Paradigmas de Programación
Práctica 8

Objetivo de la práctica y descripción de la aplicación

El objetivo de esta práctica es el de construir una calculadora de punto flotante con variables. La calculadora se incia con el comando `calc`. Tras ejecutar este comando, la calculadora muestra un mensaje de bienvenida y un prompt, ante el cual se permite la introducción por teclado de los siguientes tipos de instrucciones y comandos:

- **Instrucciones de evaluación de expresiones aritméticas:**

  Estas expresiones constan de números de punto flotante ligados con operadores aritméticos. La sintaxis permitida para los números de punto flotante es la misma que utiliza Ocaml. Los operadores permitidos son `+`, `-`, `*` y `/`. La precedencia de estos operadores es también la utilizada por Ocaml, y se pueden utilizar los paréntesis `(` y `)` para variarla.

  Ante una expresión de este tipo, la calculadora la evalúa, muestra por pantalla el resultado de dicha evaluación, y devuelve el prompt, de forma que el usuario pueda introducir otra instrucción.

  La expresión podría involucrar a variables previamente definidas mediante instrucciones de asignación, las cuales se explican en el siguiente apartado. Debido a esto, toda evaluación de una expresión debe ser hecha teniendo en cuenta el estado interno de la calculadora en ese momento, es decir, teniendo en cuenta las asignaciones de valores a variables que el usuario pueda haber efectuado previamente.

  La expresión podría hacer uso también de algunas rutinas predefinidas sobre números de punto flotante, para lo cual la calculadora debe tener integrada una pequeña librería de funciones de este tipo, como, por ejemplo, `!abs` (valor absoluto), `!sqrt` (raíz cuadrada), `!exp` (exponencial), `!log` (logaritmo natural), `!round` (redondeo), etc. Como se puede observar, el uso de una función requiere que su nombre vaya precedido del caracter `!`. Al igual que en Ocaml, la aplicación de funciones constituye la operación de máxima precedencia.

- **Instrucciones de asignación de variables:**

  Una instrucción de este tipo consta de un nombre de variable, seguido del signo igual (=) y de una expresión aritmética. La sintaxis permitida para los nombres de variables es la misma que la de los identificadores en Ocaml.

  Ante una instrucción de asignación, la calculadora evalúa la expresión aritmética, imprime por pantalla el nombre de la variable, el signo igual, el resultado de la evaluación, y el prompt, y además modifica el estado interno, anulando cualquier asignación previa de valores a dicha variable, y añadiendo la asignación actual.

- **Comandos especiales:**

  Al igual que con las funciones, el uso de un comando requiere que su nombre vaya precedido por un caracter. En este caso, por el caracter #. La calculadora dispone de estos tres comandos:

  - **Listado de variables:** Para visualizar el estado interno de la calculadora, se utiliza el comando `#var`. Ante este comando, la calculadora imprime por pantalla el nombre y el valor de cada una de las variables definidas en ese momento.

- **Listado de funciones:** Para visualizar la libería interna de funciones de la calculadora, se utiliza el comando `#fun`. Ante este comando, la calculadora imprime por pantalla el nombre y la descripción de cada una de las funciones disponibles.

- **Finalización del programa:** Para terminar la ejecución del programa se utiliza el comando `#fin`. Ante este comando, la calculadora imprime un mensaje de despedida y devuelve el control al sistema operativo.

Además de esto, la calculadora controla en todo momento los errores tipográficos ("léxicos"), sintácticos y contextuales que puedan aparecer a la hora de introducir las instrucciones y comandos. Ante este tipo de fenómenos, la calculadora imprime el correspondiente mensaje de error, y devuelve el prompt de manera que el usuario pueda introducir de nuevo la instrucción o el comando.

El siguiente **ejemplo de ejecución** muestra el comportamiento que debe tener la calculadora:

```
$ ./calc
Calculadora de punto flotante con variables...
>> 1 + 3/4
1.75

>> x = 3e-4 / 2
x = 0.00015

>> (x + 3) * 2
6.0003

>> y = 2 * x
y = 0.0003

>> z + 1
Variable z no asignada

>> #var
y = 0.0003
x = 0.00015

>> !round !sqr 34
Función sqr no implementada

>> #func
Comando func no implementado

>> #fun
abs : valor absoluto
sqrt : raíz cuadrada
exp : exponencial
log : logaritmo natural
round : redondeo

>> !round !sqrt 34
6.

>> 3,4
Error lexico

>> 3 4
Error sintactico

>> #fin
... Adiós !!!
```

Instrucciones para la realización de la práctica

Descarga de ficheros

Descargue todos los ficheros que se proporcionan junto con este enunciado como material de esta práctica y ubíquelos en un mismo directorio de trabajo.

Diccionario de variables

A partir del estudio del fichero de interfaz `diccionario.mli`, escriba un nuevo fichero `diccionario.ml` que permita construir un módulo para el manejo del contexto interno de variables de la calculadora. Para ello, implemente el tipo del diccionario, defina un valor inicial de ese tipo, e implemente las siguientes funciones (respetando, claro está, los tipos de la interfaz):

- Una función de consulta de valores (necesaria para la ejecución de las instrucciones de evaluación de expresiones aritméticas). Esta función debe activar la excepción `Variable_no_asignada` cuando se realice una consulta sobre una variable que no ha sido definida.
- Una función de asignación de valores (necesaria para la ejecución de las instrucciones de asignación de variables).
- Una función que muestre por pantalla todo el contenido del diccionario de variables (necesaria para la ejecución del comando `#var`).

Librería de funciones

A partir del estudio del fichero de interfaz `libreria.mli`, escriba un nuevo fichero `libreria.ml` que permita construir un módulo para el acceso a la librería interna de funciones de la calculadora.

Al contrario de lo que ocurre con el diccionario de variables, que es dinámico, la librería de funciones es estática y no varía a lo largo de la ejecución de las instrucciones y comandos que se escriben en el lazo interactivo de la calculadora. Por tanto, quizás no sea necesaria la implementación de un tipo de dato específico para dicha librería. En todo caso, su exportación en el fichero de interfaz es estrictamente innecesaria. Dicho de otro modo, la librería de funciones debe ser un valor interno del módulo y el fichero de interfaz del módulo no debe modificarse.

Eso sí, el módulo debe implementar los mecanismos necesarios para el acceso a las funciones y a sus descripciones, así como la función que muestre por pantalla la correspondiente ayuda sobre la utilización del contenido de la librería. En particular, la función de acceso debe activar la excepción `Funcion_no_implementada` cuando se realice un intento de uso de una función que no está definida en la librería.

El contenido explícito de la librería de funciones es de diseño libre, pero deben existir al menos las cinco operaciones mencionadas en la descripción de la aplicación (`!abs`, `!sqrt`, `!exp`, `!log` y `!round`).

Evaluación de expresiones aritméticas

A partir del estudio del fichero de interfaz `expr_arit.mli`, escriba un nuevo fichero `expr_arit.ml` que permita construir un módulo para la evaluación de expresiones aritméticas.

Obsérvese que la función de evaluación de una expresión aritmética necesita hacer uso del diccionario de variables que define el estado interno de la calculadora, así como también de su librería interna de funciones.

Ejecución de instrucciones

A partir del estudio del fichero de interfaz `instruccion.mli`, escriba un nuevo fichero `instruccion.ml` que permita construir un módulo para la ejecución de instrucciones de evaluación de expresiones aritméticas, de instrucciones de asignación y de comandos especiales.

Obsérvese que la función de ejecución de una instrucción necesita hacer uso del diccionario de variables que define el estado interno de la calculadora, y devuelve un nuevo diccionario como resultado, a no ser que:

- Se esté ejecutando un comando o una instrucción que no es de asignación (en cuyo caso se devolverá el mismo diccionario).
- O a no ser que se esté intentando lanzar un comando que no sea `#var`, `#fun` o `#fin` (en cuyo caso se activará la excepción `Comando_no_implementado`).
- O a no ser que se esté ejecutando el propio comando `#fin` (en cuyo caso se activará la excepción `Fin_de_ejecucion`).

Analizadores léxico y sintáctico

Los ficheros `scanner.mll` y `parser.mly` implementan los módulos de análisis léxico y sintáctico de las órdenes que se escriben ante el prompt de la calculadora. Al igual que el resto de ficheros proporcionados, no necesitan ser modificados. Y además, en este caso, la construcción de dichos módulos no precisa ficheros adicionales.

Así pues, lo único que es necesario conocer para su correcto uso es que estos módulos implementan sendas funciones de análisis léxico (`Scanner.token`) y análisis sintáctico (`Parser.s`). Estas funciones, convenientemente enlazadas y aplicadas sobre un valor de tipo `Lexing.lexbuf` (que podría construirse a partir de la línea que teclea el usuario ante el prompt de la calculadora), son capaces de devolver un valor de tipo `Instruccion.instruccion`. La expresión en cuestión sería ésta:

```
Parser.s Scanner.token (Lexing.from_string (read_line ()))
```

o bien simplemente

```
s token (from_string (read_line ()))
```

si todos los módulos implicados han sido previamente abiertos.

Ahora bien, obviamente, la orden tecleada por el usuario podría contener errores léxicos o sintácticos, en cuyo caso estas funciones no devolverían ningún valor, sino que activarían o bien la excepción `Scanner.Error_lexico`, o bien la excepción `Parsing.Parse_error`, respectivamente.

Lazo interactivo

Por último, escriba un nuevo fichero `calc.ml` que implemente el lazo interactivo de la calculadora, para la petición y ejecución de instrucciones y comandos.

Este fichero debe incluir una función que realice este proceso, la cual debe comenzar su ejecución con un diccionario de variables inicialmente vacío, debe escribir en pantalla el mensaje de bienvenida y el prompt, y seguidamente debe leer las líneas que el usuario introduce a través de la entrada estándar.

Mediante las funciones de análisis mencionadas en el apartado anterior, cada línea es transformada en una instrucción. Posteriormente, cada instrucción se ejecuta, y como resultado de dicha ejecución se obtiene un nuevo diccionario de variables que constituirá el nuevo estado interno de la calculadora. Como último paso de la iteración actual del lazo, se escribe de nuevo el prompt y se espera a la introducción de una nueva línea.

En todo momento, el lazo debe vigilar la posible activación de todas las excepciones que se han descrito en todos los módulos previos. Es más, en caso de activarse alguna de ellas, dicha excepción debe ser interceptada, y su correspondiente manejador de error debe escribir en pantalla un mensaje que informe adecuadamente sobre lo ocurrido, y debe retomar la iteración del lazo sin variar el estado interno de la calculadora.

Ahora bien, en el caso particular de que se active la excepción `Fin_de_ejecucion`, el programa debe escribir el mensaje de despedida y finalizar.

Compilación de la aplicación

Compile toda la aplicación y obtenga el ejecutable `calc` lanzando el comando `make all` ante el prompt del sistema operativo, para lo cual debe estar presente en el directorio de trabajo el fichero `Makefile` proporcionado.

Ejecución de la aplicación

Ejecute la aplicación lanzando el comando `calc` y verifique su correcto comportamiento probando, al menos, el mismo conjunto de instrucciones que aparece en el ejemplo de ejecución del presente enunciado.

Planificación temporal

El espacio reservado en la planificación temporal de la asignatura para la realización de esta práctica es de dos semanas.