Compilers

Lab 3

Piotr Błaszyński

16th March 2022

Tasks (explained later in the document):

- prepare a grammar for a single expression consisting of variables, numbers and operators (file def.y - example grammar in the z5 directory),
- pass the semantic values of identifiers and numbers (integer and float) from the lexical analyzer to the syntactic analyzer,
- implement the main function (only one version in def.y), with support for call parameters (argc, argv),
- write the semantic values of individual identifiers and numbers and operators into a file in matching order,
- test the function for expressions consisting of several (8-10) elements.

The language grammar rules processed by the bison generator consist of:

- The name of a non-terminal symbol,
- a colon character,
- definition consisting of terminal and non-terminal symbols.

Alternative definitions are separated from each other by a vertical dash ('|') and the last alternative (for order) should be followed by a semicolon (';'). The non-terminal symbol at the beginning of the grammar is chosen as the starting symbol, unless we indicate it explicitly (the *%start* directive). The order of the other rules is not important, but it is useful for the rules to be written in an orderly fashion. To define rules as terminal symbols you should use previously defined tokens defined in the first section and single character symbols (in apostrophes). After each

definition (also after each part of it) you can write an action (semantic) in the form of C/C++ code, which will be called if the rule is matched.

Example definition of non-terminal symbols factor and component:

```
component
        :component '*' factor    {printf("␣*␣\n");}
        |component '/' factor    {printf("␣/␣\n");}
        |factor                  {printf("skladnik␣\n");}
        ;
factor
        :ID                      {printf("variable\n");}
        |LC                      {printf("number\n");}
        |'(' exp ')'             {printf("parnetheses\n")
           ;}
        ;
```

To pass semantic values from the lexical analyzer to the syntactic analyzer, use the *yylval* union (its default name in code generated by flex). The fields of this union are defined in the file for bison (e.g. def.y). The lexeme values for numbers and identifiers (for other lexemes as well) are stored in the *yytext* variable.
An example definition of a union for passing semantic values, tokens whose semantic value will be passed must have the specified type:

```
%union
{
    char *text;
    int ival;
};
%token <text> ID
%token <ival> LC
```

In the syntactic analyzer, a semantic value can be referred to using the symbol $, and a number. The number indicates the place of the lexeme, whose semantic value we want to obtain, in the rule (so it is usually $1, but e.g. in case of array declarations it can be $2 or $3)
An example reference to the semantic value of an identifier.

```
factor
        :ID              {printf("id:␣%s\n", $1);}
        ;
```