Lambdas (Funciones anonimas)

Ernesto Rodriguez

Universidad del Itsmo

erodriguez@unis.edu.gt

Motivación

- El calculo-λ es el lenguaje de programación más antiguo que existe
- Fue la primera definición de la *computabilidad*
- Del calculo-λ se derivo la corriente de *programación funcional*
- La programación funcional ha agarrado mucha tracción los ultimos años.
- Como dice Phillip Wadler[2]: "Java por fin llego donde Church estaba en 1930."

Introducción

- Una lambda no es nada más y nada menos que una función.
- Tradicionalmente, los lenguajes siempre han separado las funciones y los valores (eg. metodos y propiedades)
- Los lambdas permiten utilizar funciones como si fueran valores
- Esto significa que una función puede almacenarce en una variable o puede pasarse como parametro a un metodo.
- La brecha entre funciones y valores se rompe al tener un lenguaje con lambdas.
- De hecho, un lenguaje de programación puede consistir solamente de lambdas, el resto son vanidades. El lenguaje Haskell[1], se apega mucho a esta filosofia.

Definición de Lambdas

El formato para definir lambdas en C# es: $(\texttt{arg}_1, \ldots, \texttt{arg}_n) \Rightarrow \{[cuerpo]\}$ en donde:

- $\texttt{arg}_1, \ldots, \texttt{arg}_n$ son variables correspondientes a los parametros de la función.
- [*cuerpo*] el el codigo que ejecuta la funcion.

Observaciones:

- Es permitido omitir los parentesis de los parametros cuando la función solamente tiene un parametro.
- Es permitido omitir los corchetes () que rodean el cuerpo de la función si dicho cuerpo solo tiene una linea.

La clase *Func*

- La clase *Func* se utiliza para definir variables de tipo *Funcion* (ie. lambdas)
- Hay varias definiciones de esta clase, según la cantidad de parametros y el tipo de retorno.
- Se utiliza de la siguiente manera: $\texttt{Func}\langle \texttt{Arg}_1, \ldots \texttt{Arg}_n, \texttt{Res} \rangle$ donde $\texttt{Arg}_1, \ldots, \texttt{Arg}_n$ corresponden a los parametros de la función y `Res` al resultado de la función.
- Por ejemplo, una función que suma dos numeros enteros, se declararia como "$\texttt{Func}\langle \texttt{int}, \texttt{int} \rangle$ *sumar*"
- Ver el metodo *Ejemplo_Func* del archivo Program.cs

La clase *Action*

- Tiene la misma funcionalidad basica que *Func*
- Difiere que se utiliza exculsivamente para funciones *impuras*.
- Esto quiere decier que no tiene tipo de retorno
- Por lo cual su utilización es: "$\texttt{Action}\langle \texttt{Arg}_1, \ldots, \texttt{Arg}_n \rangle$" en donde "$\texttt{Arg}_1, \ldots, \texttt{Arg}_n$" son los tipos de los parametros.
- Su cuerpo no necesita tener un `return` (aunque si lo puede tener.)

Funciones de orden superior

- Ya vimos que es posible utilizar funciones como valores, asignarlos a variables, incluso colocarlos en arreglos!
- No debe ser extraño que también pueden ser pasadas como parametros a otras funciones o metodos.
- Métodos y funciones que aceptan funciones en sus parametros se conocen como *funciones de orden superior*.
- Ver las funciones *Buscar*, *MaxBy* y *Filtrar* que se encuentran en el archivo Program.cs

Delegates: Motivación

- Los tipos `Func` y `Action` no permiten exponer metadata de forma facil.
- Si se cambia la firma de una funcion, esta debe ser cambiada en todos los sitios donde se utiliza.
- A menudo no es intuitivo saber que función lleva a cabo cada parametro de una función o acción anonima.
- Es dificil documentar el proposito de cada función anonima o acción

Delegates: Utilización

- Un *delegate* es una firma de un metodo, que puede utilizarse como tipo.
- Permiten asignarle nombre a cada uno de los parametros de dicha firma, mejorando la documentación.
- Los *delegates* pueden tener un nombre, lo cual facilita entender el proposito de dicho *delegate*.
- Algúnas caracteristicas de C# (como *eventos*) son más amigables de utilizar con delegates.
- De hecho, `Func` y `Action` son *delegates*.
- Ejemplos de la utilización de *delegates* se pueden encontrar en el archivo Program.cs

Referencias


Haskell.org.
Haskell.
https://www.haskell.org/.


Phillip Wadler.
Propositions as types.
https://www.youtube.com/watch?v=aeRVdYN6fE8.