EMACS 常用操作：
1）Control-X,Control-F:建立一个文件路径，可用于创建文件。
2）Control-X,Control-S:用于储存文件
3）Control-C,Control-S:进入 sml 主界面，use “filename”运行文件

Syntax:语法
Semantics:语义
Idioms:习语
Libraries:库
Tools:工具

SML(Standard ML)

Functional programming instead of procedural programming

Variables:

1)Syntax:

sequence of letters, digits, _, not starting with digit

2)Type-checking:

Look up type in current static environment

– If not there, fail

3)Evaluation:

Look up value in current dynamic environ

Syntax:

1)~5（~一元减号，只能用于~数字的形式），不能直接用-5，可以用 0-5
2）x div y 表示 x 除以 y
3）只有值默认变量名的 it
4）函数应用比中缀优先级更高，area a+b 表示 area(a)+b
5）#s s 为只有一个字符的字符串常量，则其为 char 类型。
^用于连接字符以及字符串。
6）逻辑或 orelse，逻辑与 andalso，逻辑非 not
等于=　　不等于<>　　andalso 只有前面为真时才检验后面
7）声明一个向量类型：type vec=real*real，实数序偶
8）域　{域标签=域值，…}，记录模式与域选择模式使用#域标签，类型约束时时不需要
Eg：type king={ name : string , born : int , died : int };
fun lifetime1 (k : king)=#died k-#born k;
fun lifetime2 ({ born , died,… })=died-born;
9）中缀操作符：
自定义中声明中缀操作符：infix 数字 操作符名字
数字代表优先级（优先级越高越先结合），然后函数具体实现
op 操作符名()取消了中缀写法，eg: op^(“abc” , ”def”)

nonfix 取消中缀写法 eg: nonfix * ;　　* (3 , 2) ;

10）迭代过程是尾递归的：每次迭代先展开后从最后一次迭代开始求值

函数调用时先算出其参数值，因此迭代比递归更好

Eg;阶乘函数：fact(4)=4*fact(3)=…=4*(3*(2*(1*1)))=24

迭代可以改进:通过条件表达式的特殊作用：

Eg：函数 powoftwo 测试一个数是否为 2 的幂次

```
fun even n = (n mod 2 = 0) ;
fun powoftwo n = (n=1) orelse (
even(n) andalso powoftwo (n div 2) );
```

11）严格求值：传名调用

惰性求值：传需调用：图归约：参数出现的地方用指针连接到表达式上，做到了最少的求值，但维护开销大。

12）通过辅助函数可以避免重复计算。

Let D1; D2;D3;…;Dn in E end 在函数声明表达式里求值，与辅助函数类似

Eg: fun fraction (n, d) = let val com =gcd (n, d)　in　(n div com, d div com) end;

13）嵌套函数：let…in…end 声明局部变量，可以嵌套函数，变量表达式

let b1 b2 b3 … bn in e end

14）隐藏声明：local

Local D1 in D2 end 声明 D1，D2

D1 只在 D2 中可见，在外面不可见

15）联立声明，强调彼此独立性

Val Id1=E1 and…and Idn=En　　顺序无关紧要，通常彼此独立。

16）复数的实现：二元组或结构

```
structure Complex=
    struct
    type t =real*real;
    val zero = (0.0, 0.0);
    fun sum =((x,y),(x',y'))=(x+x',y+y'):t;
    fun diff =((x,y),(x',y'))=(x-x',y-y'):t;
    fun prod=((x,y),(x',y'))=(x*x'-y*y',x*y'+x'*y):t;
    fun recip=((x,y),(x',y'))=
                let val t =x*x'+y*y';
                in(x/t,~y/t) end
    fun quo (z,z')=prod(z,recip z');
end;
```

17)Options:通过 SOME e 将 E 再转化为 option,在通过 valof 提取出来