# 课堂练习1

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。

①输入：3+ 4;

输出：val it = 7 : int

常量默认为int形，运算得到的结果同样是int型；使用+中缀运算符得到的结果，在交互式程序设计下直接返回，并为结果指定了一个缺省的标识符（名字）来存放

②输入：3 + 2.0;

输出：stdIn:2.1-2.8 Error: operator and operand don't agree [literal]

operator domain: int \* int

operand: int \* real

in expression:

3 + 2.0

常量3默认为int型，常量2.0默认为real型，而+运算符要求两个操作数的类型一致，因此返回报错

③输入：it + 6;

输出：val it = 13 : int

在①中缺省的标识符it存放的值被拿来参与运算，运算的结果同样存放在缺省的标识符it中，从而重新声明了变量it，它和变量的更新有本质区别

④输入：val it = “hello”;

输出：val it = "hello" : string

字符串常量“hello”默认为string类型，而在这里通过val声明了一个名为it的变量，用来为常量命名，因而返回了声明成功的提示信息

⑤输入：it + “ world”;

输出：stdIn:4.4 Error: overloaded variable not defined at type

symbol: +

type: string

因为+运算符没有定义int与string类型作为操作数的行为，因此产生报错

⑥输入：it + 5;

输出：stdIn:1.2-2.2 Error: operator and operand don't agree [literal]

operator domain: string \* string

operand: string \* int

in expression:

it + 5

stdIn:1.5 Error: overloaded variable not defined at type

symbol: +

type: string

中缀运算符+根据他的前面的操作数，直接推测另一个操作数应该是string类型，但实际的常量5默认为int类型，类型不匹配，因此产生了第一段报错

同时，由于string类型不能作为+的操作数，因此发生了与⑤一样的第二段报错

⑦输入：val a = 5;

输出：val a = 5 : int

和④类似，定义了变量a，作为常量5的名字

⑧输入：a = 6;

输出：val it = false : bool

在sml中=号是相等判别运算符，而不是用来赋值，因此它判断⑦中生命的变量a是否与常量6的值相等，并由它们不相等得到结果false

⑨输入：a + 8;

输出：val it = 13 : int

和③类似，但这里重新声明的变量是it，不会影响a的值

⑩输入：val twice = (fn x => 2 \* x);

输出：val twice = fn : int -> int

这是定义函数的另一种方法，直接通过定义变量的val，再通过后面的fn语句说明的函数参数与函数体，来定义函数，并将函数名记为twice

XI.输入：twice a;

输出：val it = 10 : int

这是一个使用⑩中定义的函数的语句，函数根据传入的参数，运行函数体并得到返回值，返回值用来声明一个缺省的变量it

XII.输入：let x = 1 in x end;

输出：stdIn:9.1-9.8 Error: syntax error: deleting LET ID EQUALOP

stdIn:9.11 Error: syntax error found at IN

如果在x前加val来声明它，报错就会解除，说明这个语句主要错在使用let语句时需要声明使用的变量

XIII.输入：foo;

输出：stdIn:1.2-1.5 Error: unbound variable or constructor: foo

该变量之前从未声明过，因此调用该变量单独构成一条语句时会报错。

XIV.输入：[1, “foo”];

输出：stdIn:1.2-3.3 Error: operator and operand don't agree [literal]

operator domain: int \* int list

operand: int \* string list

in expression:

1 :: "foo" :: nil

根据表中第一个元素1缺省的int类型，编译器默认该表为int类型的表，但第二个元素”foo”默认类型为字符串常量，与编译器得到的该表类型矛盾，因此产生报错。

# 1.

1.下列模式能否与类型为int list的L匹配成功？如果匹配不成功，指出该模式的类型？（假设x为int类型）

①x::L

输入：a(x::L);

输出：val it = 1 : int

可以匹配成功，测试方法是：

定义了一个参数为int list类型的函数：

fun a (L:int list)=1;

然后将变量作为参数传递给该函数

②\_::\_

输入：a(\_::\_)

输出：stdIn:5.4 Error: syntax error found at WILD

不能匹配成功，报错

但是反之，将(\_::\_)用来定义函数，就可以匹配成功，原因是它包含了int list

③x::(y::L)

输入：a(x::(y::L))

输出：val it = 1 : int

可以匹配成功，只是延展了表的大小

④(x::y)::L

输入：a((x::y)::L)

输出：stdIn:8.5-8.9 Error: operator and operand don't agree [tycon mismatch]

operator domain: int \* int list

operand: int \* int

in expression:

x :: y

不能匹配成功，因为::中缀运算符需要的操作数类型是int与一个int类型的表，从而组合成一个新的表，而：

如果事先声明了int型x和y，则x::y不满足操作数的要求，因此直接报错

如果把这个式子放在模式的位置，则会编译为一个int list类型的list

⑤[x, y]

输入：a ([x,y]);

输出：val it = 1 : int

可以匹配成功，因为[x,y]可以将x和y组合成一个list

⑥非空List

输入：a c;

其中c是一个非空list

输出：val it = 1 : int

可以匹配成功

⑦空List

输入：a d;

其中d是一个空List

输出：val it = 1 : int

可以匹配成功

# 2.

试写出与下列表述相对应的模式。如果没有模式与其对应，试说明原因。

①list of length 3

对应模式：[x,y,z]

严格匹配长度为3的list，但各种类型的list都可以

②lists of length 2 or 3

函数承载的模式：

fun a ([x,y,z])=1

|a [x,y]=2;

匹配了长度为2或3的list，各种类型的list都可以

但是考虑到模式匹配只能用一个式子表达，这样的模式应该是不存在的

③Non-empty lists of pairs

对应模式：((x , y)::L)

中缀运算符会根据前面(x,y)代表的任意类型pair，将后面的L理解为任意类型pair的表

而(x,y)保证了这个表非空

④Pairs with both components being non-empty lists

对应模式：(a::L,b::M)

同样使用了中缀运算符和任意符号，来保证pair中的参数为非空表

# 3.

分析下述程序段（左边括号内为标注的行号）：

(1) val x : int = 3

(2) val temp : int = x + 1

(3) fun assemble (x : int, y : real) : int =

(4) let val g : real = let val x : int = 2

(5) val m : real = 6.2 \* (real x)

(6) val x : int = 9001

(7) val y : real = m \* y

(8) in y – m

(9) end

(10) in

(11) x + (trunc g)

(12) end

(13)  
(14) val z = assemble (x, 3.0)

试问：第4行中的x、第5行中的m和第6行中的x的声明绑定的类型和值分别为什么？第14行表达式assemble(x, 3.0)计算的结果是什么？

第4行中的x绑定int类型，值为2

第5行中的m绑定real类型，值为12.4，因为使用了real强制类型转换后的局部变量x

第6行中的x绑定int类型，值为9001

第14行表达式结果为27

# 4.

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。

实现的函数如下：

fun zip ((s:string)::[],(x:int)::[])=(s,x)::[]

|zip ((s:string)::L,(x:int)::K)=(s,x)::zip(L,K)

|zip ((s:string)::L,[])=[]

|zip ([],(x:int)::K)=[]

|zip ([],[])=[];

（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。

实现的函数如下：

fun unzip ((s:string,x:int)::[])=(s::[],x::[])

|unzip ((s:string,x:int)::L)=

    let

      val (L1,L2)=unzip(L);

    in

      (s::L1,x::L2)

    end;

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。

不成立，如果L1、L2不等长就不成立

# 5.

指出下列代码的错误：

①(\* pi: real \*)

val pi : real = 3.14159;

没有错

②(\* fact: int -> int \*)

fun fact (0 : int) : int = 1

| fact n = n \* (fact (n - 1));

没有错

③(\* f : int -> int \*)

fun f (3 : int) : int = 9

f \_ = 4;

错误：使用通配符充当了函数的参数

④(\* circ : real -> real \*)

fun circ (r : real) : real = 2 \* pi \* r

错误：函数中2没有用real强制转换成real类型，\*运算符会根据2认为是int\*int运算导致错误

⑤(\* semicirc : real -> real \*)

fun semicirc : real = pie \* r

错误：没有参数，并且pie没有定义

⑥(\* area : real -> real \*)

fun area (r : int) : real = pi \* r \* r

错误：r本该定义为real类型

# 上机实验内容1

和课堂练习1重复

# 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 \*)

填写后的函数为：

fun mult [ ] =  1   (\* FILL IN \*)

| mult (x ::L) = x\*mult(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 \*)

填写后的函数为：

fun Mult [ ] =  1 (\* FILL IN \*)

    | Mult ((r:int list) :: R) = mult(r)\*Mult(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 \*)

填写后的函数如下：

(\* mult’ : int list \* int -> int            \*)

(\* REQUIRES: true               \*)

(\* ENSURES: mult’(L, a) … (\* FILL IN \*) \*)

(\*该函数用于将int与int list中每个元素相乘得到结果\*)

fun mult' ([ ], a) = a

    | mult' (x :: L, a) = mult' (L, x \* a);

fun Mult' ( [ ], a) = a (\* FILL IN \*)

    | Mult' ((r:int list)::R, a) = mult'(r,a)\* Mult'(R,1);  (\* FILL IN \*)

# 5.

编写递归函数square实现整数平方的计算，即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)

编写出的函数如下：

(\* 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)

fun square (n:int)=

    let

      val times=n div 2;

      val moding=n mod 2;

      fun sum 0:int=0

        |sum m:int=double(n)+sum(m-1);

    in

      if moding=1

        then n+sum(times)

      else

        sum(times)

    end;

# 6.

定义函数divisibleByThree: int -> bool，以使当n为3的倍数时，divisibleByThree n为true，否则为false。注意：程序中不能使用取余函数’mod’。

(\* divisibleByThree : int -> bool \*)

(\* REQUIRES: true \*)

(\* ENSURES: divisibleByThree n evaluates to true if n is a multiple of 3 and to false otherwise \*)

定义好的函数如下：

fun divisibleByThree 0=true

    |divisibleByThree 1=false

    |divisibleByThree 2=false

    |divisibleByThree (n:int)=divisibleByThree(n-3);

# 7.

函数evenP为偶数判断函数，即当且仅当该数为偶数时返回true。

其代码描述如下：

(\* evenP : int -> bool \*)

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

(\* ENSURES: evenP n evaluates to true iff n is even. \*)

fun evenP (0 : int) : bool = true

| evenP 1 = false

| evenP n = evenP (n - 2)

试编写奇数判断函数oddP: int -> bool，当且仅当该数为奇数时返回true。注意：代码不要调用函数evenP或mod。

编写好的函数如下：

fun oddP (0 : int) : bool = false

    | oddP 1 = true

    | oddP n = oddP (n - 2);