Compilers

Lab 2

Piotr Błaszyński

9th March 2022

Tasks (explained later in the document):

- prepare the list of lexemes in the lexical analyzer (file zX.l)
- prepare the list of tokens in the initial part of the syntactic analyzer (file def.y),
- in the lexical analyzer add return tokens - for single symbols it is the ASCII code of the symbol, for keywords, numbers, identifiers and multi-character symbols it is the token code.

The lexical analyzer description file consists of 3 parts separated by a double sign '%'. The first part is the header part - it allows you to attach header files, prepare prototypes of functions called in other parts and write definitions for flex. The second part contains text processing rules consisting of a pattern and an action which is executed if the pattern is matched. In this part (action) we can also return the token code to the syntax analyzer. The third part is the code in C language copied directly to the result file in C.

Basic rule elements: The '[' and ']' characters (square brackets) allow you to write a rule for one element of a regular expression, if such a rule is followed by a '+' character then the rule must occur in the matched text at the appropriate place at least once (it may repeat multiple times, 1-), if the next character is a '*' then the rule may occur multiple times but need not (0-). At the end of the list of rules there should be a dot ('.'), which in this context means no previous matching - which causes an error (unknown lexeme).

The syntax analyzer description file is also built with 3 sections: header, rules and actions (here called semantic - we will not use them yet at this stage), code in C/C++. At this stage we only need to define tokens (examples below) for all

keywords, numbers, identifiers and multi-character symbols (for single characters the token code is its ASCII code).

After defining tokens and parsing the def.y file with bison, we can (among other things, bison will generate a header file with numeric token codes) return token codes from within the lexical analyzer. Examples can be found in directories z4, z5 and z6.

Definition of a token without specifying a type:

```
%token LEQ
```

Definition of multiple tokens without specifying a type:

```
%token LEQ GEQ EQ
%token FOR INT DOUBLE
```

Definition of a token with type specification:

```
%token <text> ID
```

Return of a token (whole rules with actions in basic version) from lexical analyzer:

```
\=                          {return '=';}
\<\=                        {return LEQ;}
"int"             {return INT;}
[A-Za-z_][A-Za-z0-9_]*   {return ID;}
```