ADVANCED PROGRAMMING

Assignment

January 2014

The assignment should be submitted by e-mail by **Jan 14th (tne deadline time of the submission is 20:00)** to giangi@di.unipi.it and cisterni@di.unipi.it with the subject prefix **[AP-Assignment]**. You can solve exercises using a programming language of choice among: C/C++, Java, C#, F#, Javascript. The submission must include all required files for compiling and executing the proposed solution. It is allowed to discuss the problem among the class but the proposed solution should be individual.

Exercise 1

Design and implement a program that reads a single string of characters from the standard input stream. The string consists of digits, blanks, and plus signs. The string fits on a single line (it ends with either the "newline" character or the "end of file").

A valid input string will consist of a series of one or more unsigned integers separated by plus signs. Blanks are not allowed inside integers. Any two successive integers must be separated by a plus sign; each plus sign must appear between two successive integers. Blanks are allowed before and after integers and plus signs. Your program should generate a descriptive error message if it detects an invalid input string.

Two command-line flags will determine the behavior of your program.

- The '-i' flag causes the program to interpret the input string to produce the sum of the integers, and to print that sum to the standard output stream.
- The '-c' flag causes the program to create a file in the current working directory. That file, named "result.c" will contain C code to perform the computation specified by the input string and print the result to the standard output stream in the same format as the '-i' option.

The two flags are not mutually exclusive. If invoked with both flags, the program should produce both the interpreted result on the standard output stream and the result.c file in the current working directory. Your program should generate an error message if it detects invalid command-line flags or no command-line flags. The error message should both list allowable command-line flags and report invalid command line flags, if any.

Exercise 2

Design and implement a program that reads a single text file from the current working directory. The file contains the definition of an expression (actually a function) written in accordance with the following abstract syntax

e ::= int | ide | | sum(e, e) | **let** ide = exp **in** e | **fun** (ide) -> e | **app** e e

where int and ide are defined as usual. A valid expression consists of the declaration of a non-anonymous function. For example

let succ = fun(x) -> sum(x, 1) in succ(5)

is a valid expression. The expression

fun(x) -> sum(x, 1)

even though syntactically well written is not a valid expression. Your program should generate a descriptive error message if it detects an invalid expression.

The command-line flags will determine the behavior of your program. The '-compile -java' flag causes the program to create a file in the current working directory. That file, named "function.java" will contain the java code of the class definition containing as public method the function specified by the input expression. Similarly, the '-compile –csharp' causes the program to create the C# class in the current working directory.

Design the main (java or C#) program to operate with the function class.

Once implemented the language described above , implement a pre-processor featuring a quotation mechanism similar to PHP/ASP using the <@ @> markers. Inside the markers it is possible to define functions with the above syntax assuming that the result of the evaluation should replace the quotation.

For instance:

Succ of 5 is <@ let succ = fun(x) -> sum(x, 1) in succ(5) @>

Should generate the Java/C# program that prints

Succ of 5 is 6

Exercise 3

Microsoft's F# language provides several facilities for the building and high level manipulation of computations and meta-computations. One of these, workflows, allows libraries to define domain-specific sublanguages for particular kinds of computation. Using this, the Async module gives a way to write code that can execute asynchronously when necessary, without needing to explicitly describe any threads or communication. Actions that might potentially block or be long-running will automatically happen in the background, with their results retrieved as they arrive. Discuss in details what happens with the following example:

```
open System.Net
open Microsoft.FSharp.Control.WebExtensions

let urlList = [ "University of Pisa", "http://www.unipi.it/"
                "CS Dept", "http://www.di.unipi.it/"
                "Bing", "http://www.bing.com"
              ]

let fetchAsync(name, url:string) =
    async {
        try
            let uri = new System.Uri(url)
            let webClient = new WebClient()
            let! html = webClient.AsyncDownloadString(uri)
            printfn "Read %d characters for %s" html.Length name
        with
            | ex -> printfn "%s" (ex.Message);
    }

let runAll() =
    urlList
    |> Seq.map fetchAsync
    |> Async.Parallel
    |> Async.RunSynchronously
    |> ignore

runAll()
```

Is this a way to express parallel computations? How would you define the scoping of the async construct (tip: look after computation expressions that are used to implement async). Are there examples of similar mechanisms in other languages?

Exercise 4

Let us consider the Clojure programming language (http://www.clojure.org). Give a short definition of the characteristics of the Clojure programming language (i.e., scoping, memory management, etc.).