INF3110/4110—Mandatory Exercise 2

To be delivered 10.11.2008

1 Introduction

Working with functional and object-oriented programming is quite different. In order to experience some of these differences, you are going to extend the GSLAN language into GSLAN2 and introduce new formatting into your text formatter of mandatory exercise 1. In addition you have to write about your experiences with extending it.

2 Extending GSLAN into GSLAN2

This is the grammar of GSLAN.

```
<prog> ::= <exp>
<ifelse> ::= if <boolexp> then <exp> else <exp> endif
<assign> ::= <var> := <exp>
<var> ::= a | b | c
<boolexp> ::= <exp> = <exp>
<exp> ::= ( <exp> {, <exp>}* ) | <assign> | <print> | <ifelse> |
                          <number> | <var> | <exp> + <exp>
<print> ::= print(<exp>)
```

You are going to extend GSLAN with simple functions and variable references.

Functions in GSLAN2 are named f, g or h. A function is defined with the function keyword. The functions take one argument, and returns the value of the expression. A call to a function not defined should cause an error. You do not have to support recursion in your implementation. Here is the grammar for introducing functions into GSLAN.

```
<exp> ::= function <fname> ( <arg> ) <exp> endf
<arg> ::= x | y | z
<exp> ::= <fname> ( <exp> )
<fname> ::= f | g | h
```

References in GSLAN2 are called p, q and s. A reference point to the variable it is assigned to point to. Changes to the reference is to the variable it was assigned to. If an unassigned reference is accessed, an error should be caused. References are introduced with the following grammar:

```
<ref> ::= p | q | s
<assign> ::= <ref> := <var>
<exp> ::= <ref>
```

Implement in your ML and Java versions of the GSLAN interpreter. Here is an example written i GSLAN2. Notice that the scope of x is inside g. This program should output 3.

```
(
        a := 0,
        p := a,
        function f(x) p+x endf,
        function g(x) (
                x := x+a,
                if x = 2 then f(x) else f(5) endif
        ) endf,
        a := p+1,
        print(c := g(p))
)
```

2.1 Implementations in ML and Java

Beyond what is specified in this exercise, you are allowed to make your own assumptions as long as they are made explicit in the comments in the code.

You should have enough knowledge from your implementation in exercise 1 to continue working on the problem for this exercise. You should extend the interpret function in ML, and the extended program should work with the interpret function in its interface to support functions.

3 Extending the text formatter

In the previous mandatory exercise you implemented a simple text-formatter in ML and Java. In this exercise you are going to add hyphenation to your formatting implementation. Consider the text from the previous exercise, now justified by your formatter.

```
The   longest   word   used   in
shakespeare's      works      is
Honorificabilitudinitatibus  the
word appears in "Love's Labour's
Lost".  The  word,  however, was
used   long  before  Shakespeare
used   it.   The   word   is  27
characters     long    and    is
pronounced
Hono-rifica-bili-tudi-nita-tibus
.
```

This is not very nice because of the long words. This looks much better:

```
The  longest word used in shake-
speare's works is Honorificabil-
```

```
itudinitatibus  the word appears
in  "Love's  Labour's Lost". The
word, however, was used long be-
fore  Shakespeare  used  it. The
word  is  27 characters long and
is  pronounced Hono-rifica-bili-
tudi-nita-tibus.
```

The rule for where to split a word which is sufficient for this exercise is splitting a word on a vocal-consonant boundary or a consonant-vocal boundary, but without causing either of the two parts to be 1 or less characters long. If the word already contains an hyphen, then a new hyphen is not needed.

3.1 Implementation in ML

In your ML implementation you have hopefully managed to perform the formatting and justification of the text-examples. You have something similar to these functions (where some of the minor functions are omitted).

```
val headWord = fn : string -> string
val tailWord = fn : string -> string
val listOfWords = fn : string -> string list
val splitLines = fn : int -> string list -> string list list
val justifyLines = fn : int -> string list list -> string list list
```

If you have not done it in the exact same way, no problem. The following hints are made in accordance with the functions above.

A suggested approach is to split up the words into 'hyphenation'-parts. Which are recombined when splitting into lines, so the hyphens are correctly placed.

```
val listOfHyphenationParts = fn : string list -> string list
```

Which takes a list of words as input. How to structure the different parts and the recombination is up to you.

3.2 Implementation in Java

In your java implementation you now have something similar to a class Character, which is the class for a character, Word, which is a class for any word, Space, which is the class for a space in between words, Line, which is a class for a line, and Text, which is the whole text.

In order to implement hyphenation, introduce a class HyphenationPart inside the Word class. The word class is then responsible for handling the splitting of the word. The rest is up to you.

4 Requirements and deliveries

For the ML implementations you are not allowed to use reference types (reference cells). The java implementations must be object-oriented. You also have to write one to two pages about your experiences. This document should include what could and could not be reused from the previous exersice and why. Email the files listed below to martifag@ifi.uio.no.

- Gslan2.sml
- Gslan2.java
- TextProcessor2.sml
- TextProcessor2.java
- MyExperiences.pdf

Good Luck!