Handout 5

Whenever you want to design a programming language or implement a compiler for an existing language, the first task is to fix the basic "words" of the language, like what are the keywords or reserved words of the language, what are permitted identifiers, numbers and so on. One convenient way to do this is, of course, to use regular expressions. In this course we want to take a closer look at the WHILE-language. This is a simple imperative language consisting of arithmetic expressions, assignments and loops only. For example the Fibonacci program can be written in this language as follows

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

The keywords in this language will be

$$\texttt{while}, \texttt{if}, \texttt{then}, \texttt{else}, \texttt{write}, \texttt{read}$$

In addition we will have some typical operators, like `<`, `>`, `:=` and so on; numbers and strings (which we however ignore for the moment). We also need to specify what the "white space" is in our programming language as well as what comments should look like. We might specify the regular expressions for our language roughly as follows

$$\begin{array}{lcl} \mathit{KEYWORD} & := & \texttt{while} + \texttt{if} + \texttt{then} + \texttt{else} + \ldots \\ \mathit{IDENT} & := & \mathit{LETTER} \cdot (\mathit{LETTER} + \mathit{DIGIT} + \_)^* \\ \mathit{OP} & := & \texttt{:=} + \texttt{<} + \ldots \\ \mathit{NUM} & := & \mathit{DIGIT}^+ \\ \mathit{WHITESPACE} & := & \text{" "} + \backslash\texttt{n} \end{array}$$

with the usual meanings for the regular expressions *LETTER* and *DIGIT*.

The problem we have to solve is given a string of our programming language, which regular expression matches which part of the string. For example the input string

$$\texttt{if}\;\texttt{true}\;\texttt{then}\;\texttt{x + 2}\;\texttt{else}\;\texttt{x + 3}$$

needs to be recognised as

```
i f ␣ t r u e ␣ t h e n ␣ x + 2 ␣ e l s e ␣ x + 3
```

Since `if` matches the *KEYWORD* regular expression, ␣ is a whitespace and so on. This process of separating an input string into components is often called *lexing* or *scanning*. It is usually the first phase of a compiler. Note that the separation into words cannot, in general, be done by looking at whitespaces: while `if` and `true` are separated by a whitespace, the components in `x+2` are not. Another reason for recognising whitespaces explicitly is that in some languages, for example Python, whitespace matters. However in our small language we will eventually filter out all whitespaces and also comments.

Lexing will not just separate the input into its components, but also classify the components, that is explicitly record that `it` is a keyword, ␣ a whitespace, `true` an identifier and so on. But for the moment we will only focus on the simpler problem of separating a string into components. There are a few subtleties we need to consider first. For example if the input string is

```
i f f o o ␣ …
```

then there are two possibilities: either we regard the input as the keyword `if` followed by the identifier `foo` (both regular expressions match) or we regard `iffoo` as a single identifier. The choice that is often made in lexers to look for the longest possible match, that is regard the input as a single identifier `iffoo` (since it is longer than `if`).