Compilers

module of the course “Professional English”

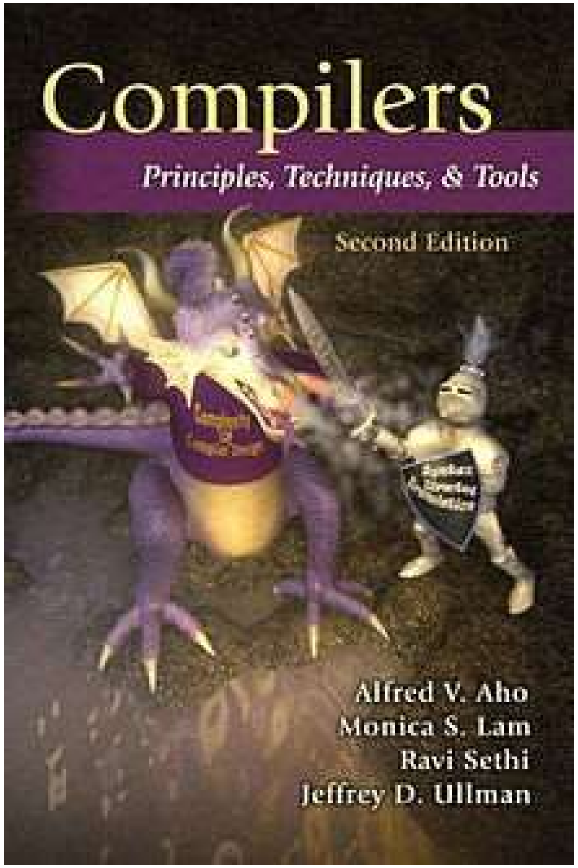
Yulia Burkatovskaya

Department of Computer Engineering

Associate professor

Dragon book

- *Compilers: Principles, Techniques, and Tools, Second Edition*
- (2006, the "Purple Dragon Book"),
- by Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman and Monica S. Lam.



1. Introduction

- Language processors
- The structure of a compiler

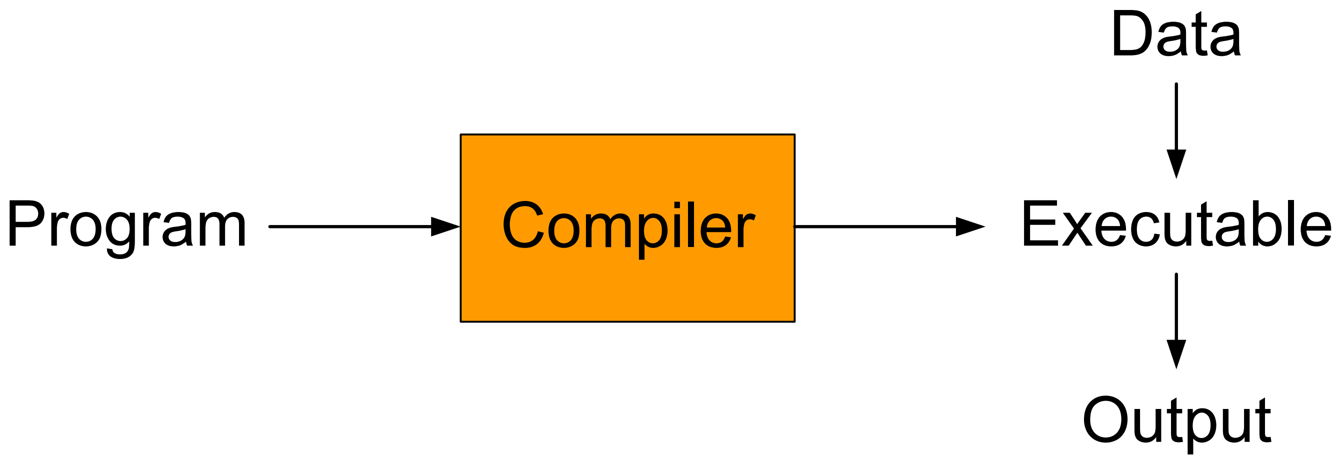
1.1. Language processors

What did he say?

How to translate it to him?

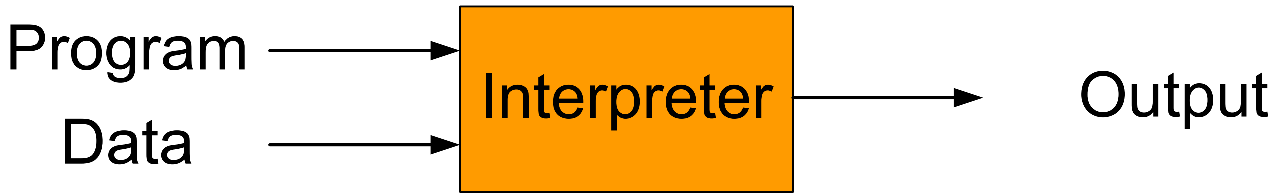
Language processors

A **compiler** is a program that can read a program in one language — the *source* language — and translate t into an equivalent program in another language — the *target* language.

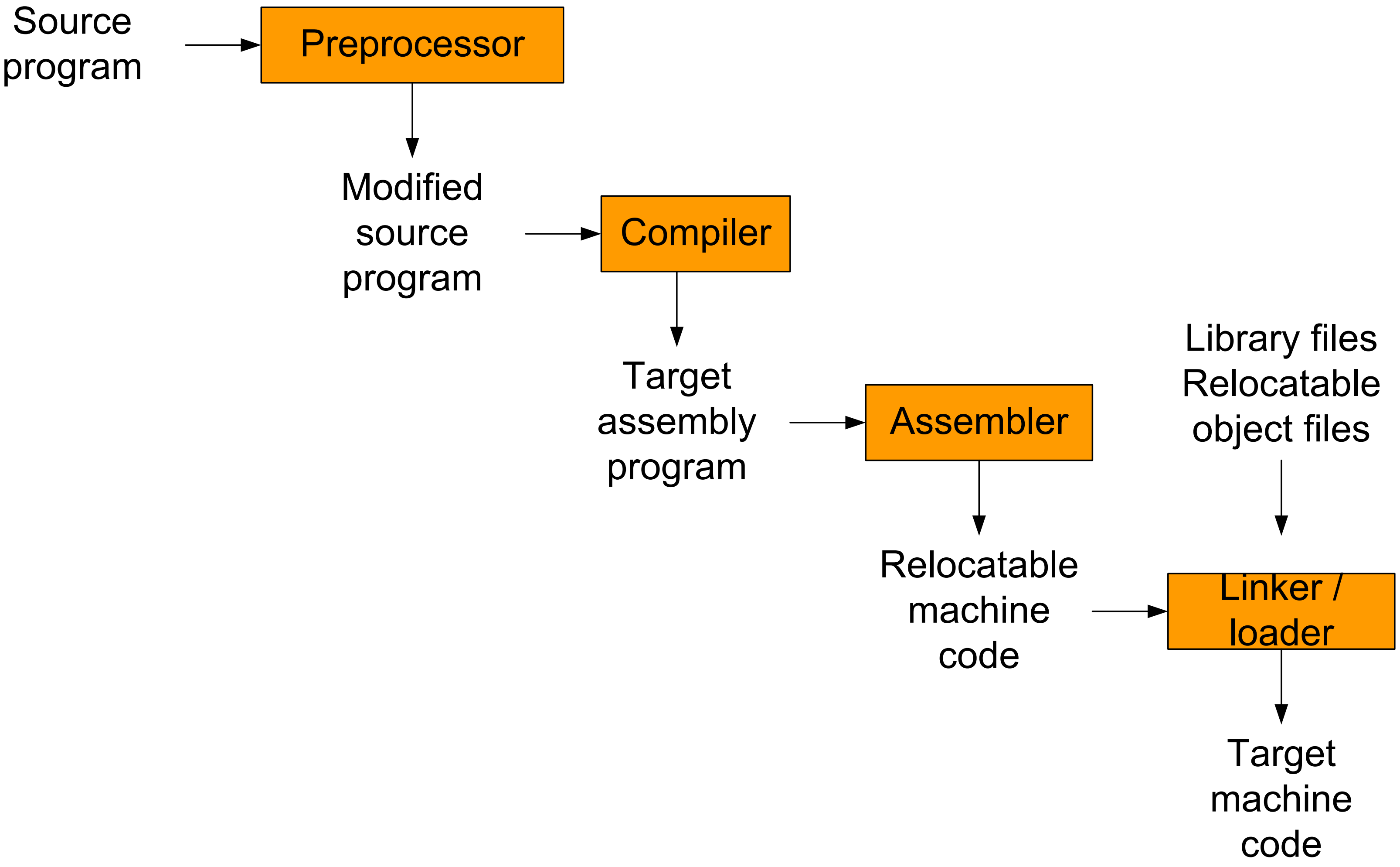


Language processors

An **interpreter** directly executes the operations specified in the source program on inputs supplied by the user.



Language processors


Source program

Preprocessor

Modified source program

Compiler

Target assembly program

Assembler

Library files Relocatable object files

Relocatable machine code

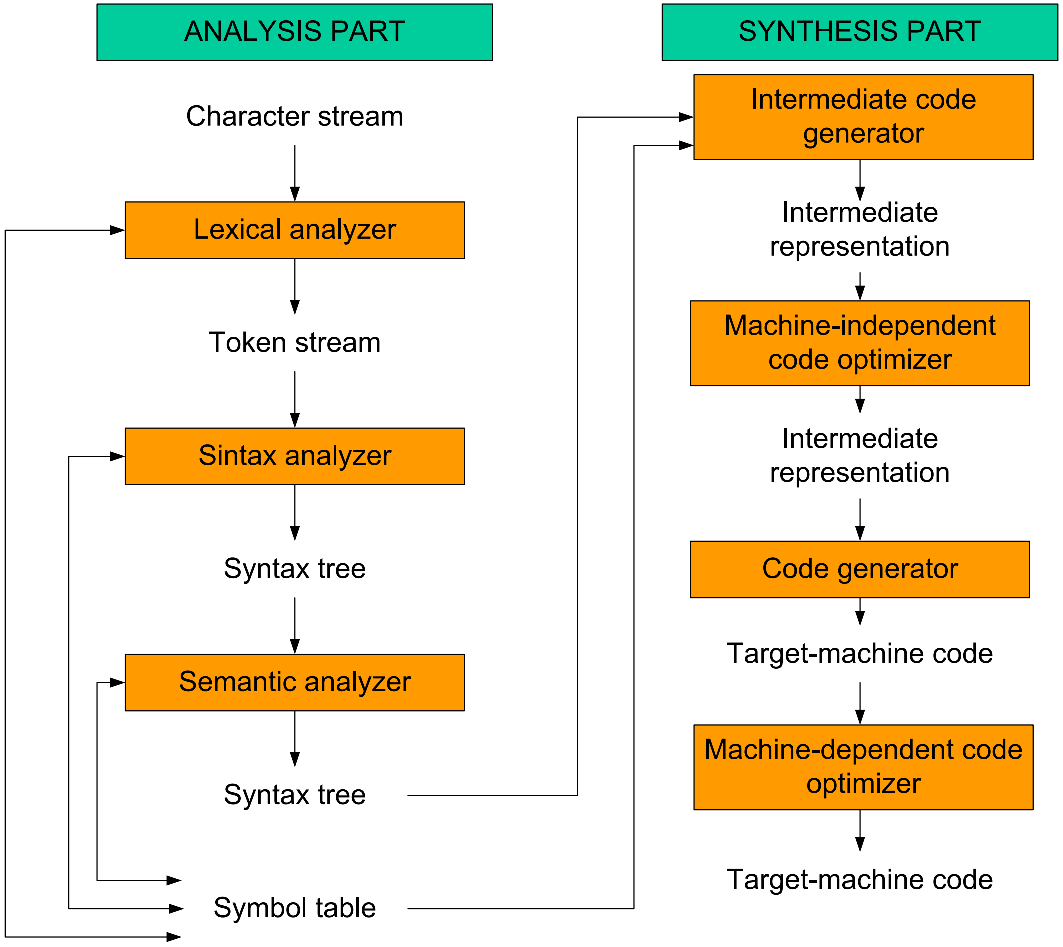
Linker / loader

Target machine code

1.2. The structure of a compiler

The steps of compilation

- Lexical analysis
- Parsing
- Semantic analysis
- Optimization
- Code generation



Lexical analysis

First step: to recognize words.

I		see		a		man	.

```
{pronoun, ’I’}
{whitespace, ’ ’}
{verb, ’see’}
{whitespace, ’ ’}
{article, ’a’}
{whitespace, ’ ’}
{noun, ’man’}
{punctuation mark, ’.’}
```

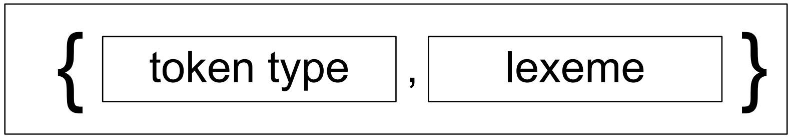
The structure of a compiler

Lexical analysis divides program into “*words*” or “*tokens*”.

if		x	==	0		then		y	=	1	;		else		z	=	2	;

```
{keyword, 'if'}
{whitespace, ' '}
{identifier, 'x'}
```

…



{ token type , lexeme }

token

Lexical analysis

Time		flies		like		an		arrow	.

```
{noun, 'time'}
{verb, 'flies'}
{prep, 'like'}
{article, 'an'}
{noun, 'arrow'}
```

Correct?

Lexical analysis

Time		flies		like		an		arrow	.

{noun, ’time’}	{noun, ’time’}
{verb, ’flies’}	{noun, ’flies’}
{prep, ’like’}	{verb, ’like’}
{article, ’an’}	{article, ’an’}
{noun, ’arrow’}	{noun, ’arrow’}

We don’t know exactly without a context.

Lexical analysis

FORTRAN EXAMPLE

```
do 5 N=1,25
```

Cycle till the label 5, the variable N changes from 1 to 25.

```
do 5 N=1.25
```

Blanks are unimportant.

Variables can be undeclared.

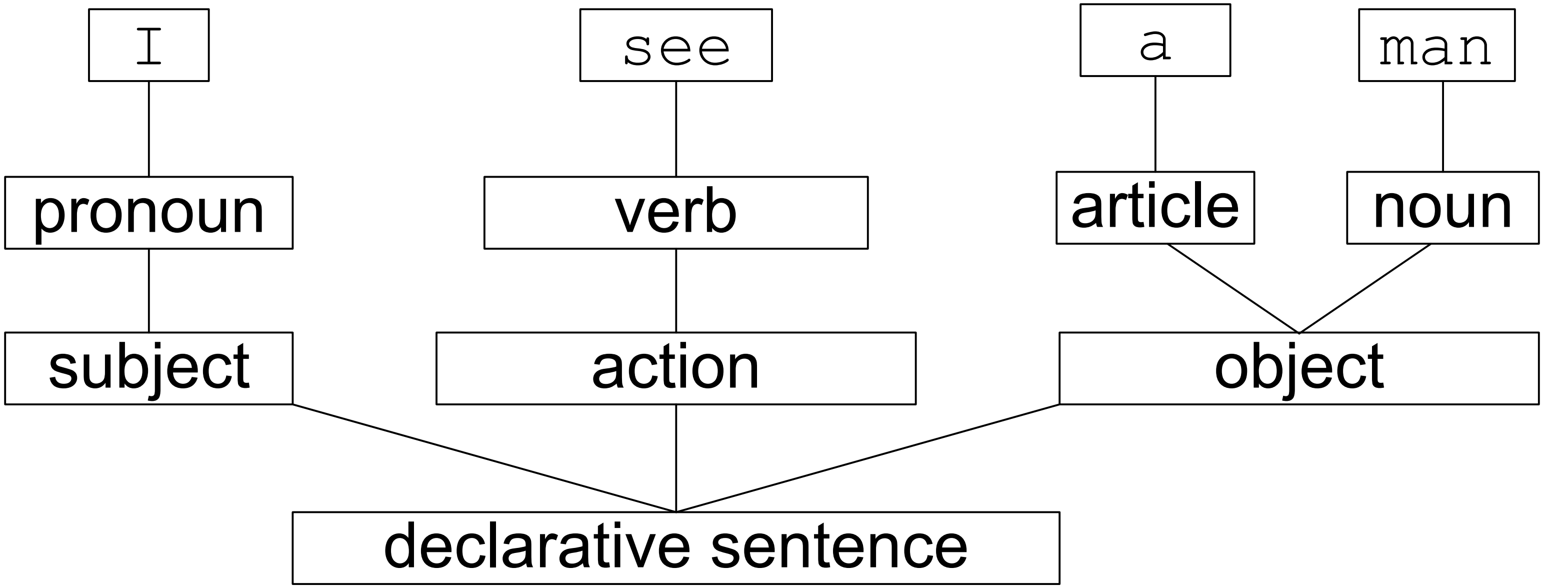
```
do5N=1.25
```

Assignment of the variable do5N.

We don’t know if ‘do’ is a keyword without going ahead.

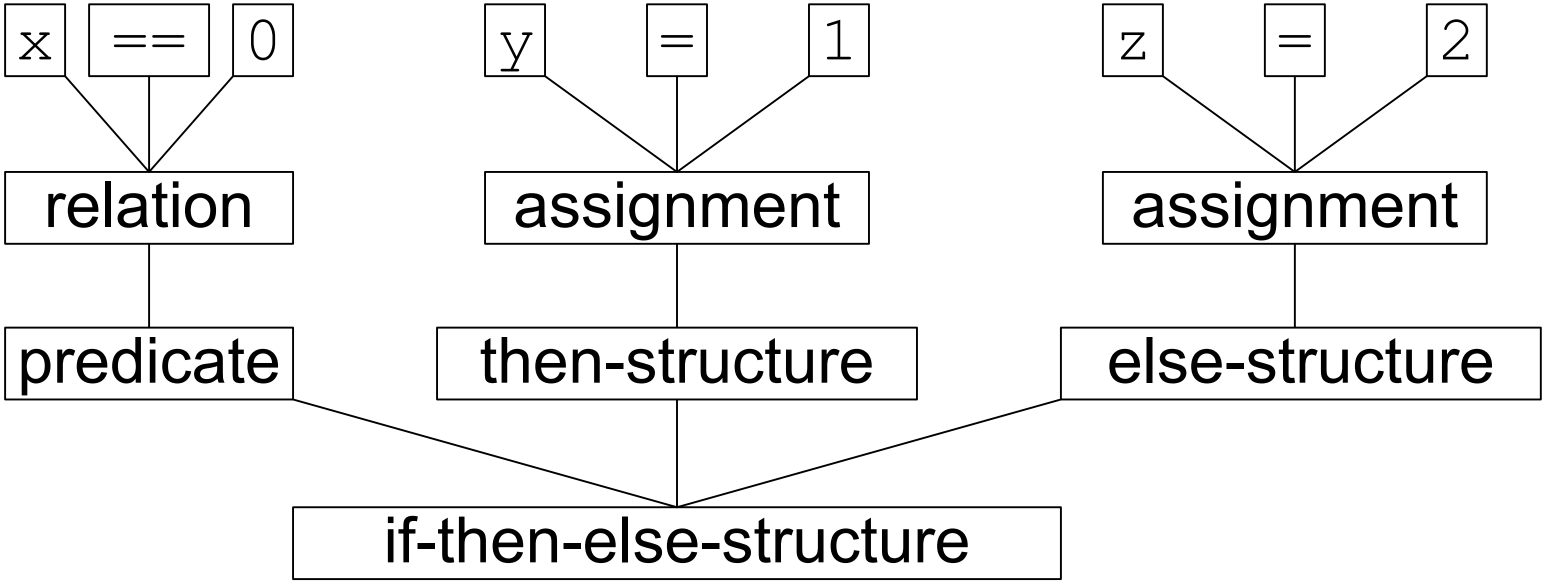
Syntax analysis

Second step: to understand the structure of the sentence.



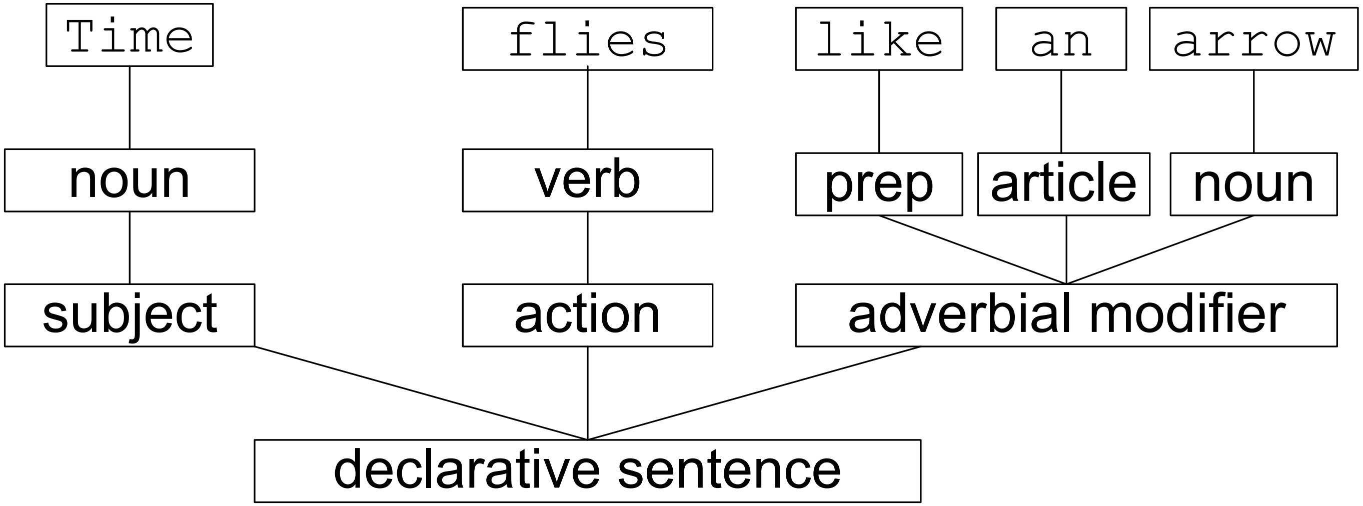
Syntax analysis

Syntax analysis (parsing) understands sentence structure.



Syntax analysis

Ambiguity!



Time

flies

like

an

arrow

noun

verb

prep

article

noun

subject

action

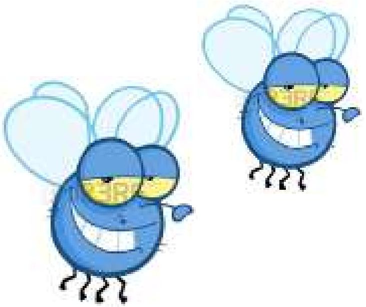
adverbial modifier

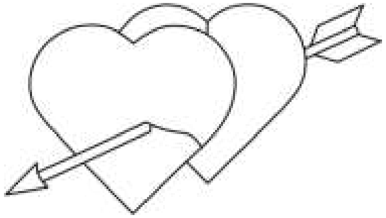
declarative sentence

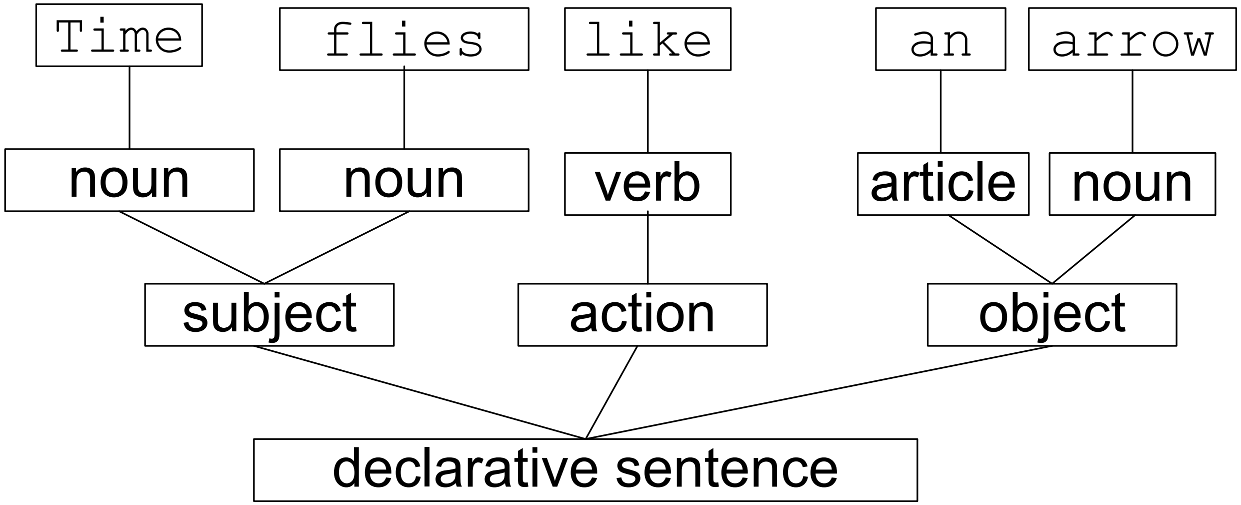
Syntax analysis

Ambiguity!

Try to guess from the context.







Time flies like an arrow

noun noun verb article noun

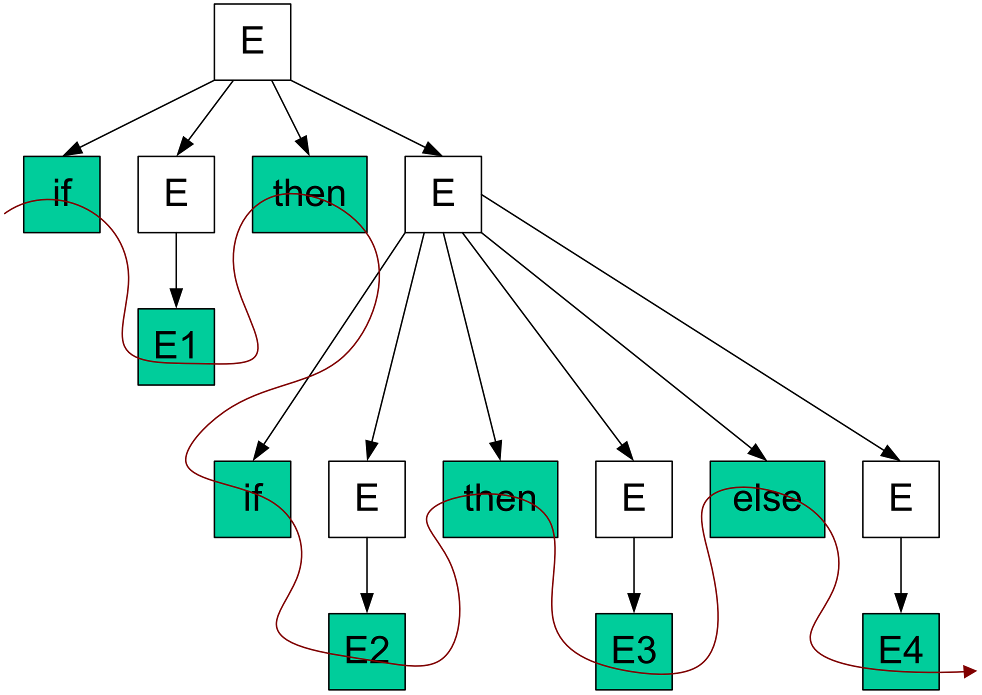
subject action object

declarative sentence

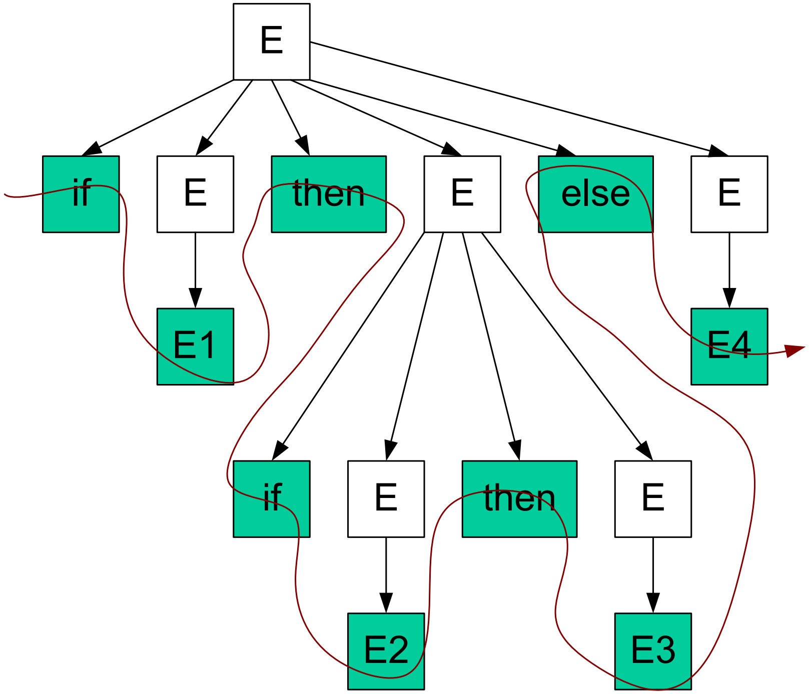
Syntax analysis

Ambiguity!

if E1 then if E2 then E3 else E4



if E1 then if E2 then E3 else E4



Semantic analysis

Third step: to understand meaning.

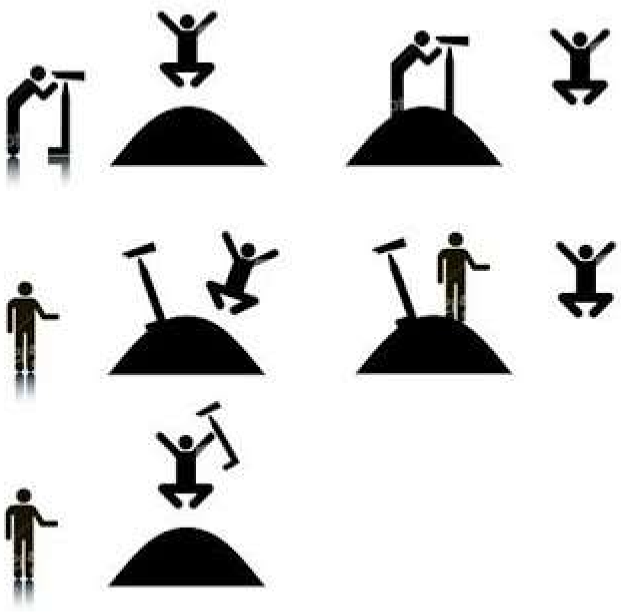
I saw the man on the hill with a telescope.

Semantic analysis

Third step: to understand meaning.

I saw the man on the hill with a telescope.

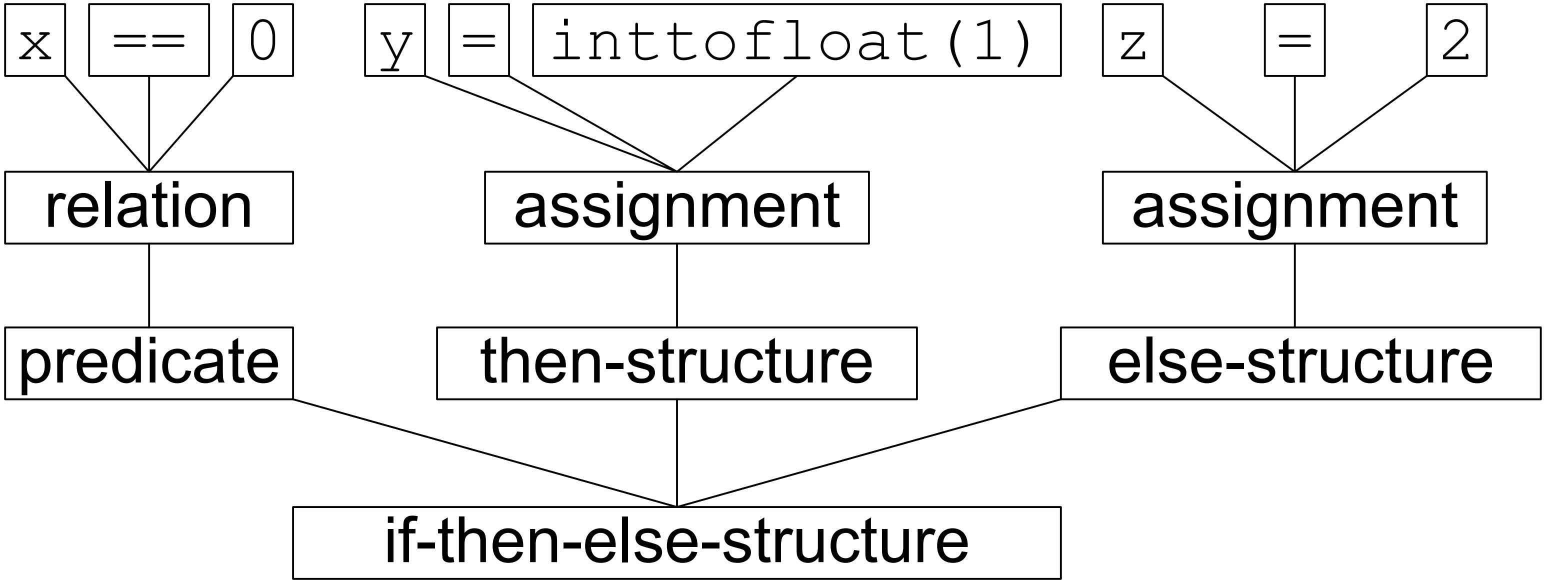
Too hard!



Semantic analysis

Semantic analysis catches inconsistencies.

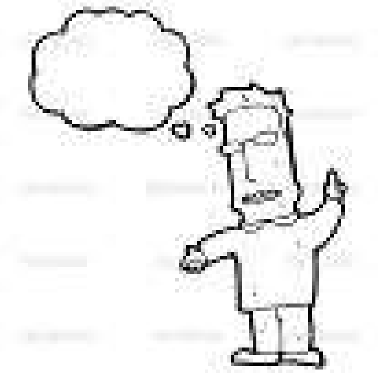
```
x,z: int;
y: float;
```



Optimization

Forth step: to say more clear

Oh… mmm… your train… let me see… it's leaving… it's leaving… in five minutes.



Your train is leaving in five minutes!

Optimization

Optimization automatically modifies programs so that they can:

- run faster;
- use less memory.

```
for (i=0;i<100;i++)
   {
      x=2*i;
      a[i]=x;
   }
```

```
for (i=0;i<100;i++)
      a[i]=2*i;
x=198;
```

Code generation

Fifth step: to translate!

We are close to the end.

Скоро все закончится!

Code generation

Code generation produces assembly code.

```
if (x==0) then
    y=1;
else
    z=2;
```

```
.686P
.MODEL FLAT, STDCALL
.DATA
.CODE
START:
TEST EAX, EAX
JZ ZERO
JMP NONZERO
ZERO:
MOV EBX, 1
NONZERO:
MOV ECX, 2
END START
```

Code generation

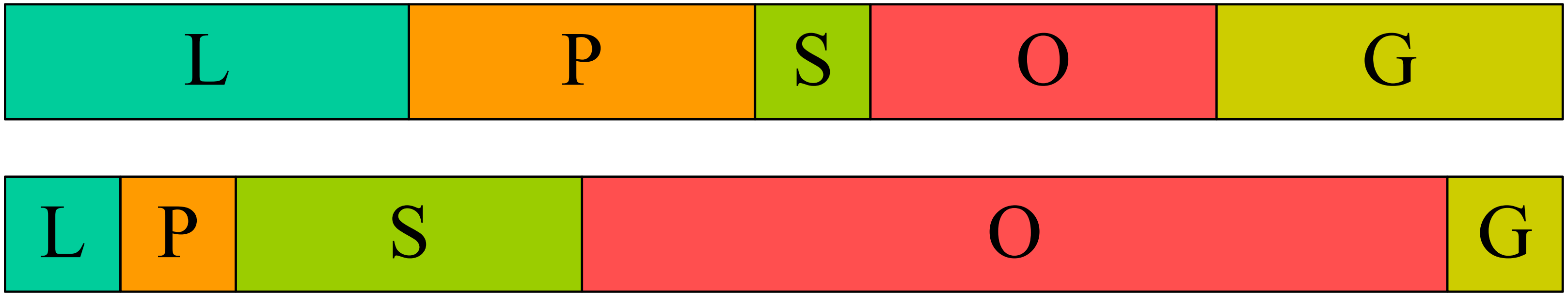
Code generation produces assembly code.

```
.386
.model flat

extrn ExitProcess:PROC
extrn MessageBoxA:PROC

.data

Ttl db "First program",0h
Msg db 'Hello,World!!!!',0h
```

```
.code

start:
push 0h
push offset Msg
push offset Ttl
push 0h
call MessageBoxA
push 0h
call ExitProcess
end    start
```

Proportion

- Things have changed since FORTRAN


L
P
S
O
G
L
P
S
O
G