Paradigma Funcional - SML

Alumno: Solano Meza Angel Daniel

Clase: Paradigmas de Programación

Introducción

Este reporte detalla el contenido de 10 archivos `.sml` en el repositorio, siguiendo la guía de SML Tour.

Desarrollo

1. `basic-data-types.sml`

Este archivo introduce los tipos de datos básicos en SML, como enteros, reales, caracteres, cadenas y booleanos.

Código:

```
val x : int = 5
val y : real = 3.14
val c : char = #"a"
val s : string = "hello"
val b : bool = true
```

Explicación:

- `int`: Números enteros.
- `real`: Números con punto decimal.
- `char`: Caracteres individuales.
- `string`: Secuencias de caracteres.
- `bool`: Valores booleanos (`true` o `false`).

2. `chaining.sml`

Demuestra cómo encadenar operaciones utilizando el operador `;`.

Código:

```
val x = 1 + 2;
val y = x * 3;
val z = y - 4;
```

Explicación:
Cada expresión se evalúa secuencialmente. El resultado de `x` se usa en `y`, y el resultado de `y` se usa en `z`.

3. `cond-expr.sml`

Explica las expresiones condicionales `if-then-else`.

Código:

```
val x = 10
val result = if x > 5 then "Greater" else "Smaller"
```

Explicación:

- `if-then-else` se usa para tomar decisiones basadas en condiciones. Aquí, si `x` es mayor que 5, `result` será "Greater"; de lo contrario, será "Smaller".

4. `data-structures.sml`

Introduce listas, tuplas y registros, mostrando cómo manipular estas estructuras.

Código:

```
val lst = [1, 2, 3]
val tuple = (1, "hello", true)
val record = {name="Alice", age=30}
```

Explicación:

- `list`: Una secuencia ordenada de elementos.
- `tuple`: Una colección fija de elementos de diferentes tipos.
- `record`: Una colección de campos con nombre.

5. `deconstr.sml`

Muestra cómo deconstruir listas y tuplas usando patrones.

Código:

```
val (x, y) = (1, 2)
val hd::tl = [1, 2, 3]
```

Explicación:

- La deconstrucción permite extraer valores de estructuras complejas. `(x, y)` extrae los elementos de una tupla, y `hd::tl` separa la cabeza de una lista de su cola.

6. `exhaustive.sml`

Cubre el uso exhaustivo de patrones para garantizar que todos los casos posibles están cubiertos.

Código:

```
fun describe x =
  case x of
      1 => "one"
    | 2 => "two"
    | _ => "other"
```

Explicación:

- `case...of` garantiza que se cubran todos los posibles valores de `x`. El patrón `_` actúa como un comodín para cualquier otro valor.

7. `functions.sml`

Introduce funciones, incluyendo funciones anónimas y recursivas.

Código:

```
fun add (x, y) = x + y
val inc = fn x => x + 1
fun factorial n = if n = 0 then 1 else n * factorial (n-1)
```

Explicación:

- `fun` define una función nombrada.
- `fn` define una función anónima.
- `factorial` es una función recursiva que calcula el factorial de un número.

8. `functors.sml`

Explica cómo usar funtores, que son módulos parametrizados.

Código:

```
signature MONOID =
sig
  type t
  val zero : t
  val add : t * t -> t
end

functor MakeMonoid(M : MONOID) =
struct
  val zero = M.zero
```

```
  fun add (x, y) = M.add (x, y)
end
```

Explicación:

- `signature` define una interfaz para un módulo.
- `functor` toma un módulo que cumple con una firma y genera un nuevo módulo.

9. `let-expr.sml`

Demuestra el uso de `let...in...end` para definir enlaces locales.

Código:

```
fun sumList lst =
  let
    fun sumHelper ([], acc) = acc
      | sumHelper (x::xs, acc) = sumHelper (xs, x + acc)
  in
    sumHelper (lst, 0)
  end
```

Explicación:

- `let...in...end` permite definir variables y funciones en un ámbito limitado, mejorando la modularidad del código.

10. `modules.sml`

Introduce módulos y firmas para organizar y encapsular código.

Código:

```
signature STACK =
sig
  type 'a stack
  val empty : 'a stack
  val push : 'a * 'a stack -> 'a stack
  val pop : 'a stack -> 'a * 'a stack
end

structure Stack : STACK =
struct
  type 'a stack = 'a list
  val empty = []
  fun push (x, s) = x :: s
  fun pop s = (hd s, tl s)
end
```

Explicación:

- `signature` define una interfaz que especifica qué funciones y tipos debe tener un módulo.
- `structure` proporciona una implementación concreta para la firma definida.

Conclusión

Estos archivos cubren una amplia gama de conceptos en SML, desde tipos de datos básicos hasta módulos avanzados y funtores. Cada archivo proporciona ejemplos prácticos para entender mejor cómo usar estos conceptos en el desarrollo de software.