Coursework 2

This coursework is worth 3% and is due on 27 November at 16:00. You are asked to

1. implement a tokeniser for the WHILE language,
2. implement a parser and an evaluator for boolean and arithmetic expressions, and
3. write a WHILE program for printing out prime numbers.

You need to submit a document containing the answers for the questions below. You can do the implementation in any programming language you like, but you need to submit the source code with which you answered the questions. However, the coursework will *only* be judged according to the answers. You can submit your answers in a txt-file or as pdf.

Question 1 (marked with 1%)

Implement a tokeniser for the WHILE language. You first need to design the appropriate regular expressions for the following nine syntactic entities:

1. keywords are

   `while`, `if`, `then`, `else`, `do`, `for`, `to`, `true`, `false` `andalso`, `orelse`, `read`, `write`

2. operators are

   `+`, `-`, `*`, `%`, `==`, `!=`, `>`, `<`, `:=`

3. strings are enclosed by `"..."`
4. parentheses are `(`, {, `)` and }
5. there are semicolons `;`
6. whitespaces are either `" "` or `\n`
7. comments either start with \ \ and run to the end of the corresponding line (indicated by `\n`), or they can also run over several lines but then need to be enclosed by `/*` as the beginning marker and `*/`as the end marker
8. identifiers are letters followed by underscores _, letters or digits
9. numbers are `0`, 1, ...

Once you have implemented all regular expressions for 1 - 9, then give the token sequence for the Fibonacci program shown below in Fig. 1.

Question 2 (marked with 1%)

Implement parser combinators and an evaluation function for arithmetic and boolean expressions. Arithmetic operations should include +, -, * and % (quotient). Boolean operations should include == (equal), != (unequal), < and >.

Using the parser and evaluation function, calculate the values for

- `17 < 3 * 3 * 3`
- `(29 - 20) * 3`
- `79 - 20 * 3`
- `2 * 2 != 12 % 3`

Question 3 (marked with 1%)

Write a program in the WHILE programming language that prints out all prime numbers between 0 and a fixed number (say 100). A partial grammar of the WHILE language is given below.

$$
\begin{array}{lcl}
\textit{Stmt} & \rightarrow & \texttt{skip} \\
 & | & \textit{Id}\ \texttt{:=}\ \textit{AExp} \\
 & | & \texttt{if}\ \textit{BExp}\ \texttt{then}\ \textit{Block}\ \texttt{else}\ \textit{Block} \\
 & | & \texttt{while}\ \textit{BExp}\ \texttt{do}\ \textit{Block} \\
 & | & \texttt{read}\ \textit{Id} \\
 & | & \texttt{write}\ \textit{Id} \\
 & | & \texttt{write}\ \textit{String} \\
\textit{Stmts} & \rightarrow & \textit{Stmt}\ \texttt{;}\ \textit{Stmts} \\
 & | & \textit{Stmt} \\
\textit{Block} & \rightarrow & \{\ \textit{Stmts}\ \} \\
 & | & \textit{Stmt} \\
\textit{AExp} & \rightarrow & \ldots \\
\textit{BExp} & \rightarrow & \ldots
\end{array}
$$

As another guidance for your program have a look at the Fibonacci program and "three-nested-loops" program shown below in Figures 1 and 2.

```
/* Fibonnacci numbers implemented in
   the WHILE language */

write "Input a number ";
read n;
x := 0;   // start values
y := 1;
while n > 0 do {
  temp := y;
  y := x + y;
  x := temp;
  n := n - 1  // decrement counter
};
write "Result ";
write y
```

Figure 1: Fibonacci program in the WHILE language.

```
start := 1000;   // start value
x := start;
y := start;
z := start;
while 0 < x do {
 while 0 < y do {
  while 0 < z do { z := z - 1 };
  z := start;
  y := y - 1
 };
 y := start;
 x := x - 1
};
```

Figure 2: The three-nested-loops program in the WHILE language. Usually used for timing measurements.