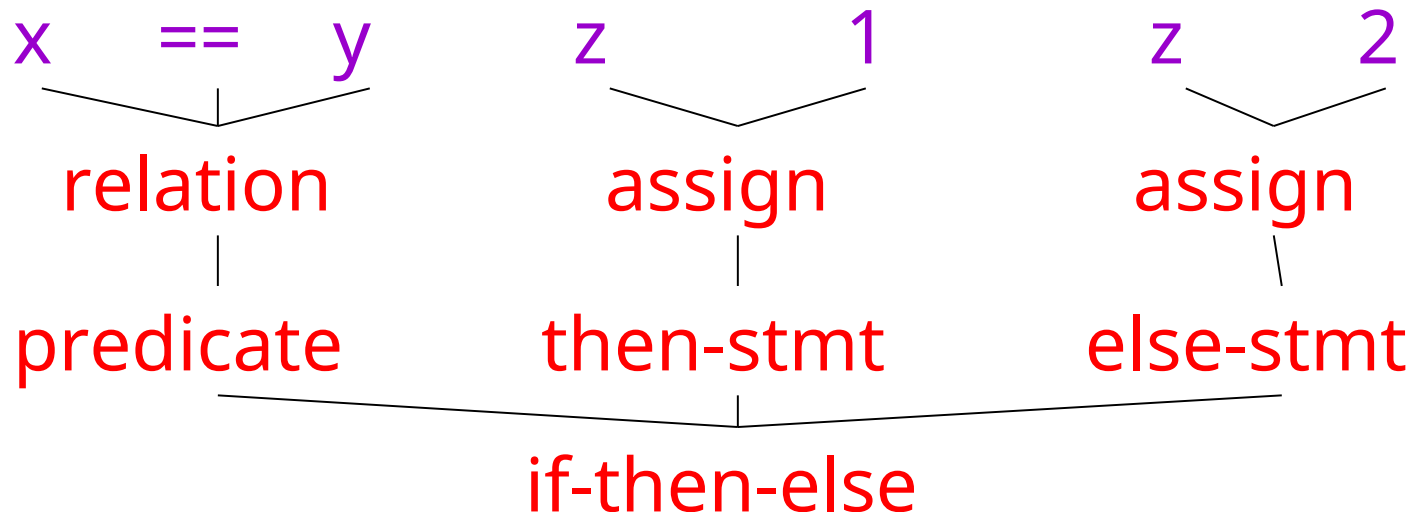
Diagramming programs

Diagramming program expressions is the same

Consider:

If $x == y$ then $z = 1$; else $z = 2$;

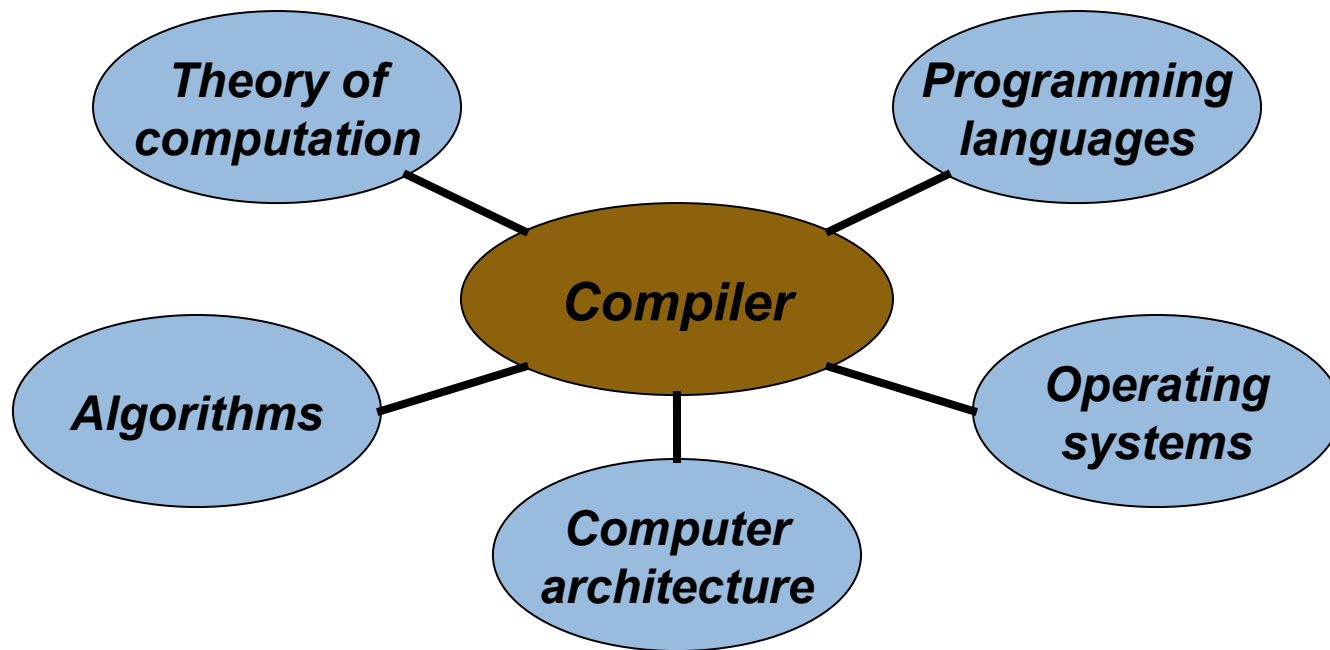
Diagrammed:





Study of compilers

- Brings together many parts of CS
 - Practical and theoretical
 - Some solved problems, others unsolved



AI Policy



The class projects are AI-free

You have to do everything yourselves, from scratch

*There is a difference between doing
because the **outcome** is valuable vs.
because the **(learning) process** is valuable*



Artificial Intelligence



We live in unprecedented times

A mere 5 years ago, if you were to ask experts when human-level intelligence would be achieved by machines, many would say “never”

- and if you were to tell them that this intelligence would arise from a language model, they’d all laugh

Today, it’s commonplace



Artificial Intelligence Superlatives



Such an existential shock to humanity has not appeared in history (the closest comparable being the atomic bomb)

The only future discoveries that can possibly compare are biological immortality or contact with intelligent alien life





AI and You

But no singularity has appeared (yet)

- LLMs seem to be plateau-ing

Many Software Engineering tasks are already better done by machines

But: Prototyping is trivial, debugging and getting things fully right is very hard

The best users of AI are people who know how to do things themselves



AI and You



And now ... *you* take the baton!

Your generation will take this technological development, evolve it, and apply it

