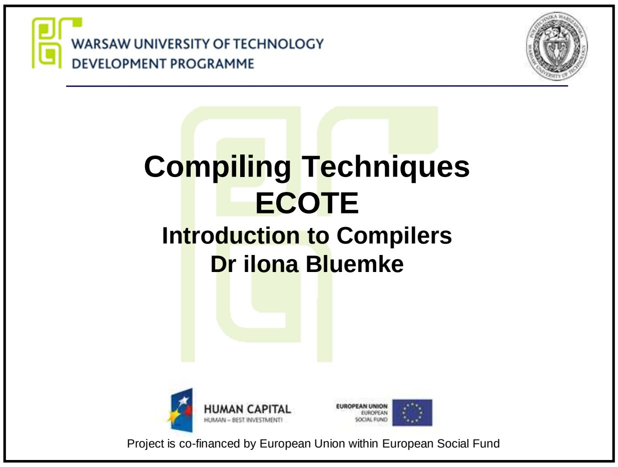


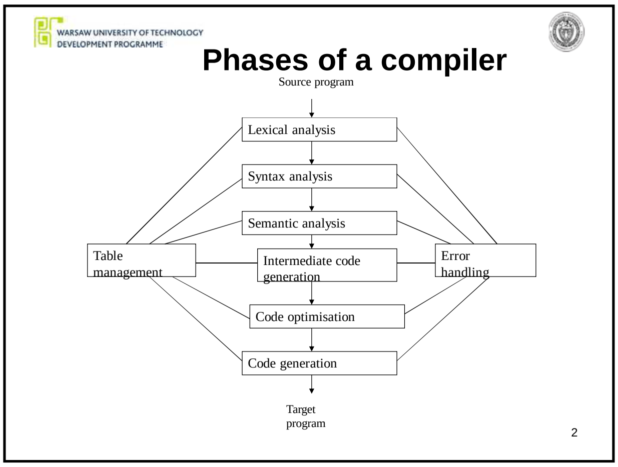
WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Compiling Techniques
ECOTE
Introduction to Compilers
Dr ilona Bluemke

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

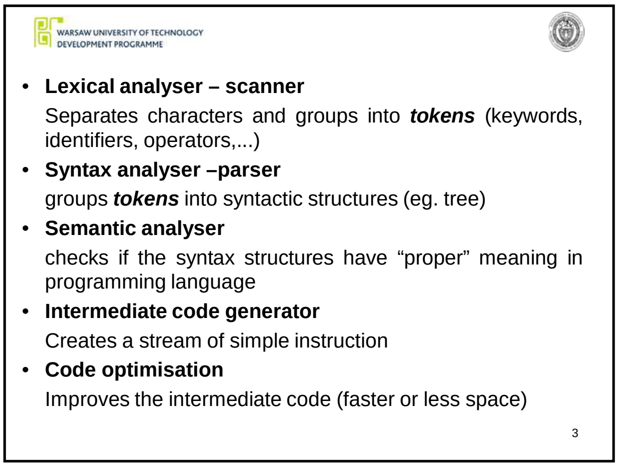
EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund



WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Phases of a compiler



WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

- **Lexical analyser – scanner**

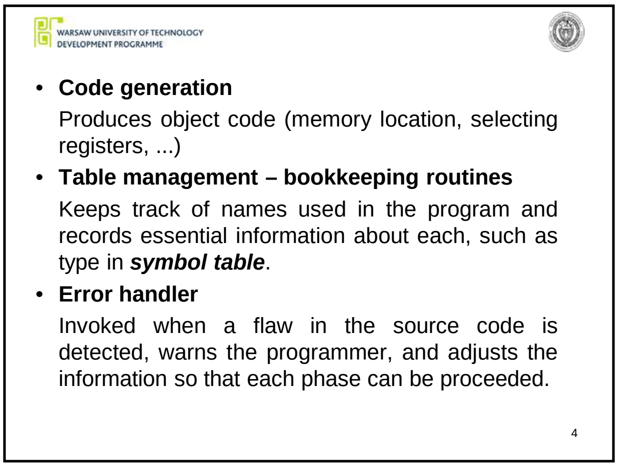
  Separates characters and groups into ***tokens*** (keywords, identifiers, operators,...)
- **Syntax analyser –parser**

  groups ***tokens*** into syntactic structures (eg. tree)
- **Semantic analyser**

  checks if the syntax structures have “proper” meaning in programming language
- **Intermediate code generator**

  Creates a stream of simple instruction
- **Code optimisation**

  Improves the intermediate code (faster or less space)



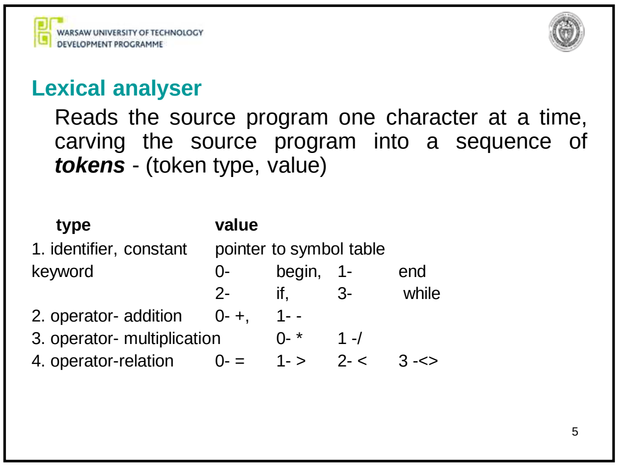
WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

- **Code generation**

  Produces object code (memory location, selecting registers, ...)
- **Table management – bookkeeping routines**

  Keeps track of names used in the program and records essential information about each, such as type in ***symbol table***.
- **Error handler**

  Invoked when a flaw in the source code is detected, warns the programmer, and adjusts the information so that each phase can be proceeded.

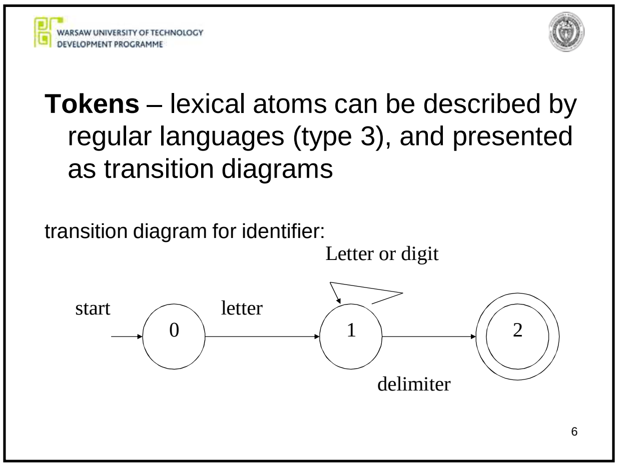


WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Lexical analyser

Reads the source program one character at a time, carving the source program into a sequence of ***tokens*** - (token type, value)

type	**value**			
1. identifier, constant	pointer to symbol table			
keyword	0-	begin,	1-	end
	2-	if,	3-	while
2. operator- addition	0- +,	1- -		
3. operator- multiplication		0- *	1 -/	
4. operator-relation	0- =	1- >	2- <	3 -<>



WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Tokens – lexical atoms can be described by regular languages (type 3), and presented as transition diagrams

transition diagram for identifier:

Code for starting state 0:

```
State 0:    C:= getchar();
            if letter(C) then goto state 1
            else fail;
```

letter(C) returns true if C is a letter

fail() retracts the lookahead pointer and starts up the next transition diagram

Code for state 1:

```
state 1     C:= getchar();
      if letter(C) or digit(C) then goto state 1
      else
            if delimiter(C) then goto state 2
            else fail;
```

delimiter(C) returns true if C is a character that could follow an identifier

Code for state 2 – identifier found

```
state 2:    retract();
            return (id, install())
```

retract() delimiter, which is not a part of identifier must be retracted from input

install() installs the identifier in symbol table

return code for identifier (type) and value – pointer to symbol table

Syntax analyser – parser

- checks that the tokens appearing in the input occur in patterns that are permitted by the specification for the source language
- groups tokens into syntactic structures (eg. tree)

Syntax of programming languages can be described by **context free languages (type 2)**, which can be generated by context free grammars.

Simplified syntax of assignment statement:

1. S → a ← E ← assignment operator
2. E → E + A
3. E → A
4. A → A * B
5. A → B
6. B → (E)
7. B → a a token type identifier/constant

statement :

abc := (arg1 + arg2) * 9

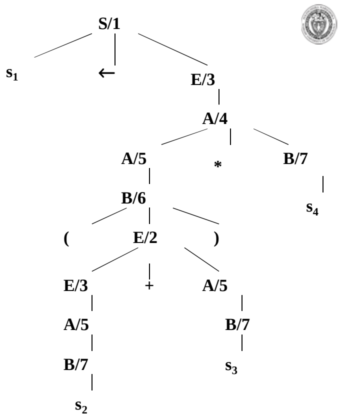
after lexical analysis:

$\mathbf{s_1 \leftarrow (s_2 + s_3) * s_4}$

Symbol table:

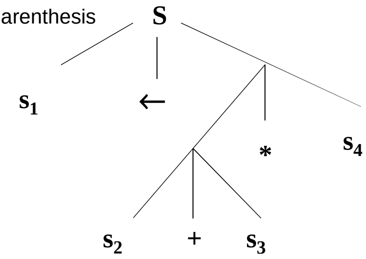
s_1	abc	variable	int
s_2	arg1	variable	int
s_3	arg2	variable	int
s_4	9	constant	int

after syntax analysis – parse tree



Parse tree optimization

- removal of simple production
- removal of parenthesis



Semantic analysis

- checks if the syntax structures have “proper” meaning in programming language
- determinates the type of intermediate results
- checks that arguments are of types that are legal for an application of operator

Intermediate code generator

Creates a stream of simple instructions, intermediate language representation.

Example

<t_1, s_2,+, s_3 >, <t_2, t_1,*, s_4 >

t_1, t_2 temporary locations

Intermediate language contains unconditional and conditional branching statements,

high-level flow of control statements (while, if-then-else) are translated into lower level tree address statements.

Code optimization

Improves the intermediate code (faster or less space)

Example

- elimination of common subexpressions

  a := b + c + d; e := b + c + f;

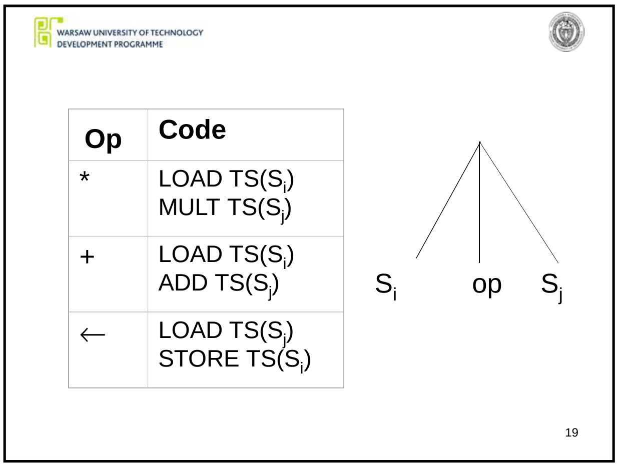
  evaluated as:

  t1 := b + c;
  a := t1 + d;
  e := t1 + f;
- loop optimization (loop invariant detection)

remove computations that produce the same result before the loop

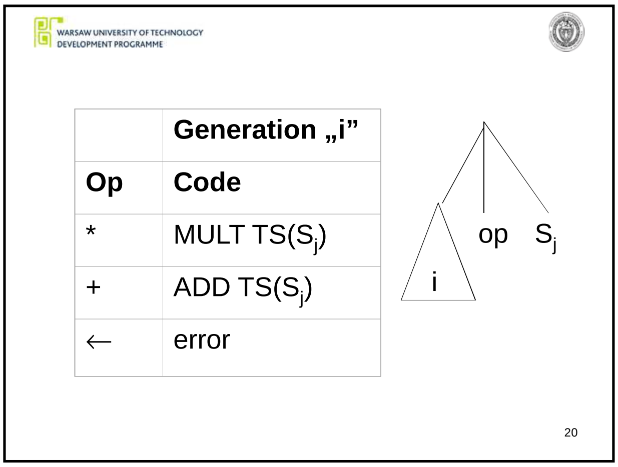
Code generation

Produces object code by converting sequence of intermediate code into a sequence of machine instructions (using **semantic equivalents**).



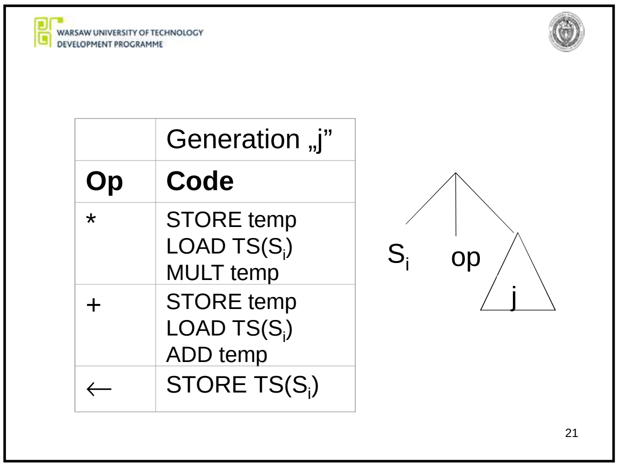
Op	Code
*	LOAD TS(S_i) MULT TS(S_j)
+	LOAD TS(S_i) ADD TS(S_j)
←	LOAD TS(S_j) STORE TS(S_i)

19



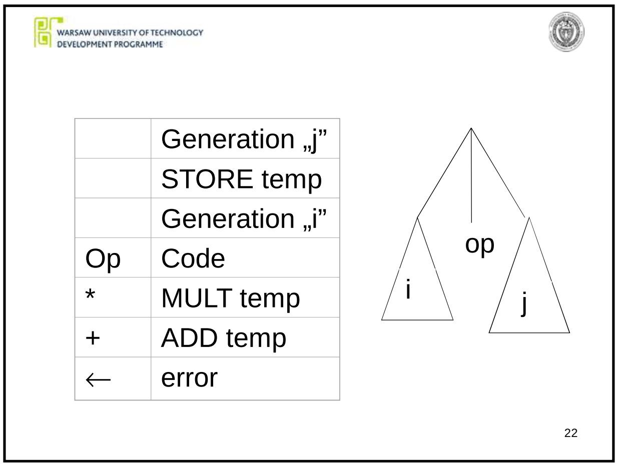
	Generation „i”
Op	**Code**
*	MULT TS(S_j)
+	ADD TS(S_j)
←	error

20



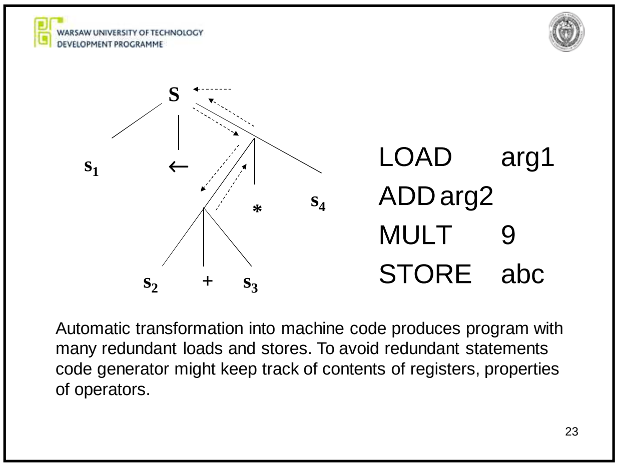
	Generation „j”
Op	**Code**
*	STORE temp LOAD TS(S_i) MULT temp
+	STORE temp LOAD TS(S_i) ADD temp
←	STORE TS(S_i)

21



	Generation „j”
	STORE temp
	Generation „i”
Op	Code
*	MULT temp
+	ADD temp
←	error

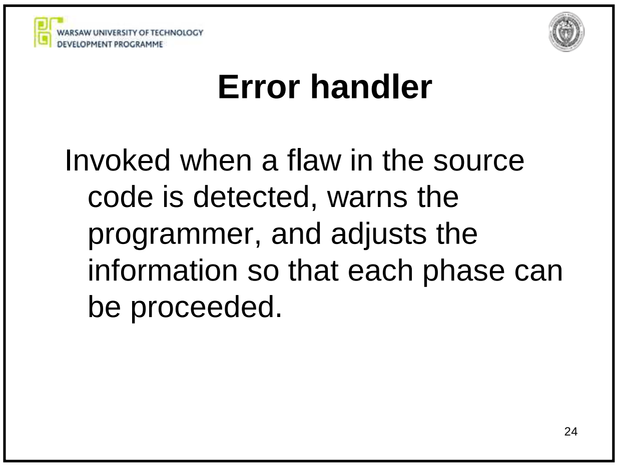
22



```
LOAD    arg1
ADD     arg2
MULT    9
STORE   abc
```

Automatic transformation into machine code produces program with many redundant loads and stores. To avoid redundant statements code generator might keep track of contents of registers, properties of operators.

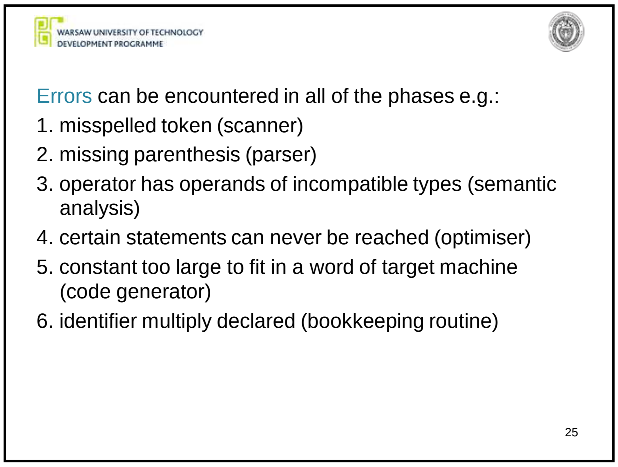
23



Error handler

Invoked when a flaw in the source code is detected, warns the programmer, and adjusts the information so that each phase can be proceeded.

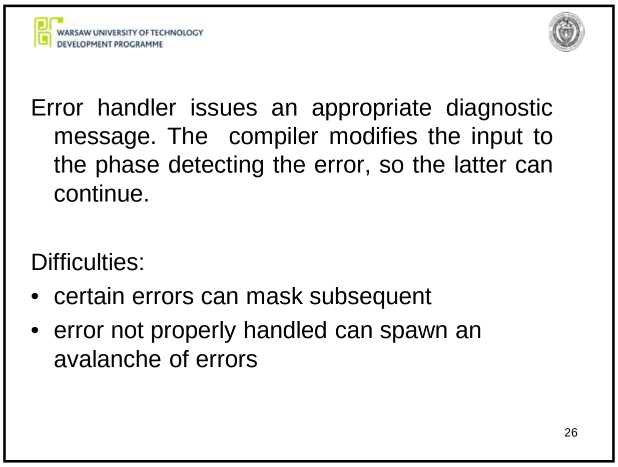
24



WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Errors can be encountered in all of the phases e.g.:

1. misspelled token (scanner)
2. missing parenthesis (parser)
3. operator has operands of incompatible types (semantic analysis)
4. certain statements can never be reached (optimiser)
5. constant too large to fit in a word of target machine (code generator)
6. identifier multiply declared (bookkeeping routine)

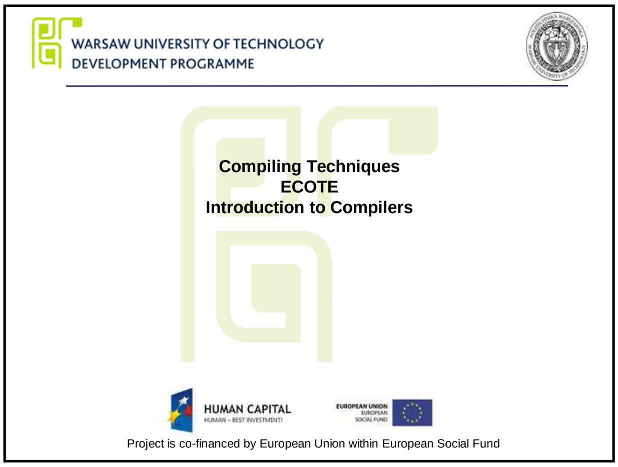


WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Error handler issues an appropriate diagnostic message. The compiler modifies the input to the phase detecting the error, so the latter can continue.

Difficulties:

- certain errors can mask subsequent
- error not properly handled can spawn an avalanche of errors



WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Compiling Techniques
ECOTE
Introduction to Compilers

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN SOCIAL FUND

Project is co-financed by European Union within European Social Fund