DALUZ Mattéo
DOS-REIS Edinho
HEOUAINE Sarah

Language PSDHD
Lexical analysis

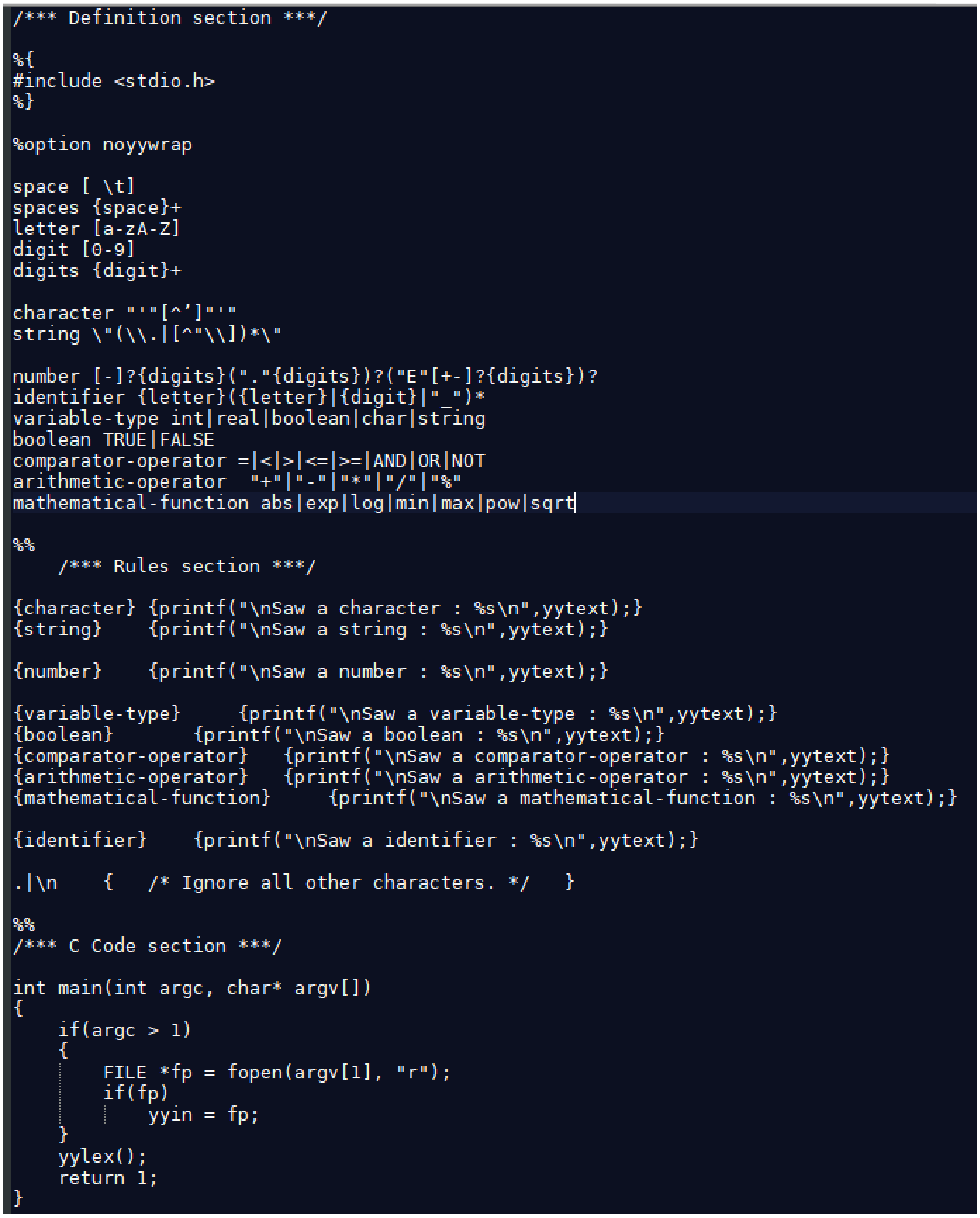
1. Changes to the grammar

Added <space> → [/t] and <spaces> → <space>+ to recognize empty spaces
Modified <character> to <character> → ‘[^’]’ to recognize all UTF-8 characters that are not ‘ and that are between single quotes
Same for <string> → **"**(\\.|[^"\\])***"** to recognize all UTF-8 characters and blank characters that are not **“** or \ and that are between double quotes

Removed <special> token because they aren’t used for characters and strings anymore

DALUZ Mattéo
DOS-REIS Edinho
HEOUAINE Sarah

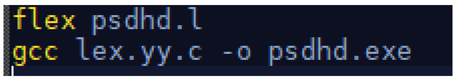
2. Lexical analysis using LEX



```
/*** Definition section ***/

%{
#include <stdio.h>
%}

%option noyywrap

space [ \t]
spaces {space}+
letter [a-zA-Z]
digit [0-9]
digits {digit}+

character "'"[^’]"'"
string \"(\\.|[^"\\])*\"

number [-]?{digits}("."{digits})?("E"[+-]?{digits})?
identifier {letter}({letter}|{digit}|"_")*
variable-type int|real|boolean|char|string
boolean TRUE|FALSE
comparator-operator =|<|>|<=|>=|AND|OR|NOT
arithmetic-operator  "+"|"-"|"*"|"/"|"%"
mathematical-function abs|exp|log|min|max|pow|sqrt

%%
    /*** Rules section ***/

{character} {printf("\nSaw a character : %s\n",yytext);}
{string}    {printf("\nSaw a string : %s\n",yytext);}

{number}    {printf("\nSaw a number : %s\n",yytext);}

{variable-type}     {printf("\nSaw a variable-type : %s\n",yytext);}
{boolean}       {printf("\nSaw a boolean : %s\n",yytext);}
{comparator-operator}   {printf("\nSaw a comparator-operator : %s\n",yytext);}
{arithmetic-operator}   {printf("\nSaw a arithmetic-operator : %s\n",yytext);}
{mathematical-function}     {printf("\nSaw a mathematical-function : %s\n",yytext);}

{identifier}    {printf("\nSaw a identifier : %s\n",yytext);}

.|\n    {   /* Ignore all other characters. */   }

%%
/*** C Code section ***/

int main(int argc, char* argv[])
{
    if(argc > 1)
    {
        FILE *fp = fopen(argv[1], "r");
        if(fp)
            yyin = fp;
    }
    yylex();
    return 1;
}
```

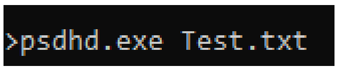
3. Token recognition testing

To test our program, we need to generate the .yy.c file first. On windows, we're using flex. Then we need to compile the .yy.c file using gcc and then we can run the output .exe file. To automate this procedure, we created a shell script that does all those steps :



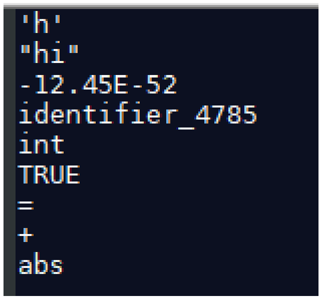
```
flex psdhd.l
gcc lex.yy.c -o psdhd.exe
```

Then, to test the lexical analizer, just run the psdhd.exe file with the test file in parameter :



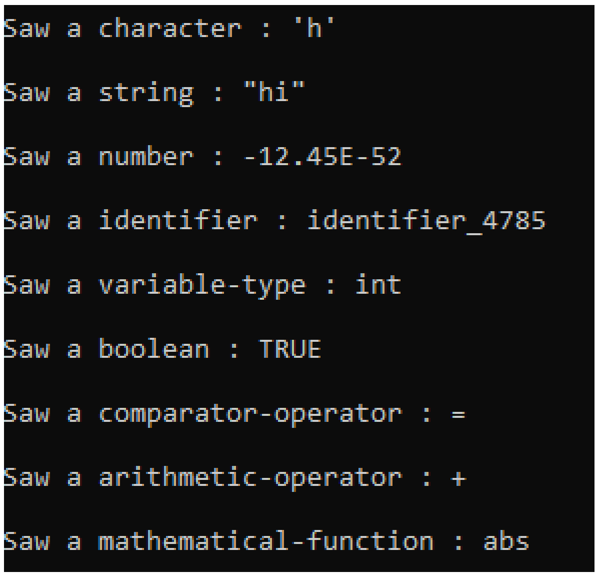
```
>psdhd.exe Test.txt
```

To recognize all the tokens we used the following test file :



```
'h'
"hi"
-12.45E-52
identifier_4785
int
TRUE
=
+
abs
```

And the output was :

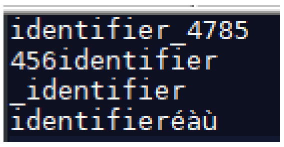


```
Saw a character : 'h'

Saw a string : "hi"

Saw a number : -12.45E-52

Saw a identifier : identifier_4785

Saw a variable-type : int

Saw a boolean : TRUE

Saw a comparator-operator : =

Saw a arithmetic-operator : +

Saw a mathematical-function : abs
```

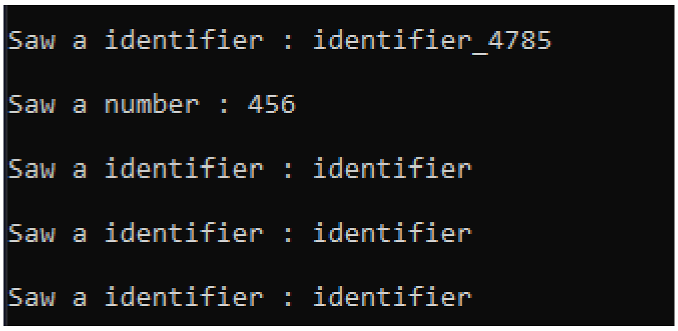
Tokens were recognized properly.

We can also test if identifiers follow the rule correctly :

With this input, only the first identifier should be recognized correctly :



```
identifier_4785
456identifier
_identifier
identifieréàù
```



```
Saw a identifier : identifier_4785

Saw a number : 456

Saw a identifier : identifier

Saw a identifier : identifier

Saw a identifier : identifier
```

We can see that only the first identifier_4785 is recognized correctly, the others are incomplete.