

函数式编程原理

实验一

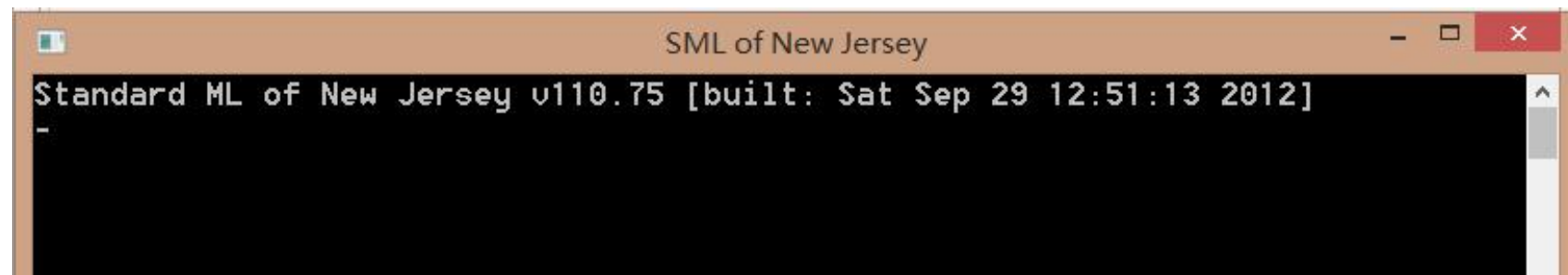
实验目的

- 了解函数式编程语言家族成员及其发展和特点
- 熟悉SML/NJ开发环境及使用
- 掌握SML基本语法和书写规则
- SML简单程序设计和程序编写

实验内容：

1. 完成函数式语言家族成员调研报告，内容可包括但不限于：庞大的函数式语言家族中有哪些成员？都由谁提出来的？各自有什么特征？没落和兴盛的原因？……
2. 安装SML/NJ开发环境，掌握其使用。
3. 编码实现相应功能。

实验提示



- 在'-'提示符下直接输入SML语句,以分号结束;
- 表达式计算的结果缺省赋值给变量"it";
- 文件的加载: `use <filename>;`
如 `use "d:\\sml\\test.sml";`
- 程序正确性检查: `val <return value> = <function> <argument value>`
如: `val 42 = eval [2,4]` 或 `42=eval[2,4]`

1. 在提示符下依次输入下列语句，观察并分析每次语句的执行结果。

- `3+ 4;`
- `3 + 2.0;`
- `it + 6;`
- `val it = "hello";`
- `it + " world";`
- `it + 5;`
- `val a = 5;`
- `a = 6;`
- `a + 8;`
- `val twice = (fn x => 2 * x);`
- `twice a;`
- `let x = 1 in x end;`
- `foo;`
- `[1, "foo"];`

2.函数sum用于求解整数列表中所有整数的和，函数定义如下：

```
(* sum : int list -> int          *)  
(* REQUIRES: true                  *)  
(* ENSURES: sum(L) evaluates to the sum of the integers in L. *)  
fun sum [ ] = 0  
  | sum (x ::L) = x + (sum L);
```

完成函数mult的编写，实现求解整数列表中所有整数的乘积。

```
(* mult : int list -> int          *)  
(* REQUIRES: true                  *)  
(* ENSURES: mult(L) evaluates to the product of the integers in L. *)  
fun mult [ ] =          (* FILL IN *)  
  | mult (x ::L) =      (* FILL IN *)
```

3.完成如下函数Mult: int list list -> int的编写,该函数调用mult 实现int list list中所有整数乘积的求解。

```
(* Mult : int list list -> int *)
```

```
(* REQUIRES: true *)
```

```
(* ENSURES: Mult(R) evaluates to the product of all the integers in the  
lists of R. *)
```

```
fun Mult [ ] = (* FILL IN *)
```

```
| Mult (r :: R) = (* FILL IN *)
```

4. 函数mult'定义如下，试补充其函数说明，指出该函数的功能。

```
(* mult' : int list * int -> int *)  
(* REQUIRES: true *)  
(* ENSURES: mult'(L, a) ... (* FILL IN *) *)
```

```
fun mult' ([ ], a) = a  
  | mult' (x :: L, a) = mult' (L, x * a);
```

利用mult'定义函数Mult' : int list list * int -> int，使对任意整数列表的列表R和整数a，该函数用于计算a与列表R中所有整数的乘积。该函数框架如下所示，试完成代码的编写。

```
fun Mult' ([ ], a) = (* FILL IN *)  
  | Mult' (r::R, a) = (* FILL IN *)
```


5. 编写递归函数square实现整数平方的计算，即 $\text{square } n = n * n$ 。

要求：程序中可调用函数double，但不能使用整数乘法（*）运算。

```
(* double : int -> int *)
```

```
(* REQUIRES: n >= 0 *)
```

```
(* ENSURES: double n evaluates to 2 * n.*)
```

```
fun double (0 : int) : int = 0
```

```
  | double n = 2 + double (n - 1)
```

6. 编写函数实现下列功能：

(1) `zip: string list * int list -> (string * int) list`

其功能是提取第一个string list中的第i个元素和第二个int list中的第i个元素组成结果list中的第i个二元组。如果两个list的长度不同，则结果的长度为两个参数list长度的最小值。

(2) `unzip: (string * int) list -> string list * int list`

其功能是执行zip函数的反向操作，将二元组list中的元素分解成两个list，第一个list中的元素为参数中二元组的第一个元素的list，第二个list中的元素为参数中二元组的第二个元素的list。

对所有元素L1: string list和L2: int list，`unzip(zip (L1, L2)) = (L1, L2)`是否成立？如果成立，试证明之；否则说明原因。