LEX

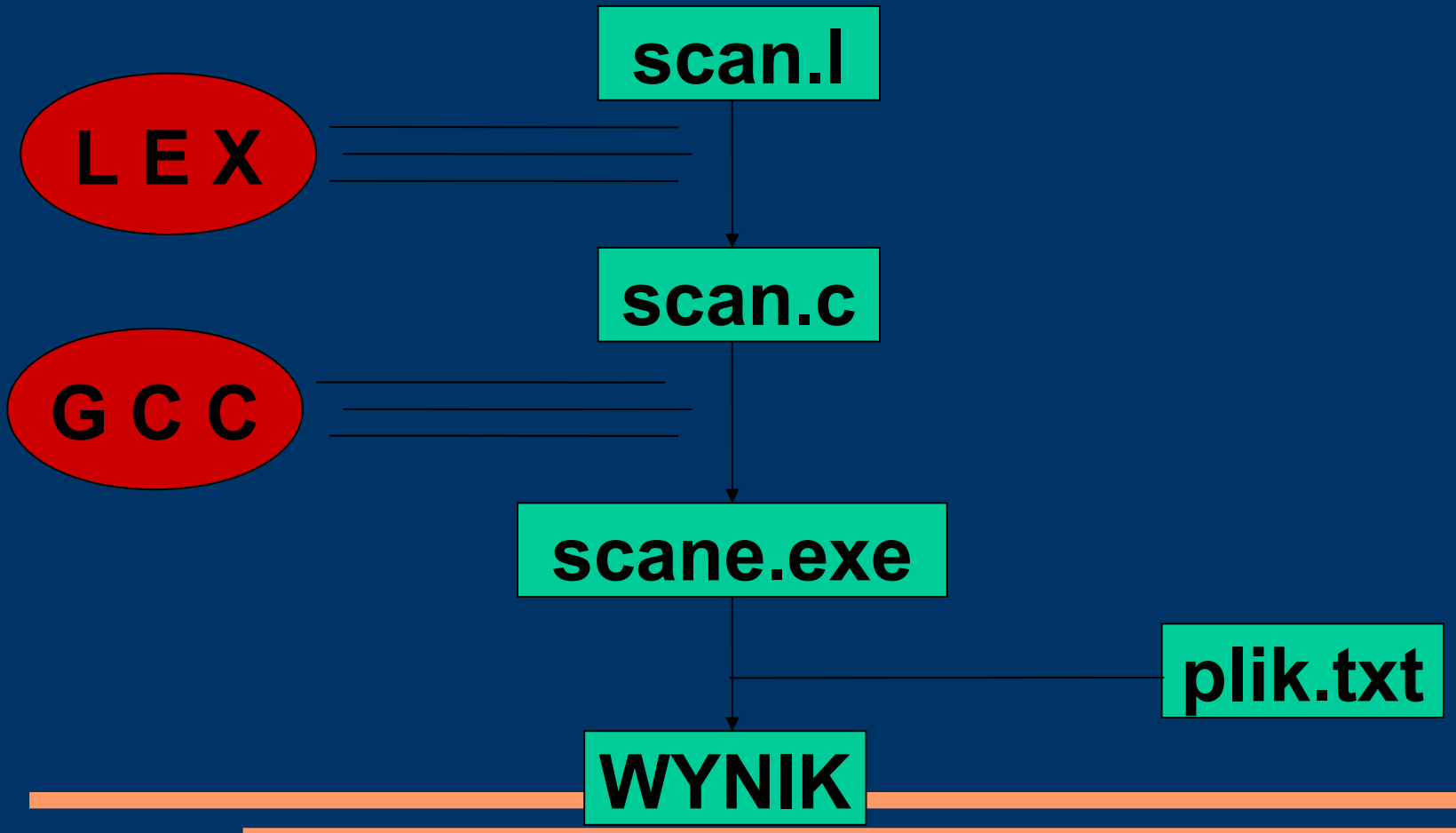
Lexical analyzer generator

GENERATOR LEX

- The task of LEX generator is generate a source code (default in C) of lexical analyzer;
- The source code is generated by Lex on the basis of a file, which contain all the processing rules;
- Rules file is created by the user;

GENERATOR L E X

Diagram of the operational organization Lex:



GENERATOR LEX

- **flex –l scan.l** (using generator LEX)
- **gcc scan.c -o scan.exe** (compilation C++)
- **scan.exe < plik.txt** (analysis plik.txt)

GENERATOR LEX

- An important feature of the analyzer is able to use it for larger applications;

- Each of the generated source code contains a function allowing you to connect the lexical analyzer to other applications. This function is **yylex();**

CREATING FILE WITH RULES

- Each file with the specifications for the LEX, should consist of three sections;
- First section is the **definition section**;
- In the definitions section, as its name suggests, we put definitions and declarations of variables, constants, declarations of states and processor macro;

CREATING FILE WITH RULES

- The definition section can include a piece of code which prescribes the system directly to the lexical analyzer;
- This code must be properly "packaged";
- The opening of portion directly prescribed to the analyzer should be preceded {%, while its closing %};

CREATING FILE WITH RULES

- An example of the construction of the definitions section:

```
%{
    #include<stdio.h>
    int variables;
    int yylex();
    int secound_variables=1;
%}
```

CREATING FILE WITH RULES

- The second section is the processing section;
- In the processing section, we put all the rules of conduct, under which the analyzer will be generated;
- Rules of conduct are is differently recipes for what the analyzer has to do when encountering a specific "problem" (symbol);

CREATING FILE WITH RULES

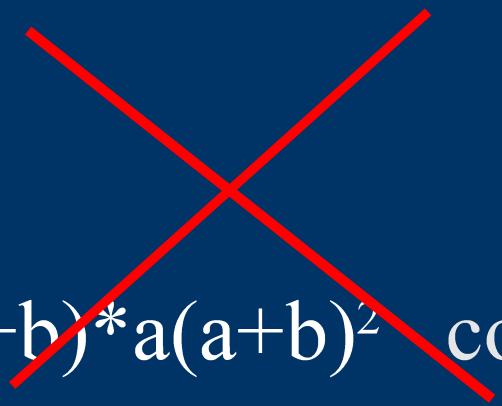
- Construction of the processing rule based on two basic parts: the pattern and the operation;
- Construction of the processing rule looks like so:

  the pattern the operation

- The pattern is written as a regular expression;
- The operation is a block of C-language instruction;

CREATING FILE WITH RULES

- An example of the construction of processing rules:



$(a+b)^*a(a+b)^2$ cout<<‘’In word A the third symbol of the end is a’’;

REGULAR EXPRESSIONS

- We give a few symbols used to write regular expressions which we can find in patterns of processing rules:

1. Symbols of line:

^… - beginning of the line;

…$ - end of line;

REGULAR EXPRESSIONS

1. Symbols of logical operations:

ab - concatenation;

a|b - alternation;

a* - closure;

a+ - positive closure (closing after deduction of empty words);

a? – optionality (symbol "and" not occur or occurs once);

REGULAR EXPRESSIONS

1. Repeat of symbol:

a{n} – repeat of symbol „a” n – times;

a{n,m} – range of repeat of symbol (i.e. a^n,…,a^m);

() – determine the degree of importance (c(a|d)|(e+));

REGULAR EXPRESSIONS

1. The class of signs:

[a-z] - means any character in the range from a small letter "a" to small letter "z";

[^a-z] – means any character outside the class [a-z] (it is negation of range);

[a-zXY] – means any character in the range [a-z] or a large letter X or Y;

[0-9] – means any digit from 0 to 9;

RAREGULAR EXPRESSIONS

1. The class of the sings:

. - means any character, which is not a symbol of end of line;

\... – the sign precedes the special sequences (like C), for example:

\n – is a sign of end of line;

\t – is a sign tab;

CREATE OF FILE WITH THE RULES

- Example of build of rule:

~~(a+b)*a(a+b)^2~~ cout<<‘’In word A the third symbol from the end is a”;

We should to write

(a|b)*a(a|b){2} cout<<‘’In word A the third symbol from the end is a’’;

RAREGULAR EXPRESSIONS

Other important rules for creating patterns:

- Patterns containing spaces are included in quotation marks;
- Comment fits between the markers /* a */;
- Unmatched characters are printed for exit;

EXAMPLE

The regular expression which accept the address of website:

[Ww]{3} \. [A-Za-z0-9._-]+ \. [A-Za-z]{3} \. [Pp] [Ll]

Example

The regular expression which accept the keywords in Ada language, for example: „begin” and „end”:

[A-Za-z]{3,5}

[A-Za-z]{5} | [A-Za-z]{3}

[Bb][Ee][Gg][Ii][Nn] | [Ee][Nn][Dd]

Example

The regular expression which accept all identifiers (variables, constants) in C language:

[A-Za-z _] [A-Za-z0-9 _]*

Example

The regular expression which accept the date:

```
([0-9]{2} \- [0-9]{2} \- [0-9]{4}) |
([0-9]{2} \. [0-9]{2} \. [0-9]{4}) |
([0-9]{4} \- [0-9]{2} \- [0-9]{2}) |
([0-9]{4} \. [0-9]{2} \. [0-9]{2})
```

```
([0-9]{2} (\- | \.) [0-9]{2} (\- | \.) [0-9]{4}) |
([0-9]{4} (\- | \.) [0-9]{2} (\- | \.) [0-9]{2})
```

Example

The regular expression which accept the address of e-mail:

[A-Za-z0-9 . _ -]+@[A-Za-z0-9 . _]+\.[A-Za-z]{2,4}

CREATE OF FILE WITH THE RULES

- The third section of file with specifications for the program LEX is a subprograms section;

- The section of subroutines can contain (as the name implies) the definitions of the functions, that will then be used by the lexical analyzer;

CREATE OF FILE WITH THE RULES

- A declarations of functions, which are included in this section, look as declarations of functions in C;
- The function yywrap() – is very important functions and has the special interest.When we run (previously generated) lexical analyzer, then the function yywrap () is executed always after processing the input data;

EXAMPLE

- Example of function, which is set out in subprogram section:

```
int main()
{
return yylex();
}
```

- It requires, of course, a prior declaration of yylex() in the form::

```
int yylex();
```

CREATE OF FILE WITH THE RULES

- When creating a file with a processing rules, you can use global variables;
- The declaration of this type variables can be found in the first section (see definition) of file;
- Global variables can of course be used in each of the next section, for example. incremented, altered, or read;

CREATE OF FILE WITH THE RULES

- In addition to global variables, you can also use the built-in variables;
- There are two particularly important variables built yyleng and yytext;
- The variable yyleng is type int and determines the length of the match;
- The variable yytext is a char (more preciously a string of chars), and show a token (stors record inside stream of input data that fits the pattern);

CREATE OF FILE WITH THE RULES

- In file with specification for Lex, we may use the regular definitions;
- Creating the definition of regular means assigning a identifier to a some regular expression;
- This identifier can be used later in the pattern;
- The regular definition is created in section of definitions;;

CREATE OF FILE WITH THE RULES

- The regular definitions we write in the section of definition, however, after the block directly written to the analyzer;
- This situation we see in the next example;

EXAMPLE

```
%{
    #include<stdio.h>
    int yylex();
%}
identyfikator   [Ii] [Ff]
%%
{identyfikator}  {printf(‘’ We get leksem IF’’);}
%%
```

CREATE OF FILE WITH THE RULES

- All three discussed a sections of the file with the specification for the program LEX, are separated by double percent sign - %%;

- Schema file with the specification (contains rules) for the LEX, we can demonstrate to the table:

CREATE OF FILE WITH THE RULES

The definition section

%%

The processing rules section

%%

The subprograms section

Ambiguity

- When we looked at the principles behind the generator LEX, we can look at the major issue of ambiguity when operating LEX;
- Example:

m* {cout<<‘’*’’;}

mmm {cout<<‘’+’’;}

We provide a data stream form:

mmmmmmmmmm;mmm

Ambiguity

- **PRINCIPLE OF LONGEST FIT** – specifies that if you have two or more rules for which patterns are met, it selects this rule for matching is the longest;;

- **PRINCIPLE OF EARLY FIT** – says that, when fit have the same length, is selected rule, which was placed first in the file specification;

Ambiguity

- Using the first principle we have

*;+

- If we have the input data of the form

mmm

then using the second principle we get

+

RETRACTION

- The concept of retraction associated with the process operation of the analyzer;
- The analyzer converts multiple patterns simultaneously in search of the best (ie. The longest). He eventually abandons patterns less "promising" by focusing on patterns that can give longer fit;
- In the event of failure returns to the abandoned patterns;...

THE END

END OF THIRD LECTURE