Inlab Assignment: Little Languages

Sriram Srinivasan

October 12, 2022

Introduction

The phrase *little languages* was introduced by Jon Bentley, who used to write an enormously popular column for the Communications of the ACM, which were then later compiled into two little books called Programming Pearls[1]. The books are a delightful assortment of little essays covering everything from language processors to algorithms to table design. They are a must read for all programmers.

Also known by the much stuffier name "Domain Specific Languages"(DSLs)

[1]

Little languages are tools that programmers build for themselves, just as skilled craftspeople make tools to make their lives easier. This assignment helps you create a language processor to translate a piece of text written in an algorithm style to that required by the `clrscode3e` LaTeX package, written for and exemplified by the popular book "Introduction to Algorithms"[2].

[2]

Problem Statement [30 marks]

Write a translator algo.py that is invoked as follows. Please see the attached resources.tar.gz file for skeleton code, where you need to replace TODO sections with your own. The only bit of knowledge you need to have is regular expressions and to use Python's `re` package. This is a quick and dirty approach. You'll eventually learn how to create proper interpreters and compilers for more complicated languages in the next semester.

```
> python3 algo.py sample.algo > sample.tex
```

To test your implementation, run the following in the same folder and make sure all tests pass

```
> python3 testalgo.py
```

Figures 1 and 2 show the input and the generated output. The generated file is ready to be included in your main document using a simple `\include`. Be sure to include `usepackage clrscode3`. You can verify that the typeset output should look figure 3.

```
proc test-algo(A, B)
    for j <- 2 to A.length do
        // Insert $A[j]$ into $A[1 .. j-1]$
        key <- A[j]
        i <- j - 1
        while i > 0 and A[i] > key do
            A[i+1] <- A[i]
            i <- i - 1
        end
        A[i+1] <- key
    end
    if x == 3 then do
        {{Do nothing}}
    end
end
```

Figure 1: Input: sample.algo

```
\begin{codebox}
\Procname{\proc{test-algo}(\id{A}, \id{B})}
\zi    \For \id{j} $\leftarrow$ 2 \To \attrib{A}{length} \Do
\zi        \Comment  Insert \id{A}[\id{j}] into \id{A}[1 $\twodots$ \id{j}$-$1]
\zi        \id{key} $\leftarrow$ \id{A}[\id{j}]
\zi        \id{i} $\leftarrow$ \id{j} $-$ 1
\zi        \While \id{i} $>$ 0 \id{and} \id{A}[\id{i}] $>$ \id{key} \Do
\zi            \id{A}[\id{i}$+$1] $\leftarrow$ \id{A}[\id{i}]
\zi            \id{i} $\leftarrow$ \id{i} $-$ 1
\zi        \End
\zi        \id{A}[\id{i}$+$1] $\leftarrow$ \id{key}
\zi    \End
\zi    \If \id{x} $\isequal$ 3 \Then \Do
\zi        {{\id{Do} \id{nothing}}}
\zi    \End
\zi\End
\end{codebox}
```

Figure 2: Output: sample.tex

```
TEST-ALGO($A$, $B$)
    for $j \leftarrow 2$ to $A.length$
        // Insert $A[j]$ into $A[1\,..\,j{-}1]$
        $key \leftarrow A[j]$
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > key$
            $A[i{+}1] \leftarrow A[i]$
            $i \leftarrow i - 1$

        $A[i{+}1] \leftarrow key$

    if $x == 3$
            Do nothing
```

Figure 3: Output: sample.tex

Algo language translation

The example input shows all the features of the language. These are the salient features.

Each line is of the form: *content text* `//` *comment text*

Comments are optional. The content part is treated as if it is in math mode by default, and the comment part is treated as if it is in text mode by default. In the latter, if there are embedded ... math mode fragments, you must treat them as shown below.

In all the fragments of math text (embedded in comments, or the content part), all words are either keywords to be replaced with the appropriate equivalent (see `keywordsToTeX` in `testalgo.py`), or else are normal identifiers, so they are surrounded by ’\Id{}’ . That is, `abc`’ becomes ’`\Id{abc}`’, but ’`for`’ becomes `’\For`’, since it is a keyword.

Field access patterns get replaced by ’`\attrib`. For example, `foo.bar.baz` becomes ’`\attribbb{foo}{bar}{baz}`’ (where the number of ’`b`’s after `attrib` represents the number of dots). Operators are translated to their LaTeX equivalents (see `testalgo.py` for the list). For example, ’>=’ becomes `’$\geq $’` and so on.