Paradigmas de Programación

Práctica 12

Calculadora de punto flotante con variables

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 inicia con la orden `calc`. Tras ejecutar esta orden, la calculadora muestra un mensaje de bienvenida y un prompt, ante el cual se permite la introducción por teclado de los siguientes tipos de comandos:

- **Evaluación de expresiones aritméticas**

  Estas expresiones constan de números de punto flotante ligados con operadores aritméticos. Dichos números se pueden introducir en notación científica. Así pues, su sintaxis está formada por: signo (opcional), parte entera (obligatoria, al menos un dígito), decimales (opcional, al menos el punto) y exponente (opcional, al menos `e` o `E` y un dígito). En notación BNF, sería como sigue:

  ```
  <float> ::= [+|-] <digit> {<digit>} [. {<digit>}] [(e|E) [+|-] <digit> {<digit>}]
  <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  ```

  Los operadores permitidos son `+`, `-`, `*`, `/`, `%` (para el módulo) y `^` (para la potencia). La precedencia (de menor a mayor) y asociatividad de estos operadores es como sigue:

  - Operadores `+` y `-` (asocian por la izquierda)
  - Operadores `*`, `/` y `%` (asocian por la izquierda)
  - Operador `^` (asocia por la derecha)

  Se pueden utilizar los paréntesis `(` y `)` para variar estas precedencias y asociatividades.

  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 otro comando. La expresión podría involucrar a variables previamente definidas mediante comandos de asignación, los cuales se explican en el siguiente apartado. Debido a esto, toda evaluación de una expresión debe ser hecha teniendo en cuenta el estado interno (contexto) 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 tiene integrada una pequeña librería de funciones de este tipo, como, por ejemplo, `sqrt` (raíz cuadrada), `exp` (exponencial), `ln` (logaritmo natural), `round` (redondeo), etc. La aplicación de funciones constituye la operación de máxima precedencia.

- **Asignación de variables**

  Un comando de este tipo consta de un nombre de variable, seguido del signo igual (`=`) y de una expresión aritmética. Los nombres de variables deben comenzar con una letra, que puede ir seguida por cualquier número de letras, dígitos y guiones bajos (_). Todas las letras deben ser minúsculas.

Ante un comando de asignación, la calculadora evalúa la expresión aritmética, escribe por pantalla el nombre de la variable, el signo igual, el resultado de la evaluación, y el prompt, y además modifica el estado interno (contexto), anulando cualquier asignación previa de valores a dicha variable, y añadiendo la asignación actual.

- **Comandos especiales**

  La calculadora dispone de un comando especial, el comando `quit`, para la finalización del programa. Ante este comando, la calculadora escribe un mensaje de despedida y devuelve el control al sistema operativo.

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

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

```
$ ./calc
Floating point calculator...
>> 1 + 3/4
1.75

>> round (sqrt (34)) * 2
12.

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

>> (x + 3) * 2
6.0003

>> y = 2 * x
y = 0.0003

>> z + 1
Variable z not defined

>> abs (x)
Function abs not defined

>> 3,4
Lexical error

>> 3 4
Syntax error

>> quit = 5
Syntax error

>> quit
... bye!!!
```

Instrucciones para la realización de la práctica

Para la correcta realización de esta práctica, acometa por orden los siguientes apartados.

- **Descarga de ficheros**

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

- **Estado interno de la calculadora (contexto de variables)**

  Estudie el fichero de interfaz `context.mli` y complete el fichero `context.ml` de forma que permita construir un módulo para el manejo del contexto interno de variables de la calculadora. Para ello, implemente el tipo del contexto, 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 evaluación de las expresiones aritméticas). Esta función debe activar la excepción `No_binding` cuando se realice una consulta sobre una variable que no ha sido definida.
  - Una función de asignación de valores (necesaria para la ejecución de los comandos de asignación de variables).

- **Librería de funciones**

  Estudie el fichero de interfaz `lib.mli` y complete el fichero `lib.ml` de forma 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 contexto de variables, que es dinámico, la librería de funciones es estática y no varía a lo largo de la ejecución de los comandos que se escriben en el lazo interactivo de la calculadora. Así pues, su exportación en el fichero de interfaz es totalmente 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.

  Como sugerencia, cabe decir que el módulo implementado anteriormente para el soporte del contexto variables podría ser utilizado también aquí para el soporte de la librería de funciones. Se trataría únicamente de permitir que el usuario acceda a las funciones implementadas, pero que no pueda modificarlas ni añadir ninguna otra.

  Sea como sea, este módulo debe implementar los mecanismos necesarios para el acceso a las funciones. Y, en particular, la función de acceso debe activar la excepción `Function_not_defined` 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 cuatro operaciones mencionadas en la descripción de la aplicación (`sqrt`, `exp`, `ln` y `round`).

- **Evaluación de expresiones aritméticas**

  Estudie el fichero de interfaz `arith.mli` y complete el fichero `arith.ml` de forma que permita construir un módulo para la evaluación de expresiones aritméticas.

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

- **Ejecución de comandos**

  Estudie el fichero de interfaz `command.mli` y complete el fichero `command.ml` de forma que permita construir un módulo para la ejecución de comandos de evaluación de expresiones aritméticas, de comandos de asignación y de comandos especiales.

  Observe que la función de ejecución de un comando necesita hacer uso del contexto de variables que define el estado interno de la calculadora, y devuelve como resultado un nuevo contexto, a no ser que:

  - Se esté ejecutando un comando que no es de asignación (en cuyo caso se devolverá el mismo contexto).
  - O a no ser que se esté ejecutando el comando `quit` (en cuyo caso se activará la excepción `End_of_program`).

- **Analizadores léxico y sintáctico**

  Los ficheros `lexer.mll` y `parser.mly` implementan los módulos de análisis léxico y sintáctico de los comandos que se escriben ante el prompt de la calculadora. Al igual que los ficheros de interfaz proporcionados, no necesitan ser modificados.

  Así pues, lo único que es necesario conocer para su correcto uso es que estos módulos implementan sendas funciones de análisis léxico (`Lexer.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 `Command.command`.

  La expresión en cuestión sería ésta:

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

  o bien simplemente

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

  si todos los módulos implicados han sido previamente abiertos.

  Eso sí, 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 `Lexer.Lexical_error`, o bien la excepción `Parsing.Parse_error`, respectivamente.

- **Lazo interactivo**

  Por último, complete el fichero `main.ml` de forma que implemente el lazo interactivo de la calculadora, para la petición y ejecución de comandos.

  Este fichero debe incluir una función que realice este proceso. Una vez que el mensaje de bienvenida haya sido escrito en pantalla, esta función debe comenzar su ejecución con un contexto de variables inicialmente vacío, debe presentar el prompt, y seguidamente debe leer las líneas que el usuario introduce a través de la entrada estándar.

  Mediante las funciones de análisis léxico y sintáctico mencionadas en el apartado anterior, cada línea es transformada en un comando. Posteriormente, cada comando se ejecuta, y como resultado de dicha ejecución se obtiene un nuevo contexto 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 `End_of_program`, 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.