



Compilers

Structure of a Compiler

1. Lexical Analysis
2. Parsing
3. Semantic Analysis
4. Optimization
5. Code Generation

- First step: recognize words.
 - Smallest unit above letters

This is a sentence.

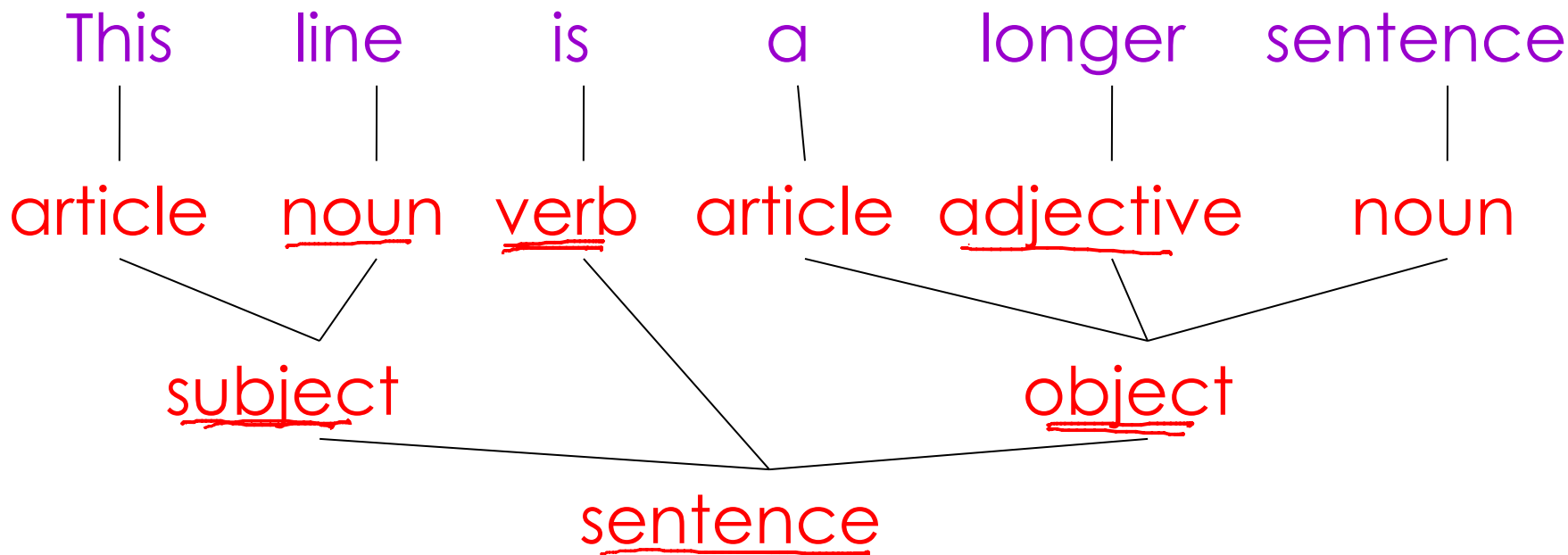
A diagram illustrating the process of recognizing words in a sentence. The sentence "This is a sentence." is written in purple. Red arrows point down to the start of each word: "This", "is", "a", "sentence.", and a final arrow at the end of the sentence. Below each word, there is a red underline, indicating that these are the smallest units above individual letters.

ist his ase nte nce

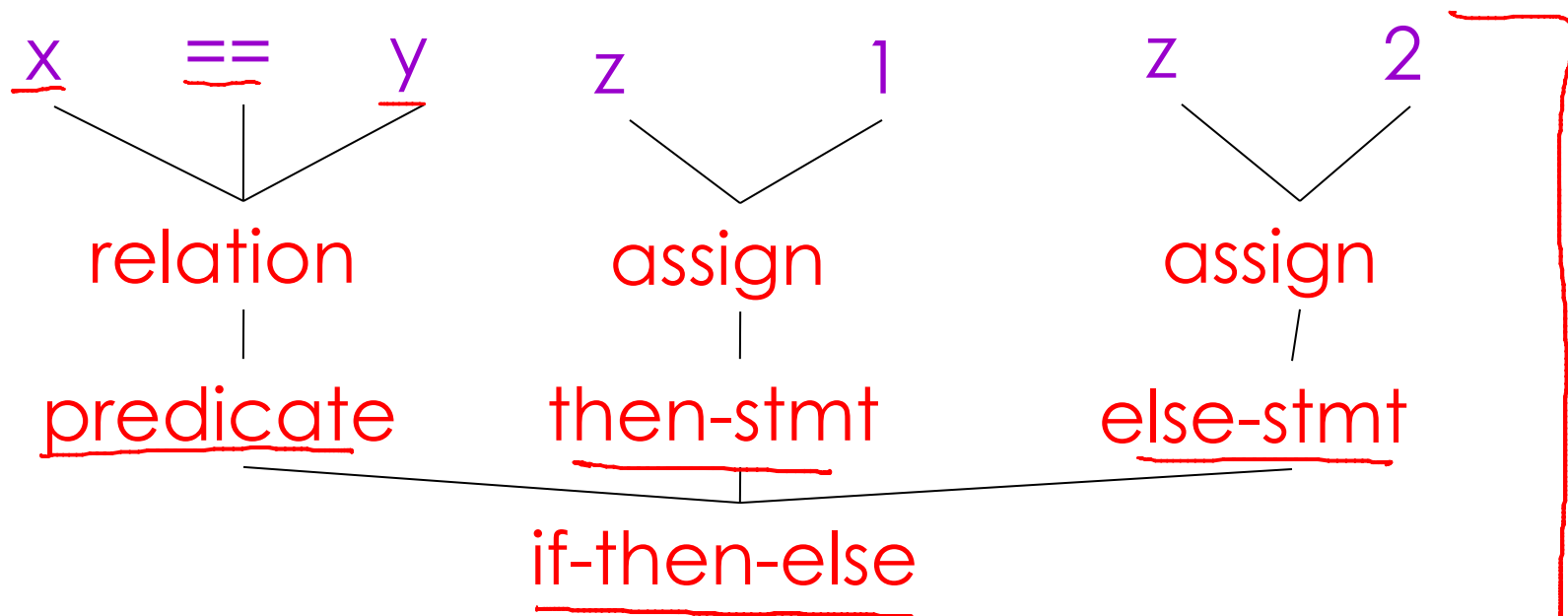
- Lexical analysis divides program text into “words” or “tokens”

if x == y then z = 1; else z = 2;

- Once words are understood, the next step is to understand sentence structure
- Parsing = Diagramming Sentences
 - The diagram is a tree



if x == y then z = 1; else z = 2;



- Once sentence structure is understood, we can try to understand “meaning”
 - This is ^{too} hard!
- Compilers perform limited semantic analysis to catch inconsistencies

- Example:

Jack said Jerry left his assignment at home.

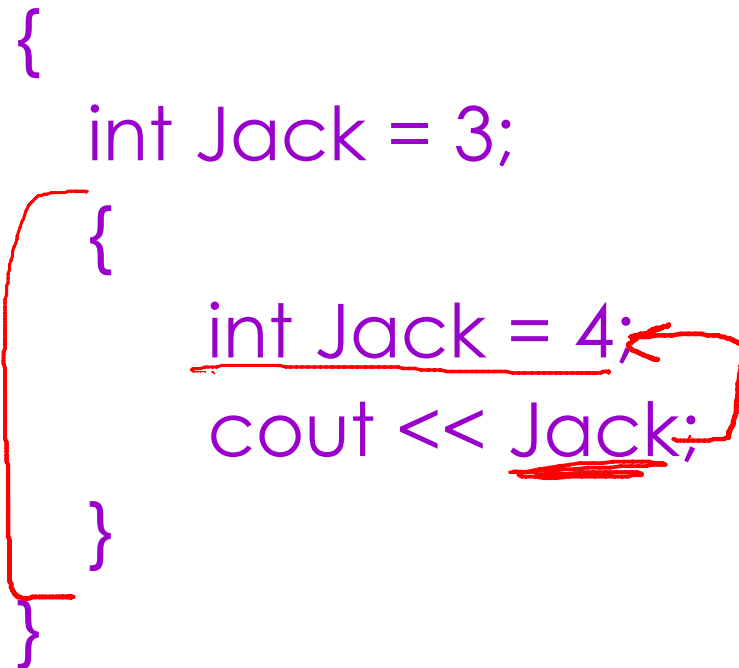


- Even worse:

Jack said Jack left his assignment at home?

- Programming languages define strict rules to avoid such ambiguities

```
{  
    int Jack = 3;  
    {  
        int Jack = 4;  
        cout << Jack;  
    }  
}
```



The diagram illustrates variable shadowing in C++. It shows a block containing a variable 'Jack' with value 3. Inside this block is a nested block that also declares a variable 'Jack' with value 4. A red bracket on the left groups the two blocks. A red arrow points from the 'Jack' in the inner block to the 'Jack' in the outer block, highlighting that the inner declaration shadows the outer one. The 'Jack' in the output statement is underlined in red.

- Compilers perform many semantic checks besides variable bindings

- Example:

Jack left her homework at home.

- A “type mismatch” between her and Jack; we know they are different people

- Optimization has no strong counterpart in English
 - ^{akin to} ~~But a little bit like~~ editing
- Automatically modify programs so that they
 - Run faster
 - Use less memory
 - Power
 - Network
 - Database

$X = Y * 0$ is the same as $X = 0$

NO!

$\text{NaN} \neq 0 = \text{NaN}$

valid for integers
invalid for FP

Code Gen

- Produces assembly code (usually)
- A translation into another language
 - Analogous to human translation

- The overall structure of almost every compiler adheres to our outline
- The proportions have changed since FORTRAN

