プログラミング言語 Standard ML 入門 (問題の解答例)

大堀 淳

2021 年 11 月 5 日

本 WEB ページは、教科書

プログラミング言語 Standard ML 入門 (改訂版) 大堀　淳著 共立出版株式会社

の中の問題の解答例です。
利用にあたっての参考情報：

- 章と節、さらに、問題番号は、教科書のものと同一です。
- 利用者の便宜のため、解答例だけではなく、問題も提示しています。
- 練習問題は，初版 17.2 節，17.3 節および第 18 章を除き，初版と同一です．初版の第 17、18 章の内容はシステム開発環境です。これら本格的なシステムプログラミングは，本書の姉妹書 SML#で始める実践 ML プログラミング」（大堀・上野，共立出版 2021）を参考にしてください．

謝辞: 本解答例を注意深く読んでくださり、種々の誤りをご指摘頂いている 花田 覚氏 (GitHub の UserProfile) に深謝いたします。

第I部

Standard ML言語

第1章　MLプログラミングの基本

1.2　式の入力と評価

問 1.1 第 18 章付録の情報などを参考に SML#コンパイラをインストールし，対話型システムを起動し，以上のような対話が可能であることを確かめよ．

解答例　SML#を，そのホームページ https://smlsharp.github.io/ja/ の情報を参考にインストールしてみよう。例えば，お使いのシステムが VittualBox や WSL2 などにセットアップされた Linux ディストリビューションであれば，ダウンロードページの情報から簡単にインストールできる．インストールが成功すれば，以下のように起動できるはずである．

```
$ smlsharp
SML# 4.0.0 (2021-04-06 05:12:50 JST) for x86_64-pc-linux-gnu with LLVM 10.0.0
# fun f x = x;
val f = fn : ['a. 'a -> 'a]
```

問 1.2 以下の各行をこの順に入力した場合の各行の結果を予測せよ．

```
1. 44;
2. it mod 3;
3. 44 - it;
4. (it mod 3) = 0;
5. if it then "Boring" else "Strange";
```

解答例　以下の通りである。

```
# 44;
val it = 44 : int
# it mod 3;
val it = 2 : int
# 44 - it;
val it = 42 : int
# (it mod 3) = 0;
val it = true : bool
# if it then "Boring" else "Strange";
val it = "Boring" : string
```

問 1.3 英大文字の ASCII コードは A から Z の順に並んでおり，英小文字の ASCII コードは a から z の順に並んでいる．この事実のみを仮定して，文字定数 `#"S"` を小文字に直すプログラムを書け.

解答例

```
# chr (ord \#"a" + (ord \#"S" - ord \#"A"));
val it = #"s" : char
```

問 1.4 英字の大文字と小文字の ASCII コードは連続していない．これらの間にある文字の数を求める式を書き評価せよ．さらにこの結果を利用して，英字の大文字と小文字の間にあるすべての文字を小さい順に並べた文字列を求めるプログラムを書け.

解答例 英字の大文字と小文字の間にある文字の数を求める式とその評価結果：

```
# if #"A" > #"a" then ord #"A" - (ord #"z" + 1) else ord #"a" - (ord #"Z" + 1);
val it = 6 : int
```

英字の大文字と小文字の間にあるすべての文字を小さい順に並べた文字列を求める式とその評価結果

```
# if (ord #"a") > (ord #"A")
  then
       str (chr ((ord #"Z") + 1))
     ^ str (chr ((ord #"Z") + 2))
     ^ str (chr ((ord #"Z") + 3))
     ^ str (chr ((ord #"Z") + 4))
     ^ str (chr ((ord #"Z") + 5))
     ^ str (chr ((ord #"Z") + 6))
     ^ str (chr ((ord #"Z") + 7))
  else
       str (chr ((ord #"z") + 1))
     ^ str (chr ((ord #"z") + 2))
     ^ str (chr ((ord #"z") + 3))
     ^ str (chr ((ord #"z") + 4))
     ^ str (chr ((ord #"z") + 5))
     ^ str (chr ((ord #"z") + 6))
     ^ str (chr ((ord #"z") + 7))
   ;

val it = "[\\]^_`" : string
```

1.4　変数の束縛と識別子

問 1.5 以下の各々の文字列について，それが値を表す変数として使用できるか否かを判定せよ.

```
1stLady name1 __ 'a _abc %1 123 ten% ##' () and + ++
```

さらに，実際に値の束縛を行い，判定が正しいことを確かめよ．

解答例 以下，変数としての使用が可の場合はSML#による束縛の例を，不可の場合はその簡単な説明を付す。

- `1stLady`

  不可。"1"と"stLady2"の２語である。変数束縛の位置に現れると，"1"が"stlady"に適応されたパターン式と見なされエラーとなる。

- `name1`

```
# val name1 = "name1";
val name1 = "name1" : string
```

- `__`

  不可。"_"は匿名パターンを表す予約語。"_"が"_"に適用されたパターン式と解釈され，"_"がコンストラクターではないのでエラーとなる。

- `'a`

  不可。""で始まる名前は型変数にのみ使用可能。

- `_abc`

  不可。"_"が"abc"に適用されたパターン式と解釈されエラーとなる。

- `%1`

  不可。`%`と `1` の２語と解釈される。

  ちなみに、`%`が 1 の型 (int、intInfo 等) を引数としてとるコンストラクタと宣言されていれば、式またはパターンとして受理される。

```
# datatpe foo = % of int
datatype foo = % of int
# %1;
val it = % 1 : foo
```

  そのような宣言がなければ、型エラーとなる。

- `123`

  不可。定数は変数として使用できない。ただし `val 123 = 123` は成功する。

- `ten%`

  不可。`ten` と`%`の 2 語と解釈される。

  ちなみに、`%`が型 τ の変数と宣言され、`ten` が τ 型を引数とするコンストラクタと宣言されていれば、受理される。

  ```
  # datatype foo = ten of string;
  datatype foo = ten of string
  # val % = "%";
  val % = "%" : string
  # ten%;
  val it = ten "%" : foo
  ```

  それ意外の場合、型エラーとなる。

- `##'`

  不可。”##”と””の 2 語と見なされ，””が型の名前に使用されていないので構文エラーとなる。

- `()`

  不可。”()”は空の組を表す式。

- `and`

  不可。”and”は予約語である。

- `+`

  可能であるが，トップレベルの環境では”+”に対して `infix` 宣言が有効となっているので，この宣言を取り消さないと通常の変数としては使用できない。

- `++`

  ```
  # val ++ = 1;
  val ++ = 1 : int
  ```

1.5　ファイルからのプログラムの入力

問 1.6 第 1.2 節で作成した 3 文字を比較するプログラムをファイル try.sml に作成し，実行せよ.

解答例

ファイル `try.sml`：

```
if #"?" < #"$" then
   if #"$" < #"*" then
     (str #"?") ^ (str #"$") ^ (str #"*")
   else if #"?" < #"*" then
     (str #"?") ^ (str #"*") ^ (str #"$")
   else (str #"*") ^ (str #"?") ^ (str #"$")
else if #"?" < #"*" then
  (str #"$") ^ (str #"?") ^ (str #"*")
else if #"$" < #"*" then
  (str #"$") ^ (str #"*") ^ (str #"?")
else (str #"*") ^ (str #"$") ^ (str #"?") ;
```

実行結果

```
$ smlsharp
# use "try.sml";
val it = "$*?" : string
```

問 1.7 今後 ML プログラムをファイルに作成するための雛形として，以下のヘッダを持つファイルを作成せよ．

```
(* SML source file.  Copyright (c) by yourName thisYear.
 *
 *)
```

このファイルを，問 1.6 で作成した `try.sml` ファイルの冒頭に挿入し，実行せよ．

解答例 省略。

第2章　関数を用いたプログラミング

2.1　関数の定義

問 2.1 関数 f と g を以下のように定義する.

```
fun f x = (x,2) ;
fun g (x,y) = x + y;
```

以下の各式について，式に誤りがなく結果が出るものはその結果を予想し，エラーとなるものはそのエラーの原因を指摘せよ.

```
f f(1);        f (f 1);        f (f(1));
f (1,2);       (f 1,f 2);      g(1,2);
g f(1);        g (f 1);        f g(1,2);
f (g(1,2));    g (f 1,f 2);    f (g 1,g 2);
```

さらに，ML で実際に評価し，結果をチェックせよ.

解答例

- f f(1); 型エラー。関数適用は左結合であるから f f(1) は ((f f)(1)) と解釈される。この場合 f f の部分が型付け不可能。
- f (f 1):

```
# f (f 1);
val it = ( ( 1, 2 ), 2 ) : (int * int) * int
```

- f (f(1)); 上記と同一の式である。
- f (1,2);

```
# f (1,2);
val it = ( ( 1, 2 ), 2 ) : (int * int) * int
```

- (f 1,f 2);

```
# (f 1,f 2);
val it = ( ( 1, 2 ), ( 2, 2 ) ) : (int * int) * (int * int)
```

- g(1,2);

```
# g(1,2);
val it = 3 : int
```

- `g f(1);` 型エラー。`g f` の部分が型付け不可能。
- `g (f 1);`

```
# g (f 1);
val it = 3 : int
```

- `f g(1,2);` 型エラー。`g f` の部分が型付け不可能。
- `f (g(1,2));`

```
# f (g(1,2));
val it = ( 3, 2 ) : int * int
```

- `g (f 1,f 2);` 型エラー。`g` の引数は `int` 型の組である必要がある。
- `f (g 1,g 2);` 型エラー。`g` の引数は `int` 型の組である必要がある。

2.3　再帰的関数

問 2.2 `factorial` 関数が正しく動作することを，`factorial 0`，`factorial 2`，`factorial 3` の各場合の動作をトレースすることによって確かめよ．

解答例 `factorial 3` の呼び出しは以下のようなトレースを生成する．

```
factorial 3
 > 3 * factorial 2
 >      > 2 * factorial 1
 >      >      > 1 * factorial 0
 >      >      >      > 1
 >      >      > 1 * 1
 >      >      > 1
 >      > 2 * 1
 >      > 2
 > 3 * 2
 > 6
```

問 2.3 以下の各数列 S_n について，S_n が満たすべき性質を n に関する漸化式として記述し，それに対応する再帰的関数を記述することによって，S_n を求めるプログラムを書け．

1. $S_n = 1 + 2 + \cdots + n$

2. $S_n = 1 + (1 + 2) + (1 + 2 + 3) + \cdots + (1 + 2 + \cdots + n)$

解答例

1. 漸化式は

$$\begin{aligned} S_0 &= 0 \\ S_n &= n + S_{n-1} \end{aligned}$$

と与えられる．よって，この関数を f とすると，以下のコードで実現できる．

```
fun f 0 = 0
  | f n = n + f (n - 1);
```

2. 漸化式は、上記の漸化式の解を S_n として、

$$\begin{aligned} T_0 &= 0 \\ T_n &= T_{n-1} + S_n \end{aligned}$$

と与えられる．よって，この関数を g とすると，以下のコードで実現できる．

```
fun g 0 = 0
  | g n = g (n - 1) + f n;
```

問 2.4 問 2.3 で定義した 2 つの関数が，それぞれ正しく数列の和を計算することを，入力 n に関する帰納法で証明せよ．

解答例

1. $S_n = 1 + 2 + ... + n$

   (a) （基底）n = 0 のとき．$S_0 = 0$, `f 0` = `0` で正しい．

   (b) （帰納段階）n = k のとき正しいと仮定する．

$$S_{k+1} = 1 + 2 + ... + k + (k + 1) = S_k + (k + 1)$$

   である．一方`f (k + 1) = (k + 1) + f k`であり，帰納法の仮定より`f k` = S_k であるから，`f (k + 1)` = S_{k+1} である．

2. $S_n = 1 + (1 + 2) + (1 + 2 + 3) + ... + (1 + 2 + ... + n)$

   (a) （基底）$n = 0$ のとき，$S_0 = 0$, `f 0` = `0` で正しい．

(b)（帰納段階）$n = k$ のとき正しいと仮定する．

$$\begin{aligned} S_{k+1} &= 1 + (1+2) + (1+2+3) + ... + (1+2+...+k) + (1+2+...+k+(k+1)) \\ &= S_k + (1+2+...+k+(k+1)) \end{aligned}$$

である．一方 `f (k + 1)` $=$ `f k + g (k + 1)` であり，帰納法の仮定と前項の結果より `f k` $= S_k$ かつ `g (k+1)` $= (1+2+...+k+(k+1))$ であるから，`f (k + 1)` $= S_{k+1}$ である．

問 2.5 以下のように再帰的に定義される数列をフィボナッチ数列と呼ぶ．

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-2} + F_{n-1} \qquad (n \geq 2) \end{aligned}$$

負でない整数 n を受け取り F_n を計算する関数 `fib` を定義せよ．

解答例

```
fun fib n = if n = 0 then 0
            else if n = 1 then 1
            else fib (n - 1) + fib (n - 2)
```

問 2.6 k を与えられた負でない整数とし，任意の $0 \leq n \leq k$ について `fib n` が F_n を計算することを，k に関する数学的帰納法で示せ．これを用いて，`fib` が，0 以上の任意の自然数に対して正しく F_n の値を計算することを確認せよ．

解答例 任意の $0 \leq n \leq k$ について `fib n` が F_n を計算することの k に関する帰納法による証明

- $k = 0, 1$ の時．`fib 0` $= 0 = F_0$ かつ `fib 1` $= 1 = F_1$ であり成立．
- $k = i (i \geq 1)$ の時成立すると仮定する．`fib (i + 1)` $=$ `fib` $i +$ `fib (i - 1)` である．帰納法の仮定より，`fib` $i = F_i$ かつ `fib (i - 1)` $= F_{i-1}$ である．また定義より，$F_i = F_{i-1} + F_{i-2}$ である．よって，`fib` $i = F_i$ となる．従って，任意の $0 \leq n \leq i+1$ について `fib n` $= F_n$ であり，成立する．

以上より，0 以上の任意の k に対して，任意の $0 \leq n \leq k$ について `fib n` $= F_n$ であるから，0 以上の任意の自然数に対して正しく F_n の値を計算する．

問 2.7 フィボナッチ数列 F_n の再帰的定義をそのまま再帰的関数として実現すると，F_n の計算に掛かる時間は n が大きくなるに従って指数関数的に増大する．F_n を n に比例する時間で計算する関数 `fastFib` を以下の洞察を参考に定義せよ．

1. フィボナッチ数列の定義は，連続する 2 つの数から次の数を決めるような構成になっている．したがって，F_k と F_{k+1} が与えられれば F_{k+2} が即座に求められる．この作業を $n-1$ 回繰り返せば，F_k と F_{k+1} から F_{k+n} が求められるはずである．

2. 前項の考え方を参考に，与えられた連続する 2 つのフィボナッチ数 F_k と F_{k+1} から，F_k の n 個先のフィボナッチ数を求める関数 G(n,F_k,F_{k+1}) を定義する．n が 0 の場合は F_k を返せばよい．n が 1 の場合は F_{k+1} を返せばよい．n が 2 以上の場合は，F_k と F_{k+1} から F_{k+2} を求め，F_{k+1} と F_{k+2} から，F_{k+1} の $n-1$ 個先のフィボナッチ数を求めればよいから，G($n-1$,F_{k+1},F_k+F_{k+1}) を呼び出せばよい．

3. F_n は F_0 から n 個先のフィボナッチ数であるから，G(n,F_0,F_1) である．したがって fastFib は，上記の動作をする補助関数 G を呼び出すことによって実現できる．

解答例

```
fun G(n,fk,fk1) = if n = 0 then fk
                  else if n = 1 then fk1
                  else G(n-1,fk1,fk+fk1);
fun fastFib n = G(n,0,1);
```

問 2.8 負でない任意の整数 n に対して G(n,F_k,F_{k+1}) は F_{k+n} を計算することを，n に関する数学的帰納法で証明せよ．この性質を使って，fastFib n が正しくフィボナッチ数 F_n を計算することを示せ．

解答例

- $n = 0, 1$ の時。

$$\begin{aligned} \texttt{G(0,}F_k\texttt{,}F_{k+1}\texttt{)} &= F_k \\ \texttt{G(1,}F_k\texttt{,}F_{k+1}\texttt{)} &= F_{k+1} \end{aligned}$$

  であり成立する．

- $n = i + 1 (i \geq 1)$ の時。G の定義より

$$\texttt{G(}i+1\texttt{,}F_k\texttt{,}\ F_{k+1}\texttt{)} = \texttt{G(}i\texttt{,}\ F_{k+1}\texttt{,}\ F_k + F_{k+1}\texttt{)}$$

  である．F_n の定義より，$F_k + F_{k+1} = F_{k+2}$ である．よって，G($i+1$，F_k，F_{k+1}) = G(i,F_{k+1},F_{k+2}) である．帰納法の仮定より，G($i+1$,F_k,F_{k+1}) = F_{k+i+1} となり成立する．

以上の特殊な場合として，G(n,F_0,F_1) = F_n であり，したがって，fastFib $n = F_n$ である．

2.4 局所変数の使用

問 2.9 問 2.7 で作成した fastFib の定義を，G が fastFib の局所関数となるように書き直せ．

解答例

```
fun fastFib n =
  let
    fun G(n,fk,fk1) = if n = 0 then fk
```

```
                        else if n = 1 then fk1
                        else G(n-1,fk1,fk+fk1);
    in
       G(n,0,1)
     end
```

2.5　相互再帰的関数

問 2.10 最低利率が 1%，残高 1000 万円以上の場合の利率が 2%，残高が 1000 万円以下の場合の利率は，残高に比例して 1%から 2%まで連続して増えるものとする．この利率を表す関数 F を定義し，F を使って I と A を定義せよ．さらに，A(100.0,10)，A(100.0,20)，A(100.0,25)，A(100.0,30) の値をそれぞれ計算せよ．

解答例 例えば以下のようにコード化できる．

```
fun F x = if x > 2000.0 then 0.02
          else 0.01 + 0.01 * (x  - 1000.0) / 1000.0;
fun I(x,n) = F(A(x,n - 1))
and A(x,n) = if n = 0 then x
             else A(x,n-1)*(1.0+I(x,n));
```

これを SML#で実行すると，以下のような結果となる．

```
# A(900.0, 10);
val it = 988.102574973 : real
# A(900.0, 20);
val it = 1095.21279511 : real
# A(900.0, 25);
val it = 1157.91903751 : real
# A(900.0, 30);
val it = 1228.19345742 : real
```

(注１）教科書では A(100.0,10)，A(100.0,20)，A(100.0,25)，A(100.0,30) をそれぞれ計算せよ，との問題であるが，この範囲では F の変化が結果に影響しないので，値を変えてある．
(注２）A(900.0，30) の計算は、筆者の現在（Mon Jul 13 05:52:44 JST 2020）の環境では 21 秒ほど掛かる。

問 2.11 問 2.10 を実際試してみるとわかる通り，関数 A は n が大きくなると処理時間が膨大になり，実用に適さない．問 2.7 で定義した fastFib の考え方に習い，A_x^n と I_x^n の組を，以下の方針で同時に高速に計算する補助関数 auxAI を考えることができる．

1. auxAI の引数は，目標とする年までの残りの年数 k，そのときの残高 x，および利率 i の値の組とする．
2. 残りの年数が 0 の場合は，引数をそのまま返せばよい．したがって auxAI(0,x,i) は (x,i) である．

3. k が 0 でない場合，与えられた x と i から 1 年先の x' と i' を求め，それらから $k-1$ 年先の値を求めればよいから，`auxAI(`k`,`x`,`i`)` は，`auxAI(k-1,`x'`,`i'`)` を呼び出すことによって計算できる．i' は F を前年の残高に適用して得られる値であり，x' は x に $1+i'$ を乗じたものである．

`auxAI` を補助関数として用い，残高 x と年数 n から A^n_x と I^n_x の組を計算する関数 `fastAI` を定義せよ．さらに，`fastAI(100.0,100)` と `fastAI(100.0,1000)` の値をそれぞれ計算せよ．

解答例 `fastAI` は以下のようにコード化できる．但し，`F` は定義済みと仮定している．

```
fun auxAI(n,x,i) = if n = 0 then (x,i)
                   else auxAI(n-1,x*(1.0 + F(x)),1.0 + F(x))
fun fastAI(x,n) = auxAI(n,x,F(x));
```

SML#での実行結果は以下の通り．

```
# fastAI(900.0,100);
val it = (4305.75128361, 1.02) : real * real
# fastAI(900.0,1000);
val it = (236702870934.0, 1.02) : real * real
```

これらは一瞬で終了する．

問 2.12 上記の 2 つのプログラムで，$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ の 20 乗と 30 乗を実際に計算し，かかった時間を比較せよ．

解答例 筆者の現在の環境で time コマンドで計測した実行時間は以下の通り。

- プログラム:`(x (20, 1, 0, 0, 1), y (20, 1, 0, 0, 1), z (20, 1, 0, 0, 1), w (20, 1, 0, 0, 1))`

```
real    0m0.099s
user    0m0.072s
sys     0m0.024s
```

- プログラム:`(x (30, 1, 0, 0, 1), y (30, 1, 0, 0, 1), z (30, 1, 0, 0, 1), w (30, 1, 0, 0, 1))`

```
real    1m14.405s
user    1m13.904s
sys     0m0.512s
```

- プログラム：`matrixPower (20, 1, 0, 0, 1)`

```
real    0m0.003s
user    0m0.000s
sys     0m0.000s
```

- プログラム：`matrixPower (30, 1, 0, 0, 1)`

```
real    0m0.003s
user    0m0.000s
sys     0m0.004s
```

問 2.13 フィボナッチ数の定義から，当然，以下の等式が成立する．

$$\begin{aligned} F_n &= F_n \\ F_{n+1} &= F_{n-1} + F_n \end{aligned}$$

今，$G_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$ とおき，G_n と G_{n-1} との関係を考えると，上記の等式は，以下のような行列を用いた式で表現できる．

$$G_n = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} G_{n-1}$$

$G_0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ であるから，定義に従い展開すると，以下の式が得られる．

$$G_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}}_{n \text{ 回の繰り返し}} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ とおくと，行列の積の結合性より，以下の結果を得る．

$$G_n = A^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

したがって，F_n は行列 A^n の第一行第二列目の成分に一致する．この性質を使い，フィボナッチ数を線形時間で求める関数 `fastFib'` を `matrixPower` を使って定義せよ．

解答例 ベクトルとの積を考えると A_n の第 1 行の和が F_n であるから以下のようにコード化できる．

```
fun fastFib' n = let
                   val (a,b,c,d) = matrixPower(n,0,1,1,1)
                 in
                   a + b
                 end;
```

問 2.14 `matrixPower` は，n に関して線形時間を必要とするが，結合律が成り立つ積に関する巾乗の計算の場合は，それをさらに対数時間に高速化可能である．以下の処理を行う `fastMatrixPower` を定義せよ．

1. $n = 0$ なら単位行列 (1,0,0,1) を返す．
2. $n > 0$ なら $m = \frac{n}{2}$ と $k = (n \ mod \ 2)$ を求め，以下の処理を行う．

(a) fastMatrixPower(m,a,b,c,d) を計算する.

(b) 上記の行列同士の積を計算する.

(c) もし $k = 1$ なら，さらに，上で求めた行列と引数で与えられた行列の積を求める.

n が 2^k の場合について，fastMatrixPower が行う行列の掛け算の回数を見積もれ.

matrixPower(1000000,1,0,0,1) と fastMatrixPower(1000000,1,0,0,1) とをそれぞれ計算し，実行速度を比較せよ.

fastMatrixPower を使って，fastFib' と同一の動作をする関数 veryFastFib を定義せよ.

解答例 fastMatrixPower は以下のようにコードできる.

```
fun matrixMult (a,b,c,d) (x,y,z,w) = (a*x+b*z,a*y+b*w,c*x+d*z,c*y+d*w)
fun fastMatrixPower (n,a,b,c,d) =
     if n = 0 then (1,0,0,1)
     else let val m = n div 2
              val k = n mod 2
          in if m = 0 then (a,b,c,d)
             else let val (x,y,z,w) = fastMatrixPower(m,a,b,c,d)
                      val (x,y,z,w) = matrixMult (x,y,z,w) (x,y,z,w)
                  in if k = 0 then (x,y,z,w)
                      else matrixMult (x,y,z,w) (a,b,c,d)
                  end
          end;
```

fastMatrixPower が引数 2^k に対して行う掛け算の回数を S_k と置くと

$$\begin{aligned} S_0 &= 0 \\ S_{k+1} &= S_k + 8 \end{aligned}$$

である．従って $8k$ 回である.

以下に、プログラムコードと現在の著者の環境で計測した実行時間を示す。

- プログラム：matrixPower (1000000,1, 0, 0, 1)

```
real    0m0.085s
user    0m0.052s
sys     0m0.032s
```

- プログラム：fastMatrixPower (1000000,1, 0, 0, 1)

```
real    0m0.002s
user    0m0.000s
sys     0m0.000s
```

veryFastFib は，fastMatrixPower を使い以下のように定義できる．

```
fun veryFastFib n =
    let
      val (a,b,c,d) = fastMatrixPower(n,1,1,1,0)
    in
      b
    end;
```

2.6　高階の関数

問 2.15　以下の関数を summation を使って定義せよ．

1. $f(x) = 1 + 2 + \cdots + n$
2. $f(n) = 1 + 2^2 + 3^2 + \cdots + n^2$
3. $f(n) = 1 + (1 + 2) + (1 + 2 + 3) + \cdots + (1 + 2 + \cdots + n)$

解答例

1. `fun f1 n = summation (fn x =>x) n;`
2. `fun f2 n = summation (power 2) n;`
3. `fun f3 n = summation f1 n;`

問 2.16　使用しているコンパイラに応じて，以下のいずれかの問題を解け．

- SML#コンパイラの場合：summation が整数から実数への関数 f に対して，$\sum_{k=1}^{n} f(k)$ を求める関数としても使用できることを確かめよ．
- オーバーロード多相性のサポートがないコンパイラの場合：整数から実数への関数 f に対して，$\sum_{k=1}^{n} f(k)$ を求める関数

  ```
  summation' : (int -> real) -> int -> real
  ```

  を定義せよ．

解答例

- ```
  $ smlsharp
  # fun summation f n = if n = 1 then f 1 else f n + summation f (n - 1);
  # val summation =
    fn : ['a::{int, word, int8, word8, ...}.  (int -> 'a) -> int -> 'a]
  # summation (fn x => real x + 1.1) 10;
  val it = 66.0 : real
  ```
  ```

- ```
  fun summation' f n =
      if n <= 0 then 0.0
      else f n + summation' f (n - 1);
  ```

問 2.17 $f(x)$ を実数から実数への関数とする．f の $[a,b]$ の区間の定積分の値 $\int_a^b f(x)dx$ は，n が十分に大きいとき，

$$\sum_{k=1}^{n}\left(f\left(a+\frac{k(b-a)}{n}\right)\times\frac{b-a}{n}\right)$$

で近似できる．f と n と a と b を受け取り，上記の値を計算する関数 `integral` を定義せよ．その際，n や k は `int` 型データであるから，$b-a$ などの `real` 型データとの演算では， 第 1.2 節で紹介した型変換関数 `real` を使用する必要がある点に注意せよ．

$\int_0^1 x^3dx$ の近似値を，$n=1000,10000,100000$ のそれぞれに対して計算せよ．

解答例 `integral` の定義例：

```
fun integral f n a b =
    let val unit = (b - a) / real n
    in summation' (fn x => f(a + real x * unit) * unit) n
    end;
```

$n=1000,10000,100000$ に対する $\int_0^1 x^3dx$ の近似値の計算例：

```
# fun cube x = x * x * x * 1.0;
val cube = fn : real -> real
val it = fn : real -> real -> real
# integral cube 1000 0.0 1.0;
val it = 0.25050025 : real
# integral cube 10000 0.0 1.0;
val it = 0.2500500025 : real
# integral cube 100000 0.0 1.0;
val it = 0.250005000025 : real
```

なお、解析的な計算結果は $1/41.0^4=0.25$ である。

問 2.18 `summation` は，以下のより一般的な計算スキーマの特殊な場合と考えることができる．

$$\Lambda_{k=1}^{n}(h,f,z)=h(f(n),\cdots,h(f(1),z)\cdots)$$

たとえば $\sum_{k=1}^{n}f(k)=\Lambda_{k=1}^{n}(+,f,0)$ と考えられる．

1. $\Lambda_{k=1}^{n}(h,z,f)$ を計算する高階関数

   ```
   accumulate h z f n
   ```

   を定義せよ．
   ```

2. summation を accumulate を使って定義せよ.

3. accumulate を使って以下の各計算をする関数を定義せよ.

   (a) $f_1(n) = 1 + 2 + \cdots + n$

   (b) $f_2(n) = 1 \times 2 \times \cdots \times n$

   (c) $f_3(n, x) = 1 \times x^1 + 2 \times x^2 + \cdots + n \times x^n$

解答例 accumulate の定義例

```
fun accumulate h z f n =
    if n = 0 then z
    else h(f n,accumulate h z f (n - 1));
```

accumulate を使った summation、f1、f2、f3 の定義例。

1. `val summation = accumulate (op +) 0;`

2. `fun f1 n = accumulate (fn (x,y) => x + y) 0 (fn x => x) n;`

3. `fun f2 n = accumulate (fn (x,y) => x * y) 1 (fn x => x) n;`

4. `fun f3 (n,x) = accumulate (fn (x,y) => x + y) 0 (fn n => n * power n x) n;`

問 2.19 以前定義した関数 power と Power の違いは引数の渡し方のみであるから，power と Power のような関数同士は相互に変換可能なはずである．このような引数の受け渡し方の変換に関して以下の設問に答えよ.

1. power を Power を使って定義し直せ．また逆に，Power を power を使って定義し直せ.

2. τ_1 * τ_2 -> τ_3 の型の関数を τ_1 -> τ_2 -> τ_3 の型の関数に変換する高階の関数 curry およびその逆変換関数 uncurry を定義せよ．ただし，任意の f, x, y について (もし型が正しければ) 常に

   curry f x y = f(x,y)
   uncurry f (x,y) = f x y

   が成り立つものとする.

3. curry と uncurry を使って Power と power の変換を行い，正しく動作することを確かめよ.

解答例

- Power と power がすでに定義されていると仮定する．この下で，以下のように定義できる.

```
val Power = fn (m,n) => power m n
and power = fn m => fn n => Power(m,n)
```

- uncurry と curry の定義例

```
fun uncurry f x y = f (x,y)
fun curry f (x,y) = f x y
```

- SML#での実行結果を以下に示す.

```
# fun Power(m,n) = if m = 0 then 1
>                  else n * Power(m - 1,n);
val Power = fn : int * int  -> int
# fun power m n = if m = 0 then 1
>                  else n * power (m - 1) n;
val power = fn : int  -> int  -> int
# fun curry f x y = f(x,y);
val curry = fn : ['a,'b,'c.('a * 'b  -> 'c)  -> 'a  -> 'b  -> 'c]
# fun uncurry f (x,y) = f x y;
val uncurry = fn : ['a,'b,'c.('a  -> 'b  -> 'c)  -> 'a * 'b  -> 'c]
# val Power = uncurry power
> and power = curry Power;
val Power = fn : int * int  -> int
val power = fn : int  -> int  -> int
# Power (2,3) ;
val it = 9 : int
# power 2 3;
val it = 9 : int
```

問 2.20 問 2.14 で定義した fastMatrixPower の構造を一般化し，0 以上の整数 n，2 項演算関数 f，f に関する単位元 e，および値 v を受け取り，f を乗算演算とする v の n 乗を高速に計算する高階の関数 fastPower を定義せよ．ただし，$v^0 = e$ とし，$n \neq 0$ の場合，以下のような演算を行うものとする．

$$\texttt{fastPower}(n, f, v, e) = \underbrace{f(v, f(v, \cdots, f}_{n \text{ 回の } f \text{ の適用}}(v, e)\cdots))$$

fastPower を用いて，fastMatrixPower を定義し直せ.

解答例

```
fun matrixMult (a,b,c,d) (x,y,z,w) = (a*x+b*z,a*y+b*w,c*x+d*z,c*y+d*w)
fun fastPower (n,f,v,e) =
     if n = 0 then e
     else let val m = n div 2
              val k = n mod 2
          in if m = 0 then v
             else let val v1 = fastPower(m,f,v,e)
```

```
                    val v2 = f (v1,v1)
                in if k = 0 then v2
                    else f(v,v2)
                end
        end;
fun fastMatrixPower(n,a,b,c,d) =
    fastPower(n, fn (x,y) => matrixMult x y, (a,b,c,d), (1,0,0,1));
```

2.7　関数式

問 2.21 以上の 2 つの f の定義のそれぞれについて，上記の g を含むプログラムの評価を行い，実行時間を比較せよ．

解答例　省略。

問 2.22 int -> bool 型の関数は，与えられた整数が条件を満たすか否かを判定する関数であるが，見方を変えれば，条件を満たす整数の集合の表現とみなすことができる．たとえば

```
fn x => x = 1 orelse x = 2 orelse x = 3
```

は集合 $\{1,2,3\}$ の表現と考えることができる．ここで exp_1 orelse exp_2 は，exp_1 と exp_2 の論理和（exp_1 または exp_2）を表す構文である．exp_1 と exp_2 の論理積（exp_1 かつ exp_2）を表す構文は exp_1 andalso exp_2 である．

この見方に従い，以下の定義を与えよ．

1. 空集合を表す関数 emptySet.
2. 整数 n を受け取り，単位集合 $\{n\}$ を生成する関数 singleton.
3. 集合 S に要素 n を加える関数 insert.
4. 要素 n が集合 S に含まれるか判定する関数 member.
5. 2 つの集合の和集合，積集合，差集合を計算する関数 union，intersection，difference.

解答例

```
val emptySet = fn x => false
fun singleton n = fn x => x = n
fun insert n S = fn x => x = n orelse S x
fun member n S = S n
fun union S1 S2 = fn x => S1 x orelse S2 x
fun intersection S1 S2 = fn x => S1 x andalso S2 x
fun difference S1 S2 = fn x => S1 x andalso not (S2 x)
```

2.8 変数のスコープ

問 2.23 以下の 2 つの宣言列を考える.

宣言列（1）

```
val x = 1
val y = 2
val x = x * 2 + y
val y = x + y * 2;
```

宣言列（2）

```
val x = 1
val y = 2
val x = x * 2 + y
and y = x + y * 2;
```

これら各宣言列の後の x と y の値は何か.

解答例 それぞれ、問題の趣旨は、実行して確認することではないが、SML#で実行すると、以下のようになる。

```
# val x = 1
> val y = 2
> val x = x * 2 + y
> val y = x + y * 2;
val x = 4 : int
val y = 8 : int

# val x = 1
> val y = 2
> val x = x * 2 + y
> and y = x + y * 2;
> ;
val x = 4 : int
val y = 5 : int
```

問 2.24 以下のプログラムで宣言される各変数のスコープを示せ.

```
fun f (x,y) =
    let fun f x = x + y;
    in x + f y
    end;
f (2,3);
```

このプログラムの評価結果は何か？

解答例 各変数のスコープは以下の通り。

- 外側の `f` : `f (2,3);`
- 外側の `x` : `in x + f y end`

- 外側の y : x + y; in x + f y end;
- 内側の f : x + y; in x + f y end;
- 内側の x : x + y

2 + (3 + 3) が計算され、SML#で実行すると、val it = 8 : int と表示されるはずである。

問 2.25　静的スコープ規則に対して，動的なスコープ規則に基づいた変数の評価も可能である．動的なスコープ規則では，変数の値は，それが実際に評価された時点での環境でその変数に束縛された値である．

入れ子になった関数定義を含む以下の式を考える．

```
let fun f x =
        let fun g y = x + y
            fun h x = g (x * 3)
        in h (x + 3)
        end
in f 10
end
```

この式を ML で評価した時と動的スコープ規則で評価したときの結果はそれぞれ何か？

解答例　静的スコープ規則を持つ ML の場合、関数 g のコード x+y の x は、f の引数と同じ 10 であり、結果は 49 となる。

一方、動的スコープ規則を持つ言語の評価は、関数 g のコード x+y の x は、呼ばれた時点での環境に記述された x の値であり、以下のように評価され、

環境	式
{}	f 10
{x:10}	h (x + 3)
{x:10}	h 13)
{x:13}	g (x * 3))
{x:13}	g 39
{x:13, y:39}	x + y
{x:13, y:39}	52

結果は 52 となる。

2.9　2 項演算子

問 2.26　16 を法とした加減乗除を行う以下の演算子を定義せよ．

演算子	型	結合性	意味
`&`	`int -> int`	`--`	16 で割った余りを求める
`&+`	`int * int -> int`	`infix 8`	16 を法とする加算
`&-`	`int * int -> int`	`infix 8`	16 を法とする減算
`&*`	`int * int -> int`	`infix 9`	16 を法とする乗算
`&=`	`int * int -> int`	`infix 2`	16 を法とする等値性テスト

たとえば，以下のように評価される．

```
- & 17 ;
val it = 1 int
- 4 &* 5 &+ 1;
val it = 5 int
```

解答例

```
infix 8 &+ &-;
infix 9 &*;
infix 2 &=;
fun & n = n mod 16
fun n &+ m = ((n mod 16) + (m mod 16)) mod 16
fun n &- m = ((n mod 16) - (m mod 16)) mod 16
fun n &* m = ((n mod 16) * (m mod 16)) mod 16
fun n &= m = ((n mod 16) = (m mod 16))
```

第3章　MLの型システム

3.1　型推論と静的型チェック

問 3.1 以下のそれぞれのプログラムはいずれも型エラーを含む．それぞれについて，型エラーの原因を指摘せよ．

- `3 * 3.14`
- `fun f1 x = (x 1,x true)`
- `fun f2 x y = if true then x y else y x`
- `fun f3 x = x x`

解答例

- `3 * 3.14`

  組み込み演算*の引数はすべて整数型かすべて実数型のどちらかでなければならず，整数型と実数型は混在できない．ML に自動的な型変換はない．

- `fun f1 x = (x 1,x true)`

  関数 x の引数の型が int 型と bool 型の 2 通りに使われている．ML では，多相型が与えられる名前は関数宣言および val 宣言されたもののみであり，パターンの中に現れる変数は，そのスコープでは単相型をもつ．

- `fun f2 x y = if true then x y else y x`

  `x y` は x の型が y の型を引数の型とする関数型であることを要求し，`y x` は y の型が x の型を引数の型とする関数型であることを要求するが，これら 2 つの条件は循環しており，これらを同時に満たす型は（有限の型の世界では) 存在しない．

- `fun f3 x = x x`

  `x x` は x の型が x の型を引数の型とする関数型であることを要求するが，この条件を満たす型は（有限の型の世界では) 存在しない．

（補足）型の意味を，型に属する値の集合と見なし，関数の意味を引数と結果の組の集合と考えると，`x x` は自分自身を部分として含む集合となり，矛盾する．さらに興味がある読者は,「ラッセルのパラドクス」に関するトピックを調べてみよ．

3.2　型の多相性と多相関数

問 3.2 `fn x => id id x` の型を以下の手順で求め，`'a -> 'a` となることを確認せよ．

1. `id` の型が `'a -> 'a` であることを用いて，`id id` の型を求めよ．
2. 一般に，関数 f が型 τ_1 `->` τ_2 を持てば，`fn x =>` f `x` も型 τ_1 `->` τ_2 を持つこと確かめよ．
3. 以上から，全体の型を求め，`'a -> 'a` となることを確認せよ．

解答例

1. `id` の型は`'a -> 'a` である．ここで，`'a` は任意の型を表す多相型変数である．そこで `id id` の 1 番目の `id` と 2 番目の `id` の型をそれぞれ `X -> X` および `Y -> Y` と置くことができる．前者が後者に適用されているので，`X = Y -> Y` であり，かつこの式全体の型は `X` である．よって `id id` の型は `Y -> Y` と書ける．
2. 関数 `f` が型 `A -> B` を持つとする．式 `f x` の型が正しいとすると `x` の型は `A` であり，かつ式 `f x` の型は `B` である．よって，式 `fn x => f x` の型は `A -> B` である．
3. 以上より，式 `fn x => id id x` の型は `id id x` の型と同じ型，すなわち `Y -> Y` である．さらに，`Y` は任意の型であるから，多相型変数を使うと，`'a -> 'a` と表される．

問 3.3 以下の各型を推定し，実際の型と比較せよ．

1. `twice cube`
2. `fn x => twice id x`
3. `fun thrice f x = f (f (f x))`

解答例　省略。

問 3.4　ML システムは，各引数の型に関するすべての条件を満たす最も一般的な解を計算することによって，関数の最も一般的な多相型を計算する．`twice` の最も一般的な多相型が `('a -> 'a) -> 'a -> 'a` であることを以下の手順で確かめよ．

1. `twice` の定義に現れる各式の型をそれぞれ以下のように仮定する．

$$\texttt{fun}\ \underline{\texttt{twice}}_{\tau_1}\ \underline{\texttt{f}}_{\tau_2}\ \underline{\texttt{x}}_{\tau_3} = \underline{\texttt{f (f x)}}_{\tau_4}{}_{\tau_5};$$

2. 関数適用 $\underline{\texttt{(f x)}}_{\tau_4}$ が型エラーを起こさないためには（1） `f` の型 τ_2 は α `->` β の形をした関数型であり，(2) 型 α は `x` の型 τ_3 と等しく，(3) 型 β は `(f x)` の型の結果の τ_4 と等しくなければならない．したがって，以下の等式が成立しなければならない．

$$\tau_2 = \tau_3\ \texttt{->}\ \tau_4$$

以上のような分析を他の要素についても行い，型の間に成り立つべき関係すべてを等式として書き出せ．

3. 2 で生成した型に関する等式の τ_1 に関する最も一般的な解を考え，それが twice の型 (’a -> ’a) -> ’a -> ’a であることを確かめよ.

解答例 twice の各変数に対して仮定された型 $\tau_1, \ldots, \tau_5$ が満たすべき等式を以下のように書き下すことができる。

$$\begin{aligned}\tau_1 &= \tau_2 \longrightarrow \tau_3 \longrightarrow \tau_5\\ \tau_2 &= \tau_3 \longrightarrow \tau_4\\ \tau_2 &= \tau_4 \longrightarrow \tau_5\end{aligned}$$

2 番目と 3 番目の等式から、以下のような等式集合が導ける。

$$\begin{aligned}\tau_1 &= \tau_2 \longrightarrow \tau_3 \longrightarrow \tau_5\\ \tau_2 &= \tau_3 \longrightarrow \tau_4\\ \tau_3 &= \tau_4\\ \tau_4 &= \tau_5\end{aligned}$$

右辺の型に対するさらなる等式がない、解けた形の等式 $\tau_4 = \tau_5$ を使い、変数 τ_4 を消去すると、

$$\begin{aligned}\tau_1 &= \tau_2 \longrightarrow \tau_3 \longrightarrow \tau_5\\ \tau_2 &= \tau_3 \longrightarrow \tau_5\\ \tau_3 &= \tau_5\end{aligned}$$

を得る。これを繰り返すと、

$$\tau_1 = (\tau_5 \longrightarrow \tau_5) \longrightarrow \tau_5 \longrightarrow \tau_5$$

を得る。τ_5 は、何も等式が存在しない自由変数である。この型を型変数’a とすると、

```
twice : (’a -> ’a) -> ’a -> ’a
```

を得る。

問 3.5 以下の各プログラムについて，もしそれが型を持たなければ（つまり型エラーを含めば）その理由を説明し，型を持てばそのの最も一般的な多相型を推定せよ.

1. `fun S x y z = (x z) (y z)`
2. `fun K x y = x`
3. `fun A x y z = z y x`
4. `fun B f g = f g g`
5. `fun C x = x C`
6. `fun D p a b = if p a then (b,a) else (a,b)`

解答例

1. `fun S x y z = (x z) (y z)`
   SML#は以下のような型を推論する．

   ```
   # fun S x y z = (x z) (y z);
   val S = fn : ['a,'b,'c.('a  -> 'b  -> 'c)  -> ('a  -> 'b)  -> 'a  -> 'c]
   ```

   (補足）この S は，コンビネータ論理において S コンビネータとして知られる基本関数である．さらに，この関数は，構成的論理学では，ヒルベルトシステムの公理 $(A \supset B \supset C) \supset (A \supset B) \supset (A \supset C)$ に対応している．興味のある読者は，大堀，ガリグ，西村,「コンピュータサイエンス入門，アルゴリズムとプログラミング言語」（岩波書店）の第 2 部を参照されたい．

2. `fun K x y = x`
   SML#は以下のような型を推論する．

   ```
   # fun K x y = x;
   val K = fn : ['a .'a  -> ['b .'b  -> 'a]]
   ```

   (補足）この K は，コンビネータ論理において K コンビネータとして知られる基本関数である．さらに，この関数は，構成的論理学では，ヒルベルトシステムの公理 $A \supset B \supset A$ に対応している．

3. `fun A x y z = z y x`
   SML#は以下のような型を推論する．

   ```
   # fun A x y z = z y x;
   val A = fn : ['a .'a  -> ['b .'b  -> ['c .('b  -> 'a  -> 'c)  -> 'c]]]
   ```

4. `fun B f g = f g g`
   SML#は以下のような型を推論する．

   ```
   # fun B f g = f g g;
   val B = fn : ['a,'b.('a  -> 'a  -> 'b)  -> 'a  -> 'b]
   ```

5. `fun C x = x C`
   この式は，C はを引数とする関数でありかつ x は C を引数とする関数であることを要求している．したがって．C の型は，C を引数とする関数を引数とする関数となり，自分自身を部分に含む型となってしまい，そのような性質をもつ型は有限な範囲では存在しない．

6. `fun D p a b = if p a then (b,a) else (a,b)`
   SML#は以下のような型を推論する．

   ```
   # fun D p a b = if p a then (b,a) else (a,b);
   val D = fn : ['a .('a  -> bool)  -> 'a  -> 'a  -> 'a * 'a]
   ```

問 3.6 以下の関数およびプログラムはどのような型を持つか.

```
fun f x = f x;
f 1;
```

さらに，2 番目の式の計算結果について考察せよ.

解答例 SML#は以下のような型を推論する.

```
# fun f x = f x;
val f = fn : ['a,'b.'a  -> 'b]
```

式 `f 1` の定義から，`f 1` を呼び出し続ける無限ループ式であることがわかる.

この性質は，式の型からも分析することができる．`f` の型から，`f 1` の型は`'a` であることがわかる．この型は，`f 1` の結果がすべての型を持ちうることを示している．したがって，この式の計算が修了し，結果を返せば，その結果は総ての型をもつ値である．しかし，すべての型どころか，`int` 型と `real` 型を同時にもつような値さえ存在しない．したがって，この式の計算が修了し結果を返すことはない.

3.3 明示的な型宣言

問 3.7 以下の関数の型を推定せよ.

1. `fun f x y z = x y z : int`
2. `fun f x y z = x (y z) : int`
3. `fun f x y z = (x y z) : int`
4. `fun f x y z = x y (z : int)`
5. `fun f x y z = x (y z : int)`

解答例 たとえば SML#の推論結果以下の通りである.

```
# fun f x y z = x y z : int;
val f = fn : ['a,'b.('a  -> 'b  -> int)  -> 'a  -> 'b  -> int]
# fun f x y z = x (y z) : int;
val f = fn : ['a .('a  -> int)  -> ['b .('b  -> 'a)  -> 'b  -> int]]
# fun f x y z = (x y z) : int;
val f = fn : ['a,'b.('a  -> 'b  -> int)  -> 'a  -> 'b  -> int]
# fun f x y z = x y (z : int);
val f = fn : ['a,'b.('a  -> int  -> 'b)  -> 'a  -> int  -> 'b]
# fun f x y z = x (y z : int);
val f = fn : ['a .(int  -> 'a)  -> ['b .('b  -> int)  -> 'b  -> 'a]]
```

問 3.8 関数 `fun f x = K x (fn y => x (x 1))` の動作と型を説明せよ．ただし K は問 3.5 で定義した関数である．

解答例 K は 2 番目の引数を捨てて 1 番目の引数を返す関数であるから，`fun f x = K x (fn y => x (x 1))` は x を受け取り x を返す関数である．従って，x の型を A とすると，f の型は `A -> A` である．ただし，`(fn y => x (x 1))` があるため x の型は `int -> int` に制限される．そこで，f の型は `(int -> int) -> int -> int` となる．

実際 SML#は以下のような型を推論する．

```
# fun f x = K x (fn y => x (x 1));
val f = fn : (int  -> int)  -> int  -> int
```

問 3.9 問 3.8 の考察から理解されるように，明示的な型宣言を使用しなくても型の制約を加えることができる場合がある．型変数を含まない型 τ を持つ式 exp_τ が与えられているとする．

1. τ より一般的な多相型を持つ式 exp が与えられたとき，その式の動作を変えずに，その式の型を τ 制限する式の定義を与えよ．
2. 恒等関数と同一の動作をし，かつ型が τ `->` τ である関数を定義せよ．

解答例

1. 簡単には，

   `if true then exp else` E_τ

   とすればよい．if 分を使わずに，より純粋なラムダ式で表現したければ，以下の手順で構築できる．式 exp の型を式 E_τ の型と強制的に一致させるには，それら二つの式を，それらの型が等式で結ばれるような式の中に埋め込めばよい．以下はその典型的な例である．

   `(fn y => fn z => z (y` E_τ`) (y` exp`))`

   この関数は exp と同一の動作をせず，要求を満さない．求める式は，評価すると exp が得られるような式でなければならない．そこで，上記の exp の部分を引数とし，

   `(fn x => ... (fn y =>fn z => z (y` E_τ`) (y` exp`)) ... )` exp

   と変形することを考える．関数の本体全体は x と同一の意味をもつ式である必要がある．そこで，`val K = fn x => fn y` を使い，型を一致させるだけに導入した上記の部分を捨て，exp 自身を返すようにすればよい．以下が要求を満す式の例である．

   `(fn x => K x (fn y =>fn z => z (y E<sub>τ</sub>) (y exp))) exp`

2. 上記を応用すれば，以下のような例が考えられる．

```
fn x => K x (fn y =>fn z => z (y E_τ) (y x))
```

たとえば，`fn x => K x (fn y =>fn z => z (y 1) (y x))`¿は `int -> int` 型の関数である．

問 3.10 `higherTwice` と同一の動作をし型が等しい関数を，明示的な型制約を加えずに定義せよ．

解答例 以下にコード例と SML#で推論された型を示す。

```
# fun higherTwice f x = f (f (K x (fn y => (y x, y id))));
val higherTwice = fn : ['a .(('a  -> 'a)  -> 'a  -> 'a)  -> ('a  -> 'a)  -> 'a  -> 'a]
```

3.4 関数名の多重定義

問 3.11 以下の各式の型を推定せよ．

1. `fn x => x > 1`
2. `fn x => fn y => fn z => (x y,x "Ada",y > z)`
3. `fn x => fn y => y (x > x)`

解答例 SML#での推論例を示す．

```
# fn x => x > 1;
val it = fn : int  -> bool
# fn x => fn y => fn z => (x y,x "Ada",y > z);
val it = fn : ['a .(string  -> 'a)  -> string  -> string  -> 'a * 'a * bool]
# fn x => fn y => y (x > x);
val it = fn : int  -> ['a .(bool  -> 'a)  -> 'a]
```

3.5 多相型の使用の制限

問 3.12 `fn f => twice twice f` が関数を引数に 4 回適用する高階の関数であることを確かめよ．`fn f => twice fourTimes f` および `fn f => fourTimes twice f` はそれぞれどのような関数か．

解答例 例えば以下のように確かめることができる．

```
# fun printStar () = print "*";
val printStar = fn : unit  -> unit
# twice twice printStar ();
****val it = () : unit
```

以下の例からわかるように、`fn f => twice fourTimes f` および `fn f => fourTimes twice f` は、どちらも関数を 16 回適用する高階関数である.

```
# twice fourTimes printStar ();
****************val it = () : unit
# fourTimes twice printStar ();
****************val it = () : unit
```

（補足）`twice fourTimes f x` は，その定義から，`fourTimes (fourTimes f) x` である．さらに `fourTimes` の定義にしたがって展開すると `(fourTimes f) ((fourTimes f) ((fourTimes f) (fourTimes f x)))` となる．このことから，`twice fourTimes f x` は `f` を `x` に 4^2 回適用する関数となることがわかる．一般に，関数を m 回適用する関数を M，関数を n 回適用する関数を N とすると `M N` は関数を n^m 回適用する関数となる．以下に $m = 4$, $n = 3$ の例を示す.

```
# fun threeTimes f x = f (f (f x));
val threeTimes = fn : ['a .('a  -> 'a)  -> 'a  -> 'a]
# fourTimes threeTimes printStar ();
********************************************************************************val it = () : un
```

問 3.13 以下の各式が値式であるか否か判定せよ.

1. `fn x => x 1`
2. `(fn x => x) 1`
3. `(fn x => x,fn x => x)`
4. `let fun f x = x in f end`
5. `let val a = 1 + 1 in fn x => x end`

解答例　以下のものがは値式である.

- `fn x => x 1`
- `(fn x => x,fn x => x)`

（補足）以下の各式は値式ではない.

- `let fun f x = x in f end`
- `let val a = 1 + 1 in fn x => x end`

しかし，ランク 1 多相性を実現している SML#では以下のように多相型が与えられる.

```
# let fun f x = x in f end;
val it = fn : ['a .'a  -> 'a]
# let val a = 1 + 1 in fn x => x end;
val it = fn : ['a .'a  -> 'a]
```

3.6 等値演算子の型の扱い

問 3.14 以下の各式について型が正しいか判定し，正しければ結果の型を予測せよ．

- `"M" = #"M"`
- `"ML" = "SML"`
- `fn x => (x,x) = (x,x)`
- `fn (x,f) =>(f (fn x => x), f x, x = x)`

解答例

- `"M" = #"M"`

  左辺は `string` 型，右辺は `char` 型であるから型エラーである．

- `"ML" = "SML"`

```
# "ML" = "SML";
val it = false : bool
```

- `fn x => (x,x) = (x,x)`

```
# fn x => (x,x) = (x,x);
val it = fn : [''a .''a  -> bool]
```

- `fn (x,f) =>(f (fn x => x), f x, x = x)`

  部分式 `f (fn x => x)` から `f` は関数を引数とする関数である．したがって，部分式 `f x` から `x` も関数である．すると，`x = x` は関数同士の比較となり，型エラーとなる．

問 3.15 整数の組を受け取り，それぞれのフィボナッチ数を計算しその組を返す関数を定義し，それにメモ関数を適用し，メモ関数が使用可能であることを確かめよ．

解答例

```
# memo;
val it = fn : [''a,'b.(''a  -> 'b)  -> ''a  -> ''a  -> 'b]
# fun f (x,y) = (fib x, fib y);
val f = fn : int * int  -> int * int
# val f = memo f (30,31);
val f = fn : int * int  -> int * int
# f (30,31);
val it = ( 2178309, 832040 ) : int * int
# f (31,30);
val it = ( 832040, 2178309 ) : int * int
```

を実行すると，`f (30,31)` は瞬時に修了することが確認できる．

問 3.16 問 2.22 における集合を表現する型は `int -> bool` のみならず，`''a -> bool` に一般化でき，したがって，問 2.22 で作成したプログラムは，eqtype に属する型であれば，どのような型の集合としても使用できることを確かめよ．

解答例 以下に SML#での例を示す．

```
# val emptySet = fn x => false;
val emptySet = fn : ['a .'a  -> bool]
# fun singleton n = fn x => x = n;
val singleton = fn : [''a .''a  -> ''a  -> bool]
# fun member n S = S n;
val member = fn : ['a .'a  -> ['b .('a  -> 'b)  -> 'b]]
# fun insert n S = fn x => x = n orelse S x;
val insert = fn : [''a .''a  -> (''a  -> bool)  -> ''a  -> bool]
# fun union S1 S2 = fn x => S1 x orelse S2 x;
val union = fn : ['a .('a  -> bool)  -> ('a  -> bool)  -> 'a  -> bool]
# fun intersection S1 S2 = fn x => S1 x andalso S2 x;
val intersection = fn : ['a .('a  -> bool)  -> ('a  -> bool)  -> 'a  -> bool]
# fun difference S1 S2 = fn x => S1 x andalso not (S2 x);
val difference = fn : ['a .('a  -> bool)  -> ('a  -> Bool.bool)  -> 'a  -> bool]
# singleton (1,2);
val it = fn : int * int  -> bool
# val S = singleton (1,"1");
val S = fn : int * string  -> bool
# val S = insert (2,"2") S;
val S = fn : int * string  -> bool
# member (1,"1") S ;
val it = true : bool
# member (1,"2") S ;
val it = false : bool
```

第4章　MLの基本データ型

4.1　単位型 (eqtype unit)

問 4.1 `before` と `ignore` はこれまでに学んだ構文を使って定義可能である．exp_1，exp_2 を任意の式とするとき，`ignore` exp_1 および exp_1 `before` exp_2 それぞれについて，同一の動作をし同一の型を持つ構文を，`ingore` および `before` を使わずに定義せよ．

解答例　以下にコード化の例を示す。

- `ignore` exp_1

  `(fn _ => ())` exp_1

- exp_1 `before` exp_2

  `(fn x => fn () => x)` exp_1 exp_2

4.2　真理値型（eqtype bool）

問 4.2 以下のそれぞれの式を評価した結果を予測せよ．

- `false andalso true before print "Is this printed?\n"`
- `true andalso false before print "How about this one?\n"`
- `false orelse true before print "Another one?\n"`
- `false andalso true orelse true before print "One more?\n"`

解答例　SML#での実行結果は以下の通り．

```
# false andalso true before print "Is this printed?\n";
val it = false : bool
# true andalso false before print "How about this one?\n";
How about this one?
val it = false : bool
```

```
# false orelse true before print "Another one?\n";
Another one?
val it = true : bool
# false andalso true orelse true before print "One more?\n";
One more?
val it = true : bool
```

4.5　文字型（eqtype char）

問 4.3 ASCII 文字表現では `ord #"a"` $<$ `ord #"b"` $<$ $\cdots <$ `ord #"z"` かつ `ord #"0"` $<$ `ord #"1"` $<$ $\cdots <$ `ord #"9"` かつ `ord #"A"` $<$ `ord #"B"` $<$ $\cdots <$ `ord #"Z"` である．この事実のみを仮定して，字句解析処理などでよく使用する以下の各関数を定義せよ．

- `isSpace : char -> bool`
  文字がスペース文字（空白，改行文字，タブ，垂直タブ，フォームフィード）か否か判定する関数．
- `isAlpha : char -> bool`
  文字が英字か否かを判定する関数．
- `isNum : char -> bool`
  文字が数字か否かを判定する関数．
- `isAlphaNumeric : char -> bool`
  文字が英字または数字か否かを判定する関数．
- `isLower : char -> bool`
  文字が英小文字か否かを判定する関数．
- `isUpper : char -> bool`
  文字が英大文字か否かを判定する関数．
- `toLower : char -> char`
  文字が英大文字なら英小文字に変換し，それ以外ならその文字をそのまま返す関数．
- `toUpper : char -> char`
  文字が英小文字なら英大文字に変換し，それ以外ならその文字をそのまま返す関数．

解答例　コード例を以下の示す。

```
fun isSpace c =
    case c of
         #" " => true
       | #"\t" => true
       | #"\n" => true
```

```
        | #"\v" => true
        | #"\f" => true
        | _ => false

fun isAlpha c = c >= #"a" andalso c <= #"z" orelse
                c >= #"A" andalso c <= #"Z"

fun isNum c = c >= #"0" andalso c <= #"9"

fun isAlphaNumeric c = isAlpha c orelse isNum c

fun isLower c = #"a" <= c andalso c <= #"z"

fun isUpper c = #"A" <= c andalso c <= #"Z"

fun toLower c =
    if isUpper c
    then chr (ord #"a" + (ord c - ord #"A"))
    else c

fun toUpper c =
    if isLower c
    then chr (ord #"A" + (ord c - ord #"a"))
    else c
```

4.6 文字列型（eqtype string）

問 4.4 文字列に含まれる英大文字をすべて英小文字に変換する関数 lower，および英小文字をすべて英大文字に変換する関数 upper を定義せよ．

解答例 以下にコード例を示す。

```
fun lower s = implode (map toLower (explode s))
fun upper s = implode (map toUpper (explode s))
```

（補足）Standard ML Basis Library の中の String.map 関数を使えば、より簡潔かつ効率よい以下のコードが可能。

```
fun lower s = String.map toLower s
fun upper s = String.map toUpper s
```

問 4.5 第一引数で与えられた文字列が，第二引数で与えられた文字列の先頭部分文字列になっているか否かを判定する関数 isPrefix : string -> string -> bool を書け．ただし，空文字列はすべての文字列の先頭部分文字列とする．

解答例 `fun isPrefix s1 s2 = s1 = substring(s2, 0, size s1)`

問 4.6 `advance` を定義せよ.

解答例 コード例を以下にしめす。このコードは、`s1`, `s2`, `from1`, `from2`, `makIndex1`, `maxIndex2` が定義された環境で動作するプログラムである。

```
fun advance n =
   if from1 + n <= maxIndex1 andalso from2 + n <= maxIndex2 then
      if substring (s1,from1+n,1) = substring (s2,from2+n,1)
          then advance (n+1)
      else n
   else n
```

問 4.7 開始位置の `s1` と `s2` の大きさによる制限に注意して，`nextStartPos` を上記の順序関係を用いて定義し，`match` を完成させよ.

解答例 コード例を以下にしめす。このコードは、`maxIndex2` が定義された環境で動作するプログラムである。

```
fun nextStartPos (i,j) = if j = maxIndex2 then (i+1,0) else (i,j+1)
```

第5章　レコード

5.3　パターンマッチングによるレコードの操作

問 5.1 レコードパターンとフィールド取り出し演算子は，どちらか一方があれば他を実現できる．

1. フィールド取り出し演算 `#`l と同一の型を持ち，同じ動作をする関数式を，レコードパターンを用いて定義せよ．
2. レコードパターンを含む関数式 `fn` $\{l_1$`=`x_1`,`l_2`=`x_2`,`$\cdots$`,`l_n`=`x_n`,...}` `=>` exp と同一の型を持ち同じ動作をする関数式を，フィールド取り出し演算子を用いて定義せよ．ただし，上記式の中の $\cdots$ は複数のフィールドパターンを表すメタ表現，`...` は ML のレコードパターンの文法の一部である．(ヒント：式 exp の中では x_1 から x_n までの変数が使用されているはずである．この点を考えて，`let val` x_1`=`exp_1 $\cdots$ `val` x_n`=`exp_n `in` exp `end` の形の式を含む式を定義することを考えよ．ここで，exp は与えられた式，exp_1 から exp_n は適当に導入する式である．)

解答例

- `#l` と同等の関数式。SML#のレコード多相を用いれば、実際に関数として定義可能である。以下はその例である。

```
# val sharp_l = fn {l,...} => l;
val sharp_l = fn : ['a#{l: 'b}, 'b. 'a -> 'b]
```

- `fn` $\{l_1$`=`x_1`,`l_2`=`x_2`,`$\cdots$`,`l_n`=`x_n`,...}` `=>` exp と同値な式。
以下がコード例である。LaTeXML のタイプセットの困難から、`Xi`、`Li`, `EXP` をそれぞれ、メタ変数 x_i、l_i、exp として記述する。

```
fn Y =>
  let
     val X1 = #L1 Y
     val X2 = #L2 Y
        ...
     val Xn = #Ln Y
  in
    EXP
  end
```

5.5　組型

問 5.2 以下の各文の評価の結果は何か.

- `val {1=x,3={2=y,...},...} = (1,2,(3,4,5),6)`
- `(#2 o #3) (1,2,(3,4,5),6)`

ただし，`o` は，トップレベルで以下のように定義された，関数合成演算子である.

```
infix 3 o
fun op o(f,g) x = (f (g x))
```

解答例 (思考実行し結果を得ないと意味はないが、...)SML#での評価結果を以下に示す。

```
# val {1=x,3={2=y,...},...} = (1,2,(3,4,5),6);
val x = 1 : int
val y = 4 : int

# (#2 o #3) (1,2,(3,4,5),6);
val it = 4 : int
```

第6章　リスト

6.1　リスト構造

問 6.1 本問では，無限集合の簡単な扱いに慣れている読者のために，リストの数学的なモデルを考察する．ポインタ構造を無視すると，上記のリスト構造は以下のように入れ子になった組とみなせる．

$$(\mathrm{v1}, (\mathrm{v2}, \cdots (\mathrm{vn}, \mathrm{nil}) \cdots))$$

v1,v2,$\cdots$,vn が属する集合を A とし，集合 A と B の直積 $A \times B$ を以下のように定義する．

$$A \times B = \{(a, b) | a \in A, b \in B\}$$

さらに，nil を唯一の要素とする集合を Nil とする．すると，n 個の要素からなるリストは以下のような集合の要素と考えられる．

$$\underbrace{A \times (A \times (\cdots (A}_{n \text{ 個の } A} \times Nil) \cdots))$$

n 個に限らず，集合 A の要素からなるすべてのリストの集合は，集合に関する以下のような方程式の解と考えることができる．

$$L = Nil \cup (A \times L)$$

この方程式の最小解を以下の手順で求めよ．

1. 集合の系列 X_i を以下のように定める．

$$\begin{aligned} X_0 &= Nil \\ X_{i+1} &= X_i \cup (A \times X_i) \end{aligned}$$

   もし L に関する方程式が解を持てば，任意の i に対して，

$$X_i \subseteq L$$

   であることを i に関する数学的帰納法で示せ．

2. この系列を用いて，集合 X を以下のように定義する．

$$X = \bigcup_{i \geq 0} X_i$$

   このとき，X は上記の方程式を満たすことを示し，したがって，X が上記方程式の最小解であることを確認せよ．

解答例

1. $\mathcal{L}$ を方程式の任意の解とし、$X_i \subseteq \mathcal{L}$ を i に関する数学的帰納法で示す。

   $(i = 0)$ の場合。

$$\begin{aligned} X_0 &= Nil \\ &\subseteq Nil \cup (A \times \mathcal{L}) \\ &= \mathcal{L} \end{aligned}$$

   $(i > k + 1)$ の場合。

$$\begin{aligned} X_{k+1} &= X_k \cup (A \times X_k) \\ &\subseteq \mathcal{L} \cup (A \times \mathcal{L}) \\ &= \mathcal{L} \end{aligned}$$

2.

$$X = Nil \cup A \times X$$

   を示せば良い。集合の等式であるから、$X \subseteq Nil \cup (A \times X)$ および $Nil \cup (A \times X) \subseteq X$ を示せばよい。

   $X \subseteq Nil \cup (A \times X)$ は、$X = \bigcup_{i \geq 0} X_i$ の要素集合 X_i について、$X_i \subseteq Nil \cup (A \times X)$ を示せばよい。定義の形から、ほぼ自明である。厳密には、i に関する帰納法による。$X_0 = Nil \subseteq Nil \cup (A \times X)$ であり、成立する。$i = K + 1$ の場合、$X_{k+1} = X_k \cup A \times X_k$ である。帰納法の仮定から $X_k \subseteq Nil \cup (A \times X)$ である、また、定義から、$A \times X_k \subseteq A \times X$ であり、従って、$X_{k+1} \subseteq A \times X$ である。

   $Nil \cup A \times X \subseteq X$ を示すには、$Nil \subseteq X$ と任意の i に関して、$A \times X_i \subseteq X$ を示せば十分である。前者は、$Nil = X_0 \subseteq X$ であり、成立する。後者は、定義より、$A \times X_i \subseteq X_i \cup A \times X_i = X_{i+1} \subseteq X$ であり成立する。

6.2　リスト型（τ list）

問 6.2 以下の各関数について型が正しいか判定し，正しければその型を推定し，正しくなければ型エラーの原因を考察せよ．

- `fun L1 a = a :: a`
- `fun L2 a = a :: [a]`
- `fun L3 (a,b) = a :: b`
- `fun L4 (a,b) = (a :: b,b :: a)`
- `fun L5 (a,b) = [a] :: b`
- `fun L6 (a,b) = a :: [b]`

- `fun L7 (a,b) = (a,b)::[(a,b)]`

解答例 以下、型が正しければ、SML#が推論する型を示す、正しくなければその理由を記す。

- `fun L1 a = a :: a`
  型エラーである。引数 a が τ と τ list として使用されている。

- `fun L2 a = a :: [a]`

```
# fun L2 a = a :: [a];
val L2 = fn : ['a. 'a -> 'a list]
```

- `fun L3 (a,b) = a :: b`

```
# fun L3 (a,b) = a :: b;
val L3 = fn : ['a. 'a * 'a list -> 'a list]
```

- `fun L4 (a,b) = (a :: b,b :: a)`
  型エラーである。引数 a と b は、a :: b は、「b の型は τ list で、a の型は τ」であるという関係を b :: a は、「a の型は τ' list で、b の型は τ'」であるという関係をそれぞれ要求するが、これら条件を同時に満たす（有限の）型は存在しない。

- `fun L5 (a,b) = [a] :: b`

```
# fun L5 (a,b) = [a] :: b;
val L5 = fn : ['a. 'a * 'a list list -> 'a list list]
```

- `fun L6 (a,b) = a :: [b]`

```
# fun L6 (a,b) = a :: [b];
val L6 = fn : ['a. 'a * 'a -> 'a list]
```

- `fun L7 (a,b) = (a,b)::[(a,b)]`

```
# fun L7 (a,b) = (a,b)::[(a,b)];
val L7 = fn : ['a, 'b. 'a * 'b -> ('a * 'b) list]
```

6.3　パターンマッチングによるリストの分解

問 6.3 zip，unzip および last を case 構文を使わずに定義せよ．

解答例

```
fun zip (nil,_) = nil
  | zip (_,nil) = nil
  | zip (h1::t1, h2::t2) = (h1,h2) :: zip (t1, t2)

fun unzip nil = (nil,nil)
  | unzip ((a,b)::t) = let val (A,B) = unzip t in (a::A, b::B) end

fun last [a] = a
  | last (_::t) = last t
```

6.4　リスト処理の基本関数

問 6.4 null，hd，tl の各関数をパタンマッチングを使って定義せよ．

解答例

```
fun null nil = true
  | null (h::_) = false

fun hd (h::_) = h

fun tl (_::t) = t
```

問 6.5 @，rev，map の各関数をパタンマッチングを使わずに定義せよ．

解答例

```
fun op @ (L1,L2) = if null L1 then L2 else hd L1 :: (tl L1 @ L2)

fun rev L = if null L then nil else rev (tl L) @ [hd L]

fun map f L = if null L then nil else f (hd L) :: map f (tl L)
```

問 6.6 以下のリスト処理関数を定義せよ．

1. 整数のリストの総和を求める関数 sumList．
2. 要素がリストに存在するか否かを判定する関数 member．

3. リスト中の重複する要素を取り除く関数 unique.

4. 'a -> bool 型の関数 P と 'a list 型のリスト L を受け取り，リスト L から関数 P が true を返す要素のみを取り出す関数 filter.

5. リストのリストを 1 つのリストにする関数 flatten．たとえば,

```
- flatten [[1],[1,2],[1,2,3]];
val it = [1,1,2,1,2,3] : int list
```

   のような動作をする.

6. 文字列のリストと文字列の組を受け取り，各リストの要素を文字列でつないだ文字列を返す関数 splice．たとえば,

```
- splice (["","home","ohori","papers","mltext"],"/");
val it = "/home/ohori/papers/mltext" : string
```

   のような動作をする.

解答例

```
fun sumList nil = 0
  | sumList (h::t) = h + sumList t

fun member a nil = false
  | member a (h::tl) = a = h orelse member a tl

fun unique nil = nil
  | unique (h::tl) =
    let
      val tl' = unique tl
    in
      if member h tl' then tl' else h::tl'
    end

fun filter P nil = nil
  | filter P (h::tl) =
    if P h then h :: filter P tl else filter P tl
```

```
fun flatten nil = nil
  | flatten (h::tl) = h @ flatten tl

fun splice (nil,_) = ""
  | splice ([x],_) = x
  | splice ((h::t),s) = h ^ s ^ splice (t,s);
```

問 6.7 以下のリスト処理関数を定義せよ.

1. 与えられたリストの先頭から始まるすべての部分リストを返す関数 prefixList および最後までのすべての部分リストを返す関数 suffixList. たとえば

```
- prefixList [1,2,3];
val it = [[1],[1,2],[1,2,3]]: int list list
- suffixList [1,2,3];
val it = [[1,2,3],[2,3],[3],[]]: int list list
```

   のような動作をする.

2. 与えられたリストの中のすべての部分リストを返す関数 allIntervals. たとえば,

```
- allIntervals [1,2,3];
val it = [[1],[1,2],[1,2,3],[2],[2,3],[3]] : int list list
```

   のような動作をする.（ヒント：suffixList, prefixList, map, flatten を組み合わせることを考えよ.）

3. 与えられたリストを集合とみなし，そのすべての部分集合を返す関数 powerSet. ただし，与えられたリストは重複を含まないと仮定してよい.

4. 与えられたリストを集合とみなし，その中から与えられた数の要素を取り出して作られるすべての順列をリストとして返す関数 allPermutations. たとえば

```
- allPermutations [1,2,3] 2;
val it = [[1,2],[2,1],[1,3],[3,1],[2,3],[3,2]]: int list list
```

   のような動作をする.（ヒント：flatten, powerSet, permutations を組み合わせることを考えよ.）

解答例

```
fun prefixList nil = nil
  | prefixList (h::tl) =
    let
      val L = prefixList tl
    in
       [h] :: map (fn y => h::y) L
    end

fun suffixList nil = [[]]
  | suffixList (h::t) =
    let
       val L = suffixList t
    in
       (h::t)::L
    end;

fun allIntervals L = flatten (map prefixList (suffixList L))

fun powerSet nil = [nil]
  | powerSet (h::t) =
    let
       val PT = powerSet t
    in
       map (fn x => h::x) PT @ PT
    end

fun permutations L =
    let fun insertAll s nil = [[s]]
          | insertAll s (h::t) =
              let val L = insertAll s t
              in (s::(h::t)) :: (map (fn x => h::x) L)
              end
    in foldr (fn (x,y) => foldr (fn (a,b) => insertAll x a @ b) nil y)
             [nil]
             L
    end;

fun allPermutations L n =
```

```
let val subs = filter (fn x => length x = n) (powerSet L)
in flatten (map  permutations subs)
end;
```

6.5　リスト処理の一般構造と汎用のリスト処理関数

問 6.8 以下の各関数が行う処理に関して上記の分析を行い，対応する Z と f を ML の式として定義せよ.

- `map`
- `@`
- `flatten`
- `filter`
- `permutations`

解答例

- `map` f L
  リストが `nil` の時は `nil`、リストが h::t の形なら，部分リスト t に対する値 R を計算すると、R は、t に f を `map` した結果であるから、全体の結果は `f h ::` R である。従って、$Z =$ `nil`、$f =$ `fn (h,R) => f h ::  R` である。

- $L1$ `@` $L2$
  第一要素 `L1` で `fold` する。`L1` が `nil` の時は第 2 要素 `L2`、`L1` が h::t の形なら，R は `t` と `L2` の連結結果のはずであるから、全体の結果は `h ::` R である。従って、$Z =$ `L2`、$f =$ `fn (h,R) => h ::  R` である。

- `flatten L`
  リストが `nil` の時は `nil`、リストが h::t の形なら，部分リスト t に対する値 R を計算すると、h はリスト、R は t を `flatten` した結果のはずであるから、全体の結果は `h @` R である。従って、$Z =$ `nil`、$f =$ `fn (h,R) => h @ R` である。

- `filter P L`
  リストが `nil` の時は `nil`、リストが h::t の形なら，部分リスト t に対する値 R を計算すると、R は t を `filter` した結果のはずであるから、全体の結果は、もし P `h` が `true` なら `h @` R、`folse` なら R である。従って、$Z =$ `nil`、$f =$ `fn (h,R) => if P h then h::R else R` である。

- `permutations`
  リストが `nil` の時は `[nil]`、リストが h::t の形なら，部分リスト t に対する値 R を計算すると、R は t を `permutation` した結果のはずであるから、全体の結果は、R の各要素に対して、その可能な位置に h を挿入したリストすべてである。従って、$Z =$ `[nil]`、$f =$ `fn (h,R) => foldr (fn (a,b) => insertAll h a @ b) nil R` である。

問 6.9 以下のリスト処理関数を foldr を使って定義せよ.

1. map, flatten, member, unique, prefixList, permutations の各関数.
2. リストと条件（bool 型の関数）を受け取り，リストの中に条件を満たす要素が存在するか否か判定する関数 exists および，リストの中のすべての要素が条件を満たすか判定する関数 forall. 定義上，任意の条件関数 P に対して，exists P nil は false，forall P nil は true を返すことに注意せよ.
3. 整数リストのプレフィックス和を求める関数 prefixSum. ただし，整数リスト $[a_1, a_2, \cdots, a_n]$ のプレフィックス和は $[a_1, a_1 + a_2, \cdots, a_1 + \cdots a_{n-1}, a_1 + \cdots + a_n]$ である.

解答例

```
fun map f L = foldr (fn (h,R) => f h :: R) nil L

fun flatten L = foldr (fn (h,R) =>  h @ R) nil L

fun member e L = foldr (fn (h,R) => R orelse e = h) false L

fun unique L =
    foldr (fn (h,R) => if member h R then R else h :: R) nil L

fun prefixList L =
    foldr (fn (x,l)=> [x]::(map (fn y => x::y) l)) nil L;

fun permutations L =
    let fun insertAll s nil = [[s]]
          | insertAll s (h::t) =
             let val L = insertAll s t
             in (s::(h::t)) :: (map (fn x => h::x) L)
             end
    in foldr (fn (x,y) => foldr (fn (a,b) => insertAll x a @ b) nil y)
             [nil]
             L
    end

fun exists p L =
    foldr (fn (y,r) => p y orelse r) false L

fun forall p L =
    foldr (fn (y,r) => p y andalso r) true L
```

```
fun prefixSum L =
    foldr (fn (x,y) => x::(map (fn z => z + x) y)) nil L
```

問 6.10 `foldr` の処理は，以下のような等式の結果を求めることと理解できる．

$$\texttt{foldr } f\ Z\ \texttt{[}a_1\texttt{,}a_2\texttt{,}\cdots\texttt{,}a_n\texttt{]} = f(a_1, f(a_2, f(\cdots, f(a_n, Z)\cdots)))$$

これに対して，以下のような動作をする `foldl` もトップレベルで定義されている．

$$\texttt{foldr } f\ Z\ \texttt{[}a_1\texttt{,}\cdots\texttt{,}a_{n-1}\texttt{,}a_n\texttt{]} = f(a_n, f(a_{n-1}, f(\cdots, f(a_1, Z)\cdots)))$$

1. `foldl` の定義を与えよ．

2. `foldl` を使って `rev` を定義せよ．

解答例

```
fun foldl f z nil = z
  | foldl f z (h::t) = foldl f (f (h,z)) t

fun rev L = foldl (op ::) nil L
```

問 6.11 関係のリスト表現で，同一の要素が一回しか含まれないとき，正規形と呼ぶことにする．上で定義した `tc` は，入力が正規形であっても結果は正規形とは限らない．

1. `tc` R が正規形とならない正規形のリスト R の例を挙げよ．
2. 正規形の入力に対しては，必ず正規形の結果を返すように，`tc` の定義を修正せよ．

解答例 正規形の入力 R に対して、`tc` R が正規形にならない例：`tc [(1,1),(1,2),(2,3)]`。
正規形を維持するための `tc` の改良：簡単な解決法は、

```
fun normalTc R = unique (tc R)
```

問 6.12 関係のリスト表現に関する以下の関数を定義せよ．

1. 関係 R と組 (a, b) に対して，$(a, b) \in R$ か否かを判定する関数 `isRelated`.
2. 関係 R と点 a に対して，集合 $\{x \mid (a, x) \in R\}$ を求める関数 `targetOf`.
3. 関係 R と点 a に対して，集合 $\{x \mid (x, a) \in R\}$ を求める関数 `sourceOf`.
4. 関係 R の逆 R^{-1} を求める関数 `inverseRel`.

解答例

```
fun isRelated (R,a) = member a R

fun targetOf (R,a) = filter (fn (x,y) => x = a) R

fun sourceOf (R,a) = filter (fn (x,y) => y = a) R

fun inverseRel R = map (fn (a,b) => (b, a)) R
```

第7章　データ構造の定義と利用

7.1　datatype文によるデータ型の定義

問 7.1 pre-order 表現に使用する区切り文字を固定すると，任意の 2 分木は，唯一の pre-order 表現を持つことを示せ．(ヒント：以下の 2 つの性質を示せばよい. (1) 任意の 2 分木は pre-order 表現を持つ. (2) 2 分木 T と T' が同一の pre-order 表現をもてば，$T = T'$ である．)

解答例　2 分木の構造に関する帰納法でしめす。

1. 木が `Empty` の時。空文字列が唯一の pre-order 表現である。
2. 木が `Node(a, T, T')` の時。帰納法の仮定より、T および T' はそれぞれ唯一の pre-order 表現 P, P' をもつ。与えられた木は pre-order 表現 $a(P)(P')$ を持つ。P, P' が唯一の表現であるから、この表現も唯一である。

問 7.2 `searchLP` および `searchRP` を書き，`decompose`，さらに `fromPreOrder` を完成させ，テストを行え.

解答例　`searchLP` および `searchRP` の例を以下にしめす。

```
fun searchLP s p =
    if substring(s,p,1) = "(" then p
    else searchLP s (p+1)

fun searchRP s p n =
    case substring(s,p,1) of
        "(" => searchRP s (p+1) (n+1)
      | ")" => if n=0 then p else searchRP s (p+1) (n - 1)
      | _ => searchRP s (p+1) n
```

問 7.3 2 分木の文字列表現には，上記の pre-order 表現以外に，以下の 2 つの表現がある.

- post-order 表現.
  2 分木を（1）左部分木, (2）右部分木, (3）ルートの順にたどって得られる文字列に対応する表現．たとえば，図 tree.pdf に示した木は (()()b)((()()d)()c)a と表される.
- in-order 表現.
  2 分木を（1）左部分木, (2）ルート, (3）右部分木の順にたどって得られる文字列に対応する表現．たとえば図 tree.pdf に示した木は (()b())a(((d())c()) と表される.

区切り文字を固定するならば，2 分木は，唯一の post-order 表現および in-order 表現を持つことを示せ.

これら 2 つの表現から `string tree` 型データを生成する関数，`fromPostOrder` および `fromInOrder` を定義せよ.

解答例 唯一の post-order 表現および in-order 表現を持つことの証明は、問 7.1 における pre-order 表現の場合と同様である。

`fromPostOrder` および `fromInOrder` を書くためには、対応する `decompose` を書く必要がある。それらの定義例を以下に示す。

```
fun decomposeIn s =
   let val lp1 = searchLP s 0
       val rp1 = searchRP s (lp1+1) 0
       val lp2 = searchLP s  rp1
       val rp2 = searchRP s (lp2+1) 0
   in {root=substring (s,rp1+1,lp2 - rp1 - 1),
       left=substring (s,lp1+1,rp1-lp1 -1),
       right=substring (s,lp2+1,rp2 - lp2-1)}
   end

fun fromInOrder s =
    if s = "" then Empty
    else let val {root,left,right} = decomposeIn s
         in Node(root,fromInOrder left,fromInOrder right)
         end

fun decomposePost s =
   let val lp1 = searchLP s 0
       val rp1 = searchRP s (lp1+1) 0
       val lp2 = searchLP s  rp1
       val rp2 = searchRP s (lp2+1) 0
   in {root=substring (s,rp2+1,size s - rp2 - 1),
       left=substring (s,lp1+1,rp1-lp1 -1),
       right=substring (s,lp2+1,rp2 - lp2-1)}
   end

fun fromPostOrder s =
    if s = "" then Empty
    else let val {root,left,right} = decomposePost s
         in Node(root,fromPostOrder left,fromPostOrder right)
         end
```

問 7.4 上の例では,「空の木」を使って 2 分木を定義したが,「子を持たない木」を使った以下のような定義も可能である.

1. データ v のみからなるノード $Leaf(v)$ は 2 分木である.
2. v がデータ，T_1, T_2 が 2 分木なら，$Node(v, T_1, T_2)$ は，v をノードのデータ，T_1, T_2 を左右の部分木とする 2 分木である.
3. v がデータ，T が 2 分木なら，$NodeL(v, T)$ は，v をノードのデータ，T を左の部分木とする 2 分木である.
4. v がデータ，T が 2 分木なら，$NodeR(v, T)$ は，v をノードのデータ，T を右の部分木とする 2 分木である.

この定義に対応するデータ型`'a newTree` を定義せよ.

解答例 問題の定義をそのままコードすると、以下の型定義を得る。

```
datatype 'a newTree
  = Leaf of 'a
  | Node of 'a * 'a newTree * 'a newTree
  | NodeL of 'a * 'a newTree
  | NodeR of 'a * 'a newTree
```

7.2 パターンマッチングを用いたデータ構造の利用

問 7.5 2 分木に対する以下の各関数を定義せよ.

1. ノードの総数を求める関数 `nodes`.
2. 型が `int tree` である木を対象とし，ノードの値の総和を求める関数 `sumTree`.
3. 引数で与えられた関数を木の各ノードのデータに適用して得られる新しい木を作る関数 `mapTree`.
4. `string tree` 型データを，前節で解説した post-order 表現，および in-order 表現に変換する以下の各関数.

```
toPostOrder :  string tree -> string
toInOrder   :  string tree -> string
```

文字列の形式は，以前同様とする.

解答例

```
fun nodes Empty = 0
  | nodes (Node(_, L, R)) = 1 + nodes L + nodes R

fun sumTree Empty = 0
  | sumTree (Node(n, L, R)) = n + sumTree L + sumTree R

fun mapTree f Empty = Empty
  | mapTree f (Node(x, L, R)) = Node(f x, mapTree f L, mapTree f R)

fun toPostOrder Empty = ""
  | toPostOrder (Node(a,b,c)) =
      "(" ^ toPostOrder b ^ ")"
    ^ "(" ^ toPostOrder c ^ ")"
    ^  a

fun toInOrder Empty = ""
  | toInOrder (Node(a,b,c)) =
      "(" ^ toInOrder b ^ ")"
    ^ a
    ^ "(" ^ toInOrder c ^ ")"
```

問 7.6 本問では `string tree` 型データと pre-order，in-order，post-order の各文字列表現との変換処理を，以下の方針に従い，7.1 節で定義した多相型の `'a newTree` 型に一般化することを考える．

1. `'a newTree` 型の文字列表現の生成には，`'a` 型の文字列表現の生成が必要である．そこで，木の文字列表現生成関数を，`'a -> string` 型の関数を引数として取る高階関数とする．文字列表現からの復元関数も同様に，`string -> 'a` 型の関数を引数として取る高階関数とする．

2. `string tree` の場合，‘(’ と ‘)’ がノードの値に含まれないと仮定し，これらを，文字列表現における部分木の区切り記号として用いた．しかし，たとえば `int * int` 型のデータを考えれば明らかなように，一般にこの仮定は成立しない．区切り文字は，可能な限り，種々の文字列表現に現れないものを選ぶ必要がある．ここでは，文字列定数`"\000"` と`"\001"` の 2 つを利用することとする．さらに，特別な型を処理したい場合に備え，変数

   ```
   val lp = "\000"
   val rp = "\001"
   ```

   を定義し，プログラムではこれらの変数を使用することとする．

3. `'a newTree` では，子ノードが存在しないことを表現する必要がある．種々の表現が可能であるが，`string tree` の場合の関数の再利用性などを考慮し，子ノードが存在しないことを，`string tree` の場合の空の木の表現`"()"`に対応させ，文字列`"\000\001"`で表すことにする．

以上の方針に従い，文字列表現への変換とその逆変換を行う以下の関数を定義せよ．

```
val newTreeToPreOrder : ('a -> string) -> 'a newTree -> string
val newTreeToInOrder : ('a -> string) -> 'a newTree -> string
val newTreeToPostOrder : ('a -> string) -> 'a newTree -> string
val preOrderToNewTree : (string -> 'a) -> string -> 'a newTree
val inOrderToNewTree : (string -> 'a) -> string -> 'a newTree
val postOrderToNewTree : (string -> 'a) -> string -> 'a newTree
```

さらに，int * int 型と文字列との変換関数を定義し，(int * int) newTree 型に対して，上記の各関数のテストを行え．

解答例

```
datatype 'a newTree
  = Leaf of 'a
  | Node of 'a * 'a newTree * 'a newTree
  | NodeL of 'a * 'a newTree
  | NodeR of 'a * 'a newTree

val lp = "\000"
val rp = "\001"
val LP = #"\000"
val RP = #"\001"

fun searchLP s p =
    if substring(s,p,1) = lp then p
    else searchLP s (p+1)

fun searchRP s p n =
    if substring(s,p,1) = lp then searchRP s (p+1) (n+1)
    else if substring(s,p,1) = rp then
      if n=0 then p else searchRP s (p+1) (n - 1)
    else searchRP s (p+1) n

fun decompose s =
   let val lp1 = searchLP s 0
       val rp1 = searchRP s (lp1+1) 0
       val lp2 = searchLP s  rp1
       val rp2 = searchRP s (lp2+1) 0
   in (substring (s,0,lp1),
       substring (s,lp1+1,rp1-lp1 -1),
```

```
        substring (s,lp2+1,rp2 - lp2-1))
    end

fun decomposeIn s =
    let val lp1 = searchLP s 0
        val rp1 = searchRP s (lp1+1) 0
        val lp2 = searchLP s  rp1
        val rp2 = searchRP s (lp2+1) 0
    in (substring (s, rp1+1, lp2 - rp1 - 1),
        substring (s, lp1+1, rp1 - lp1 -1),
        substring (s, lp2+1, rp2 - lp2 - 1))
    end

fun decomposePost s =
    let val lp1 = searchLP s 0
        val rp1 = searchRP s (lp1+1) 0
        val lp2 = searchLP s  rp1
        val rp2 = searchRP s (lp2+1) 0
    in (substring (s,rp2+1,size s - rp2 - 1),
        substring (s,lp1+1,rp1-lp1 -1),
        substring (s,lp2+1,rp2 - lp2-1))
    end

fun newTreeToPreOrder toString t =
    let
      val preOrder = newTreeToPreOrder toString
    in
      case t of
        Leaf v => toString v ^ lp ^ rp ^ lp ^ rp
      | Node(v, L, R) => toString v ^ lp ^ preOrder L ^ rp ^ lp ^ preOrder R ^ rp
      | NodeL(v, L) => toString v ^ lp ^ preOrder L ^ rp ^ lp ^ rp
      | NodeR(v, R) => toString v ^ lp ^ rp ^ lp ^ preOrder R ^ rp
    end

fun newTreeToInOrder toString t =
    let
      val inOrder = newTreeToInOrder toString
    in
      case t of
```

```
          Leaf v => lp ^ rp ^ toString v ^ lp ^ rp
        | Node(v, L, R) => lp ^ inOrder L ^ rp ^ toString v ^ lp ^ inOrder R ^ rp
        | NodeL(v, L) => lp ^ inOrder L ^ rp ^ toString v ^ lp ^ rp
        | NodeR(v, R) => lp ^ rp ^ toString v ^ lp ^ inOrder R ^ rp
      end

  fun newTreeToPostOrder toString t =
      let
        val postOrder = newTreeToPostOrder toString
      in
        case t of
          Leaf v => lp ^ rp ^ lp ^ rp ^ toString v
        | Node(v, L, R) => lp ^ postOrder L ^ rp ^ lp ^ postOrder R ^ rp ^ toString v
        | NodeL(v, L) => lp ^ postOrder L ^ rp ^ lp ^ rp ^ toString v
        | NodeR(v, R) => lp ^ rp ^ lp ^ postOrder R ^ rp ^ toString v
      end

  fun toNewTree decomp fromString s =
      let
         val (root,left,right) = decomp s
         val toTree = toNewTree decomp fromString
      in
        case (left, right) of
          ("","") => Leaf (fromString root)
        | (_, "") => NodeL (fromString root, toTree left)
        | ("",_) => NodeR (fromString root, toTree right)
        | _ => Node (fromString root, toTree left, toTree right)
      end

  fun preOrderToNewTree fromString s = toNewTree decompose fromString s
  fun inOrderToNewTree fromString s = toNewTree decomposeIn fromString s
  fun postOrderToNewTree fromString s = toNewTree decomposePost fromString s
```

int * int 型に対する処理関の定義例。

```
  fun intIntToString (x : int * int) = Dynamic.format x
  fun stringToIntInt s =
      let
        val ss = Substring.full s
        val (x,y) = Substring.splitl (fn x => x <> #",") ss
        val x = Substring.dropl (fn x => x = #"(") x
```

```
        val x = Substring.string (Substring.takel Char.isDigit x)
        val y = Substring.dropl (fn x => x = #"," orelse Char.isSpace x) y
        val y = Substring.string (Substring.takel Char.isDigit y)
        val i1 = valOf (Int.fromString x)
        val i2 = valOf (Int.fromString y)
      in
        (i1,i2)
      end

  fun intIntNewTreeToPreOrder t = newTreeToPreOrder intIntToString t
  fun intIntNewTreeToInOrder t = newTreeToInOrder intIntToString t
  fun intIntNewTreeToPostOrder t = newTreeToPostOrder intIntToString t

  fun preOrderToIntIntNewTree s = preOrderToNewTree stringToIntInt s
  fun inOrderToIntIntNewTree s = inOrderToNewTree stringToIntInt s
  fun postOrderToIntIntNewTree s = postOrderToNewTree stringToIntInt s
```

以下は (int * int) newTree に対する SML#でのテスト例である。

```
# val T = Node((1,2), Leaf (2,3), NodeR((3,4), Leaf (4,5)));
val T =
  Node ((1, 2), Leaf (2, 3), NodeR ((3, 4), Leaf (4, 5))) : (int * int) newTree
# val preOrderOfT = intIntNewTreeToPreOrder T
# val inOrderOfT = intIntNewTreeToInOrder T;
val preOrderOfT =
  "(1, 2)\^@(2, 3)\^@\^A\^@\^A\^A\^@(3, 4)\^@\^A\^@(4, 5)\^@\^A\^@\^A\^A\^A"
  : string
# val inOrderOfT = intIntNewTreeToInOrder T;
val inOrderOfT =
  "\^@\^@\^A(2, 3)\^@\^A\^A(1, 2)\^@\^@\^A(3, 4)\^@\^@\^A(4, 5)\^@\^A\^A\^A"
  : string
# val postOrderOfT = intIntNewTreeToPostOrder T;
val postOrderOfT =
  "\^@\^@\^A\^@\^A(2, 3)\^A\^@\^@\^A\^@\^@\^A\^@\^A(4, 5)\^A(3, 4)\^A(1, 2)"
  : string
# val T1 = preOrderToIntIntNewTree preOrderOfT;
val T1 =
  Node ((1, 2), Leaf (2, 3), NodeR ((3, 4), Leaf (4, 5))) : (int * int) newTree
# val T2 = inOrderToIntIntNewTree inOrderOfT;
val T2 =
  Node ((1, 2), Leaf (2, 3), NodeR ((3, 4), Leaf (4, 5))) : (int * int) newTree
```

```
# val T3 = postOrderToIntIntNewTree postOrderOfT;
val T3 =
  Node ((1, 2), Leaf (2, 3), NodeR ((3, 4), Leaf (4, 5))) : (int * int) newTree
```

問 7.7 リストデータを処理するさまざまな関数は，高階関数 foldr を使って簡単に定義できた．2 分木に対しても，foldr に対応する高階関数があると便利である．'a tree を処理する汎用の高階関数 treeFold を書いてみよう．

まず treeFold の取るべき引数を考えてみよう．foldr の場合との対応から，以下の引数が必要であると推定できる．

1. 変換の対象とする木 t.
2. 木が Empty のときの値 z.
3. 左部分木 L と右部分木 R をそれぞれ treeFold した結果とノードのデータ x から，木 Node(x,L,R) の結果を計算する関数 f.

処理の対象となる木の型を 'a tree とし，結果の型を 'b とする．すると引数 z の型は 'b であるはずである．変換関数 f は，木構成子 Node の型と対応させて，'a * 'b * 'b -> 'b の型を持つ関数と定義すると，treeFold を使って種々の木変換関数を定義する上で都合がよい．以上より treeFold は以下のような関数として定義できる．

```
# fun treeFold f z Empty = z
    | treeFold f z (Node (x, L, R)) =
          ...
val treeFold = fn : ('a * 'b * 'b -> 'b) -> 'b -> 'a tree -> 'b
```

1. treeFold の定義を完成せよ.
2. 前節で定義した各関数を treeFold を使って定義せよ.

解答例 treeFold の定義例は以下の通り。

```
fun treeFold f z Empty = z
  | treeFold f z (Node(a,b,c)) = f (a,treeFold f z b,treeFold f z c);
```

上記の各関数は、treeFold を使って以下の通り定義できる。

```
fun nodes t = treeFold (fn (x,y,z) => 1 + y + z) 0 t
fun sumTree t = treeFold (fn (x,y,z) => x + y + z) 0 t
fun mapTree f t = treeFold (fn (x,y,z) => Node(f x, y, z)) Empty t
fun toPostOrder t =
    treeFold
      (fn (x,L,R) => "(" ^ L ^ ")" ^ "(" ^  R ^ ")" ^ x)
      ""
      t
```

```
fun toInOrder t =
    treeFold
      (fn (x,L,R) => "(" ^ L ^ ")" ^ x ^ "(" ^  R ^ ")")
      ""
      t
```

7.3　パターンマッチングの拡張機能

問 7.8 上記 f、つまり、

```
fun f Empty = true
  | f (Node(_,Empty,Empty)) = true
  | f (Node(_,x as Node _, y as Node _)) = f x andalso f y
  | f _ = false
```

が true を返す 2 分木はどのような木か？

解答例 Node(_, Empty, Node _) の形のノードを含まない木、つまり、完全 2 分木である。

7.4　システム定義のデータ型

問 7.9 'a list -> 'a option 型を持つ以下の関数を定義せよ.

1. リストの先頭要素を返す関数 car.
2. 先頭を除いたリストを返す関数 cdr.
3. リストの最後の要素を返す関数 last.

解答例

1. リストの先頭要素を返す関数 car
   ```
   fun car (h::_) = SOME h | car _ = NONE
   ```
2. 先頭を除いたリストを返す関数 cdr
   ```
   fun cdr (_::t) = SOME t | cdr _ = NONE
   ```
3. リストの最後の要素を返す関数 last
   ```
   fun last nil = NONE | last [h] = SOME h | last (_::t) = last t
   ```

7.5 データ型を使用したプログラミング例

問 7.10 キーと値の組のリストを受け取り，それらを含む辞書を作成する関数

```
val makeDict : (string, 'a) list -> 'a dict
```

を定義し，それを使って辞書を作成し，上記関数のテストを行え．

解答例 makeDict 関数の定義例は以下の通り。各関数のテストは省略。

```
fun makeDict L =
   foldr
     (fn ((key, v), dict) => enter(key, v, dict))
     Empty
     L
```

問 7.11 上記の例はキーを string 型に制限されている．この制限を取り除く 1 つの方法は，string 型の比較演算を直接含む enter と lookUp 関数を定義する代わりに，比較関数を引数として受け取り，enter と lookUp 関数に相当する関数を作り出す，以下の型を持つ高階の関数 makeEnter と makeLookUp を定義することである．

```
type ('a,'b) dict = ('a * 'b) tree
val makeEnter : ('a * 'a -> order) -> 'a * 'b * ('a,'b) dict -> ('a,'b) dict
val makeLookUp : ('a * 'a -> order) -> 'a * ('a,'b) dict -> 'b
```

上記の関数を定義し，テストを行え．

解答例 makeEnter の型は、foldl などと一緒に使うことを考えると、

```
val makeEnter : ('a * 'a -> order) -> ('a * 'b) * ('a,'b) dict -> ('a,'b) dict
```

の方が便利である。そこで、この型の関数として定義例を以下に示す。

```
fun makeEnter f ((k, v), Empty) = Node((k,v), Empty, Empty)
  | makeEnter f ((k, v), dict as Node((k',v'),L,R)) =
    (case f (k,k') of
        EQUAL => dict
      | GREATER => Node((k',v'), L, makeEnter f ((k,v), R))
      | LESS => Node((k',v'), makeEnter f ((k,v),L), R))
fun makeLookUp f (k, Empty) = NONE
  | makeLookUp f (k, dict as Node((k',v'),L,R)) =
    (case f (k,k') of
        EQUAL => SOME v'
      | GREATER => makeLookUp f (k, R)
      | LESS => makeLookUp f (k, L))
```

問 7.12 makeEnter と makeLookUp を使って，int 型データをキーとし，string 型データを値として持つ (int,string) dict 型の辞書の操作関数を作成し，テストを行え．

解答例 SML#での定義とテストの例を以下に示す。

```
# fun enterIntDict x = makeEnter Int.compare x;
val enterIntDict = fn : ['a. (int * 'a) * (int * 'a) tree -> (int * 'a) tree]
# fun lookUpIntDict x = makeLookUp Int.compare x;
val lookUpIntDict = fn : ['a. int * (int * 'a) tree -> 'a option]
# val D = foldl enterIntDict Empty [(1,"1"), (2, "2"), (3, "3")];
val D =
  Node
  ((1, "1"), Empty, Node ((2, "2"), Empty, Node ((..., ...), Empty, Empty)))
  : (int * string) tree
# val items = map (fn x => lookUpIntDict (x, D)) [1,2,3,4]
val items = [SOME "1", SOME "2", SOME "3", NONE] : string option list
```

7.6　無限なデータ構造の定義と利用

問 7.13 FILTER を使い naturalNumbers から偶数の無限リスト evenNumbers を定義しテストを行え．

```
# NTH 10000000  evenNumbers;
val it = 20000000 : int
```

となるはずである．

解答例 定義とテストの SML#セッションを以下に示す。

```
# val evenNumbers = FILTER (fn x => x mod 2 = 0) naturalNumbers;
val evenNumbers = CONS (0, fn) : int inflist
# NTH 10000000  evenNumbers;
val it = 20000000 : int
```

問 7.14 無限リストに対する以下の関数を定義せよ．

- 最初の n 個の要素を除いたリストを返す関数
  `DROP : int -> 'a inflist -> 'a inflist`
- 最初の n 個の要素を通常のリストにして返す関数
  `TAKE : int -> 'a inflist -> 'a list`
- n 番目の要素から始まる m 個の要素を通常のリストにして返す関数
  `VIEW : int * int -> 'a inflist -> 'a list`

解答例 定義例を以下に示す。

```
fun DROP 0 L = L
  | DROP n L = DROP (n - 1) (TL L);
fun TAKE 0 L = nil
  | TAKE n L = HD(L) :: TAKE (n - 1) (TL L);
fun VIEW (n,m) L = TAKE m (DROP n L);
```

問 7.15 'a inflist inflist 型のデータは，2 次元の無限配列とみなすことができる．このデータに関する以下のプログラムを作成せよ．

1. 外側のリストの n 番目を x 座標，内側のリストの m 番目を y 座標とする点 (x, y) を求める関数 point を定義せよ．
2. int * int -> 'a 型の関数 f が与えられたとき，f のグラフを表す'a inflist inflist 型のデータを返す関数 graph を定義せよ．たとえば以下のような動作をする．

   ```
   - fun f (x,y) = x + y;
   val f = fn : int * int -> int
   - point (10,15) (graph f);
   val it = 25 : int
   ```

3. 2 次元空間に配置された無限列は，図 enum.pdf に示す順に座標をたどることによって通し番号を付けることができる．'a inflist inflist 型のデータを，この通し番号に従い並べて得られる無限リストを求める関数 enumerate を定義せよ．

解答例 以下に各関数の定義例を示す。

```
fun point(x,y) L = NTH y (NTH x L);
fun graph f  =
    let fun fromx x =
            let fun fromy y  = CONS(f(x,y),fn () => fromy (y + 1))
            in CONS(fromy 0,fn () => fromx (x + 1))
            end
    in fromx 0
    end
fun enumerate G =
    let fun next (0,a) = (a+1,0)
          | next (a,b) = (a - 1, b + 1)
        fun from a = CONS(point a G, fn () => from (next a))
    in from (0,0)
    end
```

第8章　参照型

8.1　参照型（eqtype τ ref）と逐次評価

問 8.1 x を int ref 型の式とする．以下の両式の違いは何か．

1. !x before x:= !x + 1
2. (x:= !x + 1; !x)

解答例　前者は、x を!x+1 で更新した後、参照しその値を返す式であるのに対して、後者は、更新した後、参照しその値を返す式である。

問 8.2 上記 3 つの構文は，他の構文の評価順序の約束を使って定義することができる．たとえば（exp_1; ⋯; exp_n）は #n（exp_1, ⋯, exp_n）の略記法とみなせる．exp_1 before exp_2 および while exp_1 do exp_2 の定義を与えよ．

解答例　例えば以下のような定義が可能である。

- exp_1 before exp_2：

  (fn x => fn () => x) exp_1 exp_2

- while exp_1 do exp_2：

```
let
   val E1 = fn () => exp1
   val E2 = fn () => exp2
   fun f () = if E1() then (E2(); f()) else ()
in
   f ()
end
```

問 8.3 以下の式の評価の結果を予測せよ．

1. (fn f => (print "a\n";f))
   (fn x => (print "b\n";x))
   (print "c\n";1);

2.
```
(fn f => (print "a\n";f))
((fn x => (print "b\n";x))
 (print "c\n";1));
```

3.
```
{S = "S" before print "S\n",
 M = "M" before print "M\n",
 L = "L" before print "L\n"};
```

解答例　「予測せよ」の解答例は困難であるが、... SML#による結果は以下の通り。

1.
```
# (fn f => (print "a\n";f))
> (fn x => (print "b\n";x))
> (print "c\n";1);
a
c
b
val it = 1 : int
```

2.
```
# (fn f => (print "a\n";f))
> ((fn x => (print "b\n";x))
>  (print "c\n";1));
c
b
a
val it = 1 : int
```

3.
```
# {S = "S" before print "S\n",
>  M = "M" before print "M\n",
>  L = "L" before print "L\n"};
S
M
L
val it = {L = "L", M = "M", S = "S"} : {L: string, M: string, S: string}
```

8.2　履歴に依存するプログラム

問 8.4 `toString` を書き，`gensym` を完成させよ.

解答例

```
fun toString L = implode (map chr (rev L))
```

問 8.5 上記のプログラムを一般化し，異なる文字のリストを受け取り，この文字を使って識別名を生成する関数を返す関数

```
makeGensym  : char list -> unit -> string
```

を定義せよ．ただし，文字列は，与えられたリストの順によって決まる文字の大小関係から導かれる順序とする．たとえば，`makeGensym [#"S",#"M",#"L"]` で生成される関数は，`"S"`, `"SS"`, `"SM"`, `"SL"`, `"MS"`, `"MM"`, `"ML"`, `"LS"`, ··· のような文字列の系列を生成するものとする．

解答例

```
fun makeGenSym L =
  let val seed = ref [0]
      fun next nil = [0]
        | next (h::t) = if h = (length L - 1) then 0::(next t) else (h+1)::t
      fun toString s = implode (map (fn x => List.nth (L,x)) (rev s))
  in
     fn () => toString (!seed) before seed := next (!seed)
  end
```

8.3 変更可能なデータ構造

問 8.6 `dataDlist`, `rightDlist`, `leftDlist` を定義せよ．

解答例

```
fun dataDlist (ref (CELL{data,...})) = data
fun rightDlist (ref (CELL{right,...})) = right
fun leftDlist (ref (CELL{left,...})) = left
```

問 8.7 `deleteDlist` と `fromListToDlist` を定義せよ．

解答例

```
fun deleteDlist dlist =
    case dlist of
      ref NIL => ()
    | ref (CELL{left=l1 as ref (CELL{right=r2,left=l2,...}),
                right=r1 as ref (CELL{right=r3,left=l3,...}),
                ...}) =>
        if l1 = l2 then dlist := NIL
        else (dlist := !r1; r2 := !r1; l3 := !l1)

fun fromListToDlist L = foldl (fn (x,y) => (insertDlist x y;y)) (ref NIL) (rev L)
```

補足: fromListToDlist は、リストの末尾から循環リストに追加していく関数である。foldr は末尾再帰の実装が困難なため、

```
fun fromListToDlist L = foldr (fn (x,y) => (insertDlist x y;y)) (ref NIL) L
```

よりスタックオーバフローなどの危険も少なく効率がよい。

この観点から、リストを生成する関数、例えば toList なども以下のような実装が望ましい。

```
fun toList L =
  let
    fun loop (l,A) visited =
      if member l visited then rev A
      else loop (rightDlist l, dataDlist l::A) (l::visited)
  in loop (rightDlist (leftDlist L), nil) nil
  end
```

さらに、循環リストは、リストの末尾からも辿ることができるので、以下のより効率よい自然なコードが可能である。

```
fun toList L =
  let
    fun loop (l,A) visited =
      if member l visited then A
      else loop (leftDlist l, dataDlist l::A) (l::visited)
  in loop (leftDlist L, nil) nil
  end
```

また、循環リストのリスト表現を逆順のリストとする定義すると、以下の関数となる。

```
fun toListL L =
  let
    fun loop (l,A) visited =
      if member l visited then A
      else loop (rightDlist l, dataDlist l::A) (l::visited)
  in loop (rightDlist (leftDlist L), nil) nil
  end
```

対応する循環リストへの変換は、

```
fun fromListToDlistL L = foldl (fn (x,y) => (insert x y;y)) (ref NIL) L
```

問 8.8 2 つの循環 2 重リストを連結する関数

```
concatDlist : 'a dlist -> 'a dlsit -> unit
```

を定義せよ．ただし，連結の結果は，どちらのリストから見ても，セルを右にたどると，以前のセルの後に他の循環 2 重リストのセルが付け加えられているようにせよ．

解答例 以下にコード例を示す。

```
fun concatDlist D1 D2 =
    case (D1, D2) of
      (ref NIL, _) => D1 := !D2
    | (_, ref NIL) => D2 := !D1
    | (ref (CELL{left=d1l as ref (CELL{right=d1lr,...}),...}),
       ref (CELL{left=d2l as ref (CELL{right=d2lr,...}),...})) =>
       let
         val d1lCell = !d1l
         val d1lrCell = !d1lr
       in
         (d1l := !d2l;
          d1lr := !d2lr;
          d2l := d1lCell;
          d2lr := d1lrCell)
       end
```

問 8.9 以下の関数を定義せよ．

1. 循環 2 重リンクリストをコピーする関数 copyDlist.
2. 与えられた関数を循環 2 重リンクリストの各要素に適用して得られる値を要素とする，新しい循環 2 重リンクリストを生成する関数 mapDlist.
3. リストの foldr および foldl に相当する関数 foldrDlist および foldlDlist.

解答例 これら関数の適切な定義には、以下の点を含む種々の注意深い考察が必要である。

- 循環構造を辿る際の終了条件の適切な判定
- left および right フィールドをたどると自分自身に戻ってくる CELL への参照型データの作成

以下に、これら 2 点を考慮したコード例を示す。なお、copyDlist は mapDlist とほぼ同様なコードであり、後者を使って定義している。

```
fun mapDlist f d =
    let
      fun newElem x nil = NONE
        | newElem x ((h,newH)::t) =
          if x = h then SOME newH
          else newElem x t
```

```
        fun copy l copied =
            case l of
              ref NIL => ref NIL
            | ref (CELL{left, right, data}) =>
              (case newElem l copied of
                 NONE =>
                 let
                   val newL = ref NIL
                   val copied = (l, newL)::copied
                   val l = copy left copied
                   val r = copy right copied
                 in
                   (newL := CELL{left = l, right = r, data = f data};
                    newL)
                 end
               | SOME newL => newL
              )
      in
        copy (rightDlist (leftDlist d)) nil
      end

  fun copyDlist d = mapDlist (fn x => x) d

  fun foldrDlist F z d =
      let
        fun member x nil = false
          | member x (h::t) = x = h orelse member x t
        fun f d z visited =
            if member d visited then z
            else F (dataDlist d, f (rightDlist d) z (d::visited))
      in f (rightDlist (leftDlist d)) z nil
      end

  fun foldlDlist F z d =
      let
        fun member x nil = false
          | member x (h::t) = x = h orelse member x t
        fun f d z visited =
            if member d visited then z
```

```
        else f (rightDlist d) (F (dataDlist d, z)) (d::visited)
in f (rightDlist (leftDlist d)) z nil
end
```

8.6 SML#のランク 1 多相性

問 8.10 本節の冒頭で与えた f と g の（ランク 1 多相性を持つ型システムの下での）型を予想し，SML#で確認せよ.

解答例 以下は、SML#での評価結果である。

```
# val f = ((fn x => x) 1, fn x => fn y => (x + 1,ref y));
val f = (1, fn) : int * (int -> ['a. 'a -> int * 'a ref])
# val g = (#2 f) 1;
val g = fn : ['a. 'a -> int * 'a ref]
```

第9章　例外処理

9.1　例外の定義と生成

問 9.1 関数 f の型 'a -> 'b は，任意の値 e に対して，関数適用 f e の結果が任意の型を持つことを意味する．値は何らかの型を持っているはずであるから，値を返す関数はこのような型を取り得ない．この f の型は，f が値を返さないという特殊な性質を反映している．この点を参考に，型 'a -> 'b を持つ関数を，例外を使わずに定義せよ．

解答例　典型的な例は、以下の無限ループ関数である。

```
fun f x = f x;
```

それ意外にも、組込み型を使えば、例えば、

```
fun f x = hd nil
```

など、種々可能である。

9.3　例外を使ったプログラミング

問 9.2 関数 enter での辞書の登録に際して，すでに同一名のキーが登録されていたら例外 DuplicateEntry を発生させるように変更せよ．

解答例

```
exception DuplicateEntry
fun enter (key,v,dict) =
    case dict of
         Empty => Node((key,v),Empty,Empty)
       | Node((key',v'),L,R) =>
           if key = key' then raise DuplicateEntry
           else if key > key' then
                Node((key',v'),L, enter (key,v,R))
           else Node((key',v'),enter (key,v,L),R)
```

問 9.3 int list 型データに含まれる値の積を求める関数を書け．ただしその中で，もしリストの途中に 0 が見つかったら処理を中断して，ただちに 0 を返すような処理を，例外を使って実現せよ．

解答例

```
fun prodList L =
  let
    exception Zero
    fun f nil r = r
      | f (0::t) r = raise Zero
      | f (h::t) r = f t (h * r)
  in
    f L 1
    handle Zero => 0
  end
```

問 9.4 makeMemoFun を使って作ったフィボナッチ関数と上記の fastFib を両方定義し，その速度を比較せよ.

解答例 SML#でコンパイルし時間計測をする手順も含め説明する。
makeMemoFun を使って作ったフィボナッチ関数：

1. 以下の内容の memoFib.sml を用意

```
fun fib 0 = 1
  | fib 1 = 1
  | fib n = fib (n - 1) + fib (n - 2)
fun makeMemoFun f =
    let exception NotThere
        val memo = ref (fn x => (raise NotThere))
    in fn x => !memo x
          handle NotThere =>
                let val v = f x
                    val oldMemo = !memo
                in (memo := (fn y => if x = y then v else oldMemo y);
                    v)
                end
    end
val memoFib = makeMemoFun fib
val _ = memoFib 43
```

2. 以下の内容の memoFib.smi を用意

```
_require "basis.smi"
```

3. コンパイルし実行。

```
$ smlsharp -o memoFib memoFib.sml
$ time memoFib
real    0m3.380s
user    0m3.380s
sys     0m0.000s
```

fasetFib 関数：

1. 以下の内容の fastFib.sml を用意

```
local
   exception NotThere
   fun f memo 0  = 1
     | f memo 1  = 1
     | f memo n =
         !memo n
           handle NotThere =>
                  let val v = f memo (n - 1) + f memo (n - 2)
                      val oldMemo = !memo
                      val _ = memo := (fn y => if n = y then v else oldMemo y)
                  in v
                  end
in val fastFib = f (ref  (fn x => raise NotThere))
end

val _ = fastFib 43
```

2. 以下の内容の fastFib.smi を用意

```
_require "basis.smi"
```

3. コンパイルし実行。

```
$ smlsharp -o fastFib fastFib.sml
$ time fastFib
real    0m0.002s
user    0m0.000s
sys     0m0.000s
```

以上の通り、fastFib の方が圧倒的に速いことが確認できる。

9.4　多相型を引数とする例外

問 9.5 例外が多相型をパラメータとして持ち得るとすると型システムでは検出できない型エラーを起こす例を，第 8.3 節の `polyIdRef` を参考に作成せよ．

解答例

```
exception foo of 'a
raise foo "a" handle foo x => x + 1
```

第10章　モジュールシステム

10.1　Structure文によるモジュールの定義と利用

問 10.1 第8.3節で定義した循環2重リストを使い，IntQueue と置き換え可能なストラクチャ ImperativeIntQueue を定義し，そのテストを行え．

解答例　この例を実装するために、まず、循環 2 重リストのコードを以下の DList ストラクチャにまとめる。

```
structure DList =
struct
  datatype 'a cell
    = NIL
    | CELL of {data:'a, left:'a cell ref, right:'a cell ref}
  exception EMPTY_DLIST
  type 'a dlist = 'a cell ref
  fun emptyDlist () = ref NIL
  fun rightDlist (ref (CELL{right,...})) = right | rightDlist _ = raise EMPTY_DLIST
  fun leftDlist (ref (CELL{left,...})) = left | leftDlist _ = raise EMPTY_DLIST
  fun dataDlist (ref (CELL{data,...})) = data | dataDlist _ = raise EMPTY_DLIST
  fun singleton a =
      let
        val l = ref NIL
        val r = ref NIL
        val c = CELL{left=l, right=r, data=a}
      in  (l:=c; r:=c; ref c)
      end
  fun member x nil = false
    | member x (h::t) = x = h orelse member x t
  fun insert a dlist =
      case dlist of
        ref (CELL{left=l1 as ref (CELL{right=r1,...}),...}) =>
        let val newcell = CELL{data=a,
                               right=ref (!dlist),
                               left=ref (!l1)}
        in (dlist:=newcell; l1:=newcell; r1:=newcell)
```

```
          end
        | ref NIL =>
          let
            val l = ref NIL
            val r = ref NIL
            val cell = CELL{data=a,left=l,right=r}
          in  (dlist:=cell; l:=cell; r:=cell)
          end
        | _ => raise EMPTY_DLIST
  fun deleteDlist dlist =
      case dlist of
        ref NIL => ()
      | ref (CELL{left=l1 as ref (CELL{right=r2,left=l2,...}),
                  right=r1 as ref (CELL{right=r3,left=l3,...}),
                  ...}) =>
          if l1 = l2 then dlist := NIL
          else (dlist := !r1; r2 := !r1; l3 := !l1)
    fun toList L =
        let fun f l visited =
                if member l visited then nil
                else (dataDlist l)::(f (rightDlist l) (l::visited))
        in f (rightDlist (leftDlist L)) nil
        end
    fun fromList (L:int list) = foldl (fn (x,y) => (insert x y;y)) (ref NIL) L
    fun concatDlist D1 D2 =
        case (D1, D2) of
          (ref NIL, _) => D1 := !D2
        | (_, ref NIL) => D2 := !D1
        | (ref (CELL{left=d1l as ref (CELL{right=d1lr,...}),...}),
           ref (CELL{left=d2l as ref (CELL{right=d2lr,...}),...})) =>
          let
            val d1lCell = !d1l
            val d1lrCell = !d1lr
          in
            (d1l := !d2l;
             d1lr := !d2lr;
             d2l := d1lCell;
             d2lr := d1lrCell)
          end
```

```
    | _ => raise EMPTY_DLIST
fun mapDlist f d =
    let
      fun newElem x nil = NONE
        | newElem x ((h,newH)::t) =
          if x = h then SOME newH
          else newElem x t
      fun copy l copied =
          case l of
            ref NIL => ref NIL
          | ref (CELL{left, right, data}) =>
            (case newElem l copied of
               NONE =>
               let
                 val newL = ref NIL
                 val copied = (l, newL)::copied
                 val l = copy left copied
                 val r = copy right copied
               in
                 (newL := CELL{left = l, right = r, data = f data};
                  newL)
               end
             | SOME newL => newL
            )
    in
      copy d nil
    end
fun copyDlist d = mapDlist (fn x => x) d
fun foldrDlist F z d =
    let
      fun f d z visited =
          if member d visited then z
          else F (dataDlist d, f (rightDlist d) z (d::visited))
    in f (rightDlist (leftDlist d)) z nil
    end
fun foldlDlist F z d =
    let
      fun f d z visited =
          if member d visited then z
```

```
            else f (rightDlist d) (F (dataDlist d, z)) (d::visited)
        in f (rightDlist (leftDlist d)) z nil
        end
  end
```

ImperativeIntQueue ストラクチャは、この DList ストラクチャを使って、以下のように定義される。

```
  structure ImperativeIntQueue =
  struct
    exception EmptyQueue
    type queue = int DList.dlist
    fun newQueue() = DList.emptyDlist() : queue
    fun enqueue (item,queue) = DList.insert item queue
    fun dequeue queue =
        let
          val last = DList.leftDlist queue
          val data = DList.dataDlist last
        in
          (DList.deleteDlist last; data)
        end
        handle DList.EMPTY_DLIST => raise EmptyQueue
  end
```

以下は、実行結果である。

```
  # val q = ImperativeIntQueue.newQueue();
  val q = ref NIL : int DList.cell ref
  # val _ = ImperativeIntQueue.enqueue(1,q);
  # val x = ImperativeIntQueue.dequeue q;
  val x = 1 : int
```

問 10.2 FastIntQueue が正しく待ち行列を実現していることを確認せよ．すなわち，空の待ち行列から始めて enqueue と dequeue 操作を繰り返したときの動作が，単純なリストによる場合と同一であることを示せ.

解答例　空の待ち行列から始めて enqueue(Q，e_i) を n 回、dequeue を m $(n > m)$ 実施した場合を考える。単純なリストの場合、リストの中身は、$[e_n, e_{n-1}, \ldots, e_{n-m}]$ $(n > m))$ であり、次の dequeue 操作で、e_{n-m} が返される。任意の $n, m(n > m)$ について、FastQueue の場合も、この状態と同等の状態が保たれることが示せればよい。FastIntQueue の操作途中の状態は、２つのリストの組 $(L1, L2)$ である。この２つのリストの中の要素を

$$\begin{aligned} L1 &= [e_1^1, \ldots, e_n^1] \\ L2 &= [e_1^2, \ldots, e_n^2] \end{aligned}$$

とする。すると、

$$[e_1^1, \ldots, e_n^1, e_n^2, \ldots, e_1^2] = [e_n, e_{n-1}, \ldots, e_{n-m}]$$

が成立し、次に返される値 e_1^2 は、e_{n-m} であり、リストを用いた待ち行列と同一である。この性質が、任意の $n, m(n > m)$ で成り立つから、FastQueue は単純なリストを用いた実装と同一の振る舞いをする。

問 10.3 enqueue と dequeue がランダムに行われる場合の，dequeue の平均の実行時間を見積もれ.

解答例 dequeue に掛かる実行時間を

1. リストに要素を 1 つ追加するための必要な時間 $CONS$
2. リストから先頭要素を 1 つ取り出すのに必要な時間 CAR

の回数で見積もる。L_2 が空でない合は $CAR = 1, CONS = 0$ である。L_2 が空の場合、L_1 の長さを N とすると、$CAR = N + 1, CONS = N$ に等しく、これら回数の平均を見積もることである。

この問題は、amortize cost（償却原価）の考え方を使えば、統計的・解析的な計算をせず、即座に求めることができる。dequeue で取り除かれる要素 e に着目する。この要素は、enqueue で $L1$ のリストの先頭に追加され、さらに、dequeue 操作でとりだされる前のいずれかの dequeue 操作で $L1$ のリストから取り除かれ、$L2$ のリストの先頭に追加される。このコストは $CAR = 1, CONS = 1$ である。そこで、この e が待ち行列に追加された時、この要素を dequeue するための将来必要なコスト原価として計上しておく、と考える。すると、dequeue 操作では、$L2$ が空の時に、$L1$ から $L2$ へ移動させるコストはすでにこの原価に含まれている、と考え、取り除かれる要素 e のコストは、e を取り除くための原価 $(CAR = 1, CONS = 1)$ +実際に掛かるコスト（$CAR = 1$）と計算できる。以上から、dequeue 操作の平均時間は $CAR = 2, CONS = 1$ である。

ちなみに、enqueue 操作の平均時間は、$CONS = 1$ であるから、FastQueue の要素あたりの enqueue、dequeue コストは $CONS = 2, CAR = 2$ であることがわかる。

問 10.4 IntQueue を FastIntQueue に置き換えてテストを行え.

解答例 教科書にあるコードの通り IntQueue が定義されている環境で、たとえば、以下のようなコードが可能である。

```
structure Q = IntQueue
val q = Q.newQueue()
val _ = Q.enqueue(1, q)
val _ = Q.enqueue(2, q)
val _ = Q.enqueue(3, q)
val a = Q.dequeue q
val b = Q.dequeue q
val c = Q.dequeue q
```

SML#の対話型環境では、以下のような実行結果を得る。

```
# structure Q = IntQueue;
structure Q =
  struct
    type queue = int list ref
```

```
    exception EmptyQueue = IntQueue.EmptyQueue
    val newQueue = fn : unit -> int list ref
    val enqueue = fn : ['a. 'a * 'a list ref -> unit]
    val removeLast = fn : ['a. 'a list -> 'a list * 'a]
    val dequeue = fn : ['a. 'a list ref -> 'a]
  end
# val q = Q.newQueue();
val q = ref [] : int list ref
# val _ = Q.enqueue(1, q);
# val _ = Q.enqueue(2, q);
# val _ = Q.enqueue(3, q);
# val a = Q.dequeue q;
val a = 1 : int
# val b = Q.dequeue q;
val b = 2 : int
# val c = Q.dequeue q;
val c = 3 : int
#
```

この Q ストラクチャの定義のみを、FastIntQueue に置き換えると以下のコードを得る。

```
structure Q = FastIntQueue
val q = Q.newQueue()
val _ = Q.enqueue(1, q)
val _ = Q.enqueue(2, q)
val _ = Q.enqueue(3, q)
val a = Q.dequeue q
val b = Q.dequeue q
val c = Q.dequeue q
```

SML#の対話型環境では、以下のような実行結果を得る。

```
# structure Q = FastQueue;
structure Q =
  struct
    type elem = int32
    type queue = elem list ref * elem list ref
    exception EmptyQueue = FastQueue.EmptyQueue
    val newQueue = fn : unit -> elem list ref * elem list ref
    val enqueue = fn : ['a, 'b. 'a * ('a list ref * 'b) -> unit]
    val dequeue = fn : ['a. 'a list ref * 'a list ref -> 'a]
  end
```

```
# val q = Q.newQueue();
val q = (ref [], ref []) : int list ref * int list ref
# val _ = Q.enqueue(1, q);
# val _ = Q.enqueue(2, q);
# val _ = Q.enqueue(3, q);
# val a = Q.dequeue q;
val a = 1 : int
# val b = Q.dequeue q;
val b = 2 : int
# val c = Q.dequeue q;
val c = 3 : int
```

この結果から、同一の動作をしており、置き換え可能であることが確認できる。

10.2 モジュールのシグネチャの指定

問 10.5 FastIntQueue に不透明な QUEUE シグネチャ制約を加えたストラクチャAbsFastIntQueue を定義し, AbsIntQueue と置き換え可能であることを確かめよ.

解答例 AbsIntQueue を使用する以下のようなコードを考える。

```
signature QUEUE = sig
  exception EmptyQueue
  type queue
  val newQueue : unit -> queue
  val enqueue : int*queue -> unit
  val dequeue : queue -> int
end
structure AbsIntQueue = IntQueue :> QUEUE;
structure Q = AbsIntQueue;
val q = Q.newQueue();
val _ = Q.enqueue(1, q);
val _ = Q.enqueue(2, q);
val _ = Q.enqueue(3, q);
val a = Q.dequeue q;
val b = Q.dequeue q;
val c = Q.dequeue q;
```

SML#での実行結果は、以下の通りである。

```
# structure AbsIntQueue =
  struct
```

```
      type queue  <hidden>
      exception EmptyQueue = IntQueue.EmptyQueue
      val newQueue = fn : unit -> queue
      val enqueue = fn : int * queue -> unit
      val dequeue = fn : queue -> int
    end
# structure Q =
    struct
      type queue  <hidden>
      exception EmptyQueue = IntQueue.EmptyQueue
      val newQueue = fn : unit -> queue
      val enqueue = fn : int * queue -> unit
      val dequeue = fn : queue -> int
    end
# val q = _ : AbsIntQueue.queue
# val a = 1 : int
# val b = 2 : int
# val c = 3 : int
```

AbsIntQueue を AbsFastIntQueue に置き換えたコードは以下の通りである。

```
structure AbsFastIntQueue = FastIntQueue :> QUEUE;
structure Q = AbsFastIntQueue;
val q = Q.newQueue();
val _ = Q.enqueue(1, q);
val _ = Q.enqueue(2, q);
val _ = Q.enqueue(3, q);
val a = Q.dequeue q;
val b = Q.dequeue q;
val c = Q.dequeue q;
```

SML#での実行結果は、以下の通りである。

```
# structure AbsFastIntQueue =
    struct
      type queue  <hidden>
      exception EmptyQueue = FastIntQueue.EmptyQueue
      val newQueue = fn : unit -> queue
      val enqueue = fn : int * queue -> unit
      val dequeue = fn : queue -> int
    end
# structure Q =
```

```
  struct
    type queue  <hidden>
    exception EmptyQueue = FastIntQueue.EmptyQueue
    val newQueue = fn : unit -> queue
    val enqueue = fn : int * queue -> unit
    val dequeue = fn : queue -> int
  end
# val q = _ : AbsFastIntQueue.queue
# val a = 1 : int
# val b = 2 : int
# val c = 3 : int
```

10.4 モジュールを使ったプログラミング例

問 10.6 以下の実行結果を予測せよ.

```
BF.bf (fromPreOrder "1(2(3()())(4()()))(5()())");
```

さらに，実際にテストを行い，結果を確認せよ.

解答例 幅優先探索であるから、ルートから同一レベルのルートが順に探索される。従って、1,2,5,3,4 の順に探索されるはずである。実際、SML#での実行結果は以下の通りである。

```
# BF.bf (fromPreOrder "1(2(3()())(4()()))(5()())");
val a = ["1", "2", "5", "3", "4"] : string list
```

問 10.7 `int tree` を要素とする待ち行列 `ITQueue` とそれを使って幅優先探索を行うストラクチャ `BFI` を定義せよ．さらに，第 7.1 節で定義した `fromPreOrder` 関数を改良し，`int tree` を返す関数 `fromIntPreOrder` を定義し，問 10.6 同様のテストを行え．`fromIntPreOrder` の定義には，基本ライブラリで提供されている，文字列を整数に変換する関数 `Int.fromString : string -> int option` を使用せよ.

解答例 `ITQueue`、`BFI`、`fromIntPreOrxder` の定義例を以下に示す。

```
structure ITQueue :> POLY_QUEUE where type elem = int tree =
struct
  exception EmptyQueue
  type elem = int tree
  type queue = elem list ref * elem list ref
  fun newQueue () = (ref [],ref []) : queue
  fun enqueue (i,(a,b))  = a := i :: (!a)
  fun dequeue (ref [],ref []) = raise EmptyQueue
    | dequeue (a as ref L, b as ref []) =
```

```
          let val (h::t) = rev L
          in (a:=nil; b:=t; h)
          end
      | dequeue (a,b as ref (h::t)) = (b := t; h)
end

structure BFI = struct
   structure Q = ITQueue
   fun bf t =
     let val queue = Q.newQueue()
         fun loop () =
             (case Q.dequeue queue of
                    Node(data,l,r) => (Q.enqueue (l,queue);
                                       Q.enqueue (r,queue);
                                       data::loop())
                  | Empty => loop())
             handle Q.EmptyQueue => nil
     in (Q.enqueue (t,queue); loop())
     end
end

fun fromIntPreOrder s =
    if s = "" then Empty
    else let val (root,left,right) = decompose s
             val SOME data = Int.fromString root
         in Node(data, fromIntPreOrder left, fromIntPreOrder right)
         end
```

SML#による BFI.bf のテスト結果は以下の通りである。

```
# val intList = BFI.bf (fromIntPreOrder "1(2(3()())(4()()))(5()())");
val intList = [1, 2, 5, 3, 4] : int list
```

問 10.8 string tree を要素とする待ち行列を，関数型待ち行列による方法，および，単純なリストによる実現法の 2 通りの方法で定義し，それぞれに対して BF の Q ストラクチャの定義を変更し，いずれの場合も BF が正しく動くことを確かめよ.

解答例　それぞれの BF の定義例を以下にしめす。以下では、単純なリストの場合のストラクチャ名を BFSimple としてある。

```
structure STQueue :> POLY_QUEUE where type elem = string tree = struct
  exception EmptyQueue
```

```
  type elem = string tree
  type queue = elem list ref * elem list ref
  fun newQueue () = (ref [],ref []) : queue
  fun enqueue (i,(a,b))  = a := i :: (!a)
  fun dequeue (ref [],ref []) = raise EmptyQueue
    | dequeue (a as ref L, b as ref []) =
        let val (h::t) = rev L
        in (a:=nil; b:=t; h)
        end
    | dequeue (a,b as ref (h::t)) = (b := t; h)
end

structure BF = struct
  structure Q = STQueue
  fun bf t =
    let val queue = Q.newQueue()
        fun loop () =
            (case Q.dequeue queue of
                  Node(data,l,r) => (Q.enqueue (l,queue);
                                     Q.enqueue (r,queue);
                                     data::loop())
                | Empty => loop())
            handle Q.EmptyQueue => nil
    in (Q.enqueue (t,queue); loop())
    end
end
structure STQueueSimple :> POLY_QUEUE where type elem = string tree = struct
  exception EmptyQueue
  type elem = string tree
  type queue = elem list ref
  fun newQueue() = ref nil : queue
  fun enqueue (item,queue) =
      queue := item :: (!queue)
  fun removeLast nil = raise EmptyQueue
    | removeLast [x] = (nil,x)
    | removeLast (h::t) =
      let val (t',last) = removeLast t
      in (h::t',last)
      end
```

```
    fun dequeue queue =
        let val (rest,last) = removeLast (!queue)
        in (queue:=rest; last)
        end
end

structure BFSimple = struct
   structure Q = STQueueSimple
   fun bf t =
     let val queue = Q.newQueue()
         fun loop () =
             (case Q.dequeue queue of
                    Node(data,l,r) => (Q.enqueue (l,queue);
                                       Q.enqueue (r,queue);
                                       data::loop())
                  | Empty => loop())
             handle Q.EmptyQueue => nil
     in (Q.enqueue (t,queue); loop())
     end
end
```

SML#によるテスト実行結果は以下の通りである。

```
# val a = BF.bf (fromPreOrder "1(2(3()())(4()()))(5()())");
val a = ["1", "2", "5", "3", "4"] : string list
# val b = BFSimple.bf (fromPreOrder "1(2(3()())(4()()))(5()())");
val b = ["1", "2", "5", "3", "4"] : string list
```

問 10.9 BF ストラクチャに，以下の引数を受け取り結果を返す汎用な関数 bffold の定義を追加せよ.

1. 木が空のとき返す値
2. 空でない木に対して，ルートノードのデータと，ルート以外のノードを幅優先でたどり処理した結果を受け取り，最終結果を返す関数.

BF ストラクチャの bf 関数をごくわずか変更するだけで，簡単に定義できるはずである.

bffold は，上記の関数 bf とリストの foldr 関数を組み合わせて得られる関数と同一の動きをするはずである．以下の 2 つの関数が同一の動作をすることをテストによって確かめよ.

- BF.bffold f z t
- foldr f z (BF.bf t)

解答例 bffold を含む BF の定義例を示す。bf と bffold はほぼ同一の構造をもち、かつ前者は後者を使って簡単に定義できるため、以下の例では、bf の定義を bffold を使って定義してある。

```
structure BF = struct
   structure Q = STQueue
   fun bffold f z t =
     let val queue = Q.newQueue()
         fun loop () =
             (case Q.dequeue queue of
                    Node(data,l,r) => (Q.enqueue (l,queue);
                                       Q.enqueue (r,queue);
                                       f (data, loop()))
                  | Empty => loop())
              handle Q.EmptyQueue => z
     in (Q.enqueue (t,queue); loop())
     end
   fun bf t = bffold (op ::) nil t
end
```

以下は、SML#によるテスト例である。

```
# val a = BF.bf (fromPreOrder "1(2(3()())(4()()))(5()())");
val a = ["1", "2", "5", "3", "4"] : string list
# val b = BF.bffold (op ::) nil (fromPreOrder "1(2(3()())(4()()))(5()())");
val b = ["1", "2", "5", "3", "4"] : string list
# val c = BF.bffold (op ^) "" (fromPreOrder "1(2(3()())(4()()))(5()())");
val c = "12534" : string
# val d = foldr (op ^) "" (BF.bf (fromPreOrder "1(2(3()())(4()()))(5()())"));
val d = "12534" : string
```

10.5 functor 文を使ったモジューラープログラミング

問 10.10 上記の QueueFUN を完成し，それを使って STQueue と ITQueue を定義せよ．
さらに，それらを定義の下で，BF ストラクチャおよび BFI ストラクチャを作成しなおし，

```
BF.bf (fromPreOrder "1(2(3()())(4()()))(5()())");
BFI.bf (fromIntPreOrder "1(2(3()())(4()()))(5()())");
```

などのテストプログラムを実行し，正しく動くことを確かめよ．

解答例 コード例を以下に示す。

```
functor QueueFUN(type ty) :> POLY_QUEUE where type elem = ty =
struct
  exception EmptyQueue
```

```
  type elem = ty
  type queue = elem list ref * elem list ref
  fun newQueue () = (ref [],ref []) : queue
  fun enqueue (i,(a,b))  = a := i :: (!a)
  fun dequeue (ref [],ref []) = raise EmptyQueue
    | dequeue (a as ref L, b as ref []) =
        let val (h::t) = rev L
        in (a:=nil; b:=t; h)
        end
    | dequeue (a,b as ref (h::t)) = (b := t; h)
end
structure ITQueue = QueueFUN(type ty = int tree)
structure STQueue = QueueFUN(type ty = string tree)
structure BF = struct
  structure Q = STQueue
  fun bf t =
    let val queue = Q.newQueue()
        fun loop () =
            (case Q.dequeue queue of
                   Node(data,r,l) => (Q.enqueue (r,queue);
                                      Q.enqueue (l,queue);
                                      data::loop())
                 | Empty => loop())
            handle Q.EmptyQueue => nil
    in (Q.enqueue (t,queue); loop())
    end
end
structure BFI = struct
  structure Q = ITQueue
  fun bf t =
    let val queue = Q.newQueue()
        fun loop () =
            (case Q.dequeue queue of
                   Node(data,l,r) => (Q.enqueue (l,queue);
                                      Q.enqueue (r,queue);
                                      data::loop())
                 | Empty => loop())
            handle Q.EmptyQueue => nil
    in (Q.enqueue (t,queue); loop())
```

```
    end
end
```

以下は、SML#によるテスト例である。

```
# val a = BF.bf (fromPreOrder "1(2(3()())(4()()))(5()())");
val a = ["1", "2", "5", "3", "4"] : string list
# val b = BFI.bf (fromIntPreOrder "1(2(3()())(4()()))(5()())");
val b = [1, 2, 5, 3, 4] : int list
```

問 10.11 種々の要素の型に対する待ち行列を作成するもう 1 つの方法は，待ち行列を 'a queue のような多相型にすることである．しかし多相型を持つ値に対する参照は許されない．そこで，参照型を使わずに，待ち行列のシグネチャを以下のように変更することにする．

```
signature FUNQUEUE = sig
   type 'a queue
   val newQueue : unit -> 'a queue
   val enqueue : 'a * 'a queue -> ' a queue
   val dequeue : 'a queue -> 'a * 'a queue
end
```

IntQueue および FastIntQueue をそれぞれこのシグネチャを持つように変更し，汎用の待ち行列ストラクチャを作成せよ．

解答例 以下にコード例を示す。このコードでは、FUNQUEUE シグネチャに EmptyQueue 例外を追加してある。また、汎用の待ち行列名をそれぞれ Queue、FastQueue とし不透明なシグネチャ制約を付してある。

```
signature FUNQUEUE = sig
  type 'a queue
  exception EmptyQueue
  val newQueue : unit -> 'a queue
  val enqueue : 'a * 'a queue -> 'a queue
  val dequeue : 'a queue -> 'a * 'a queue
end

structure Queue :> FUNQUEUE =
struct
  exception EmptyQueue
  type 'a queue = 'a list
  fun newQueue() = nil : 'a queue
  fun enqueue (item,queue) = item :: queue
  fun dequeue nil = raise EmptyQueue
    | dequeue [x] = (x, nil)
```

```
      | dequeue (h::t) =
        let val (last, t') =  dequeue t
        in (last, h::t')
        end
  end

  structure FastQueue :> FUNQUEUE = struct
    exception EmptyQueue
    type 'a queue = 'a list * 'a list
    fun newQueue () = ([],[]) : 'a queue
    fun enqueue (i,(a,b))  = (i :: a, b)
    fun dequeue ([],[]) = raise EmptyQueue
      | dequeue (L, []) =
          let
              val (h::t) = rev L
          in
             (h, ([], t))
          end
     | dequeue (a,h::t) = (h, (a,t))
  end
```

以下は、SML#によるテスト例である。

```
  # val q = Queue.newQueue() : int Queue.queue;
  val q = _ : int Queue.queue
  # val q = Queue.enqueue(1, q);
  val q = _ : int Queue.queue
  # val q = Queue.enqueue(2, q);
  val q = _ : int Queue.queue
  # val q = Queue.enqueue(3, q);
  val q = _ : int Queue.queue
  # val (a,q) = Queue.dequeue q;
  val a = 1 : int
  val q = _ : int Queue.queue
  # val (b,q) = Queue.dequeue q;
  val b = 2 : int
  val q = _ : int Queue.queue
  # val (c,q) = Queue.dequeue q;
  val c = 3 : int
  val q = _ : int Queue.queue
  # val q' = FastQueue.newQueue() : int FastQueue.queue;
```

```
val q' = _ : int FastQueue.queue
# val q' = FastQueue.enqueue(4, q');
val q' = _ : int FastQueue.queue
# val q' = FastQueue.enqueue(5, q');
val q' = _ : int FastQueue.queue
# val q' = FastQueue.enqueue(6, q');
val q' = _ : int FastQueue.queue
# val (d,q') = FastQueue.dequeue q';
val d = 4 : int
val q' = _ : int FastQueue.queue
# val (e,q') = FastQueue.dequeue q';
val e = 5 : int
val q' = _ : int FastQueue.queue
# val (f,q') = FastQueue.dequeue q';
val f = 6 : int
val q' = _ : int FastQueue.queue
```

問 10.12 入出力バッファを生成するファンクタの定義を，FUNQUEUE シグネチャを持つストラクチャを受け取り BUFFER シグネチャを持つストラクチャを生成するように変更せよ.

解答例 以下にコード例を示す。

```
functor BufferFUN(structure FQueue : FUNQUEUE )
        :> BUFFER =
struct
  exception EndOfBuffer
  type channel = char FQueue.queue ref
  fun openBuffer () = ref (FQueue.newQueue()) : channel
  fun input ch =
     let
       val (out, queue) = (FQueue.dequeue (!ch))
     in
       (ch := queue; out)
     end
     handle FQueue.EmptyQueue => raise EndOfBuffer
  fun output(ch, c) = ch := FQueue.enqueue (c, !ch)
end
```

第II部

Standard ML基本ライブラリ

第12章　基本ライブラリの利用法

12.1　ライブラリの種類とそのシグネチャの表示法

問 12.1 Math ストラクチャを利用して，対数関数の底を変換する関数

```
val convBase : (real -> real) -> real -> real -> real
```

を定義せよ．たとえば，

```
- val log2 = convBase Math.log10 2.0;
val log2 = fn : real -> real
- log2 1024.0;
10.0 : real
```

のように動作する．

解答例

```
fun convBase (f:real -> real) b x = f x / f b
```

問 12.2 よく使用するモジュールのシグネチャをプリントしておくと，モジュールの参照カードの役割を果たし，便利である．第 I 部で概説した bool，char，int，list，real，string の各データ型の機能は，それぞれ Bool，Char，Int，List，Real，String の各ストラクチャで実現されている．これらストラクチャのシグネチャをプリントし，参照カードを作成せよ．

解答例 例えば、SML#の対話型環境で、Bool および Char を structure Bool = Bool; structure Char = Char; のように再定義することによって、以下のような出力が得られる。

```
# structure Bool = Bool;
structure Bool =
  struct
    datatype bool = false | true
    val not = fn : bool -> bool
    val toString = fn : bool -> string
    val scan = fn : ['a. ('a -> (char * 'a) option) -> 'a -> (bool * 'a) option]
    val fromString = fn : string -> bool option
```

```
    end

  # structure Char = Char;
  structure Char =
    struct
      type char = char
      type string = string
      val minChar = #"\^@" : char
      val maxChar = #"\255" : char
      val maxOrd = 255 : int
      val ord = <builtin> : char -> int
      val chr = <builtin> : int -> char
      val succ = fn : char -> char
      val pred = fn : char -> char
      val compare = fn : char * char -> order
      val < = <builtin> : char * char -> bool
      val <= = <builtin> : char * char -> bool
      val > = <builtin> : char * char -> bool
      val >= = <builtin> : char * char -> bool
      val contains = fn : string -> char -> bool
      val notContains = fn : string -> char -> bool
      val isAscii = fn : char -> bool
      val toLower = fn : char -> char
      val toUpper = fn : char -> char
      val isAlpha = fn : char -> bool
      val isAlphaNum = fn : char -> bool
      val isCntrl = fn : char -> bool
      val isDigit = fn : char -> bool
      val isGraph = fn : char -> bool
      val isHexDigit = fn : char -> bool
      val isLower = fn : char -> bool
      val isPrint = fn : char -> bool
      val isSpace = fn : char -> bool
      val isPunct = fn : char -> bool
      val isUpper = fn : char -> bool
      val toString = fn : char -> string
      val toRawString = fn : char -> string
      val scan = fn : ['a. ('a -> (char * 'a) option) -> 'a -> (char * 'a) option]
      val fromString = fn : string -> char option
```

```
  val toCString = fn : char -> string
  val fromCString = fn : string -> char option
end
```

12.2 General ストラクチャ

問 12.3 以下の汎用エラー処理関数を定義せよ．関数とその引数，さらにエラーの場合に返す値を受け取り，関数を引数に適用するとともに，上記の基本演算のエラーをキャッチしエラーメッセージを表示した後，エラーの場合の値を返す，

```
catchAll : ('a -> 'b) -> 'a -> 'b -> 'b
```

たとえば，

```
- catchAll (fn (x,y) => x div y) (1,0) 999;
divide by zero
val it = 999 : int
```

のように動作する．

解答例 以下に SML#での定義と実行の例を示す。

```
# fun catchAll f x z =
      f x handle en => (print (exnMessage en ^ "\n"); z);
val catchAll = fn : ['a, 'b. ('a -> 'b) -> 'a -> 'b -> 'b]
# catchAll (op div) (1,0) 999;
Div at (interactive):9.12
val it = 999 : int
```

（注）General ストラクチャの 0 除算例外名は `divide by zero` ではなく `Div` である。

第13章　配列を用いたプログラミング

13.2　配列のソートアルゴリズム

問 13.1 sort 関数中にコメントで記述されている配列の 2 つの要素を入れ替える関数 swap を定義せよ.

解答例 問 13.3 の解答を参照。

問 13.2 以上の方針に従い，sort 関数中にコメントで記述されている pivot の設定処理を書け.

解答例 問 13.3 の解答を参照。

問 13.3 partition 関数の中にコメントで記述されている scanRight と scanLeft 関数を作成し，ArrayQuickSort ストラクチャを完成させよ.

解答例 ArrayQuickSort ストラクチャ全体のコード例を以下に示す。

```
signature SORT = sig
  val sort : 'a array * ('a * 'a -> order) -> unit
end
structure ArrayQuickSort : SORT  =
struct
local
  open Array
in
  fun sort (array,comp) =
    let
      fun swap (i,j) =
          let val temp = sub(array,i)
          in (update(array,i,sub(array,j)); update(array,j,temp))
          end
        fun getPivot (i,j) =
            let
              val delta = (j-i) div 4
              val i = i + delta
              val j = i + delta * 3
              val mid = i + (j-i) div 2
```

```
            val ie = sub(array,i)
            val je = sub(array,j)
            val me = sub(array,mid)
          in
            case (comp(ie,me),comp(me, je))
             of (LESS, LESS) => (mid,me)
              | (LESS, _) => (case comp(ie, je) of LESS => (j,je) | _ => (i,ie))
              | (_, GREATER) => (mid,me)
              | _ => (case comp(ie, je) of LESS => (i,ie) | _ => (j,je))
          end
      fun qsort (i,j) =
          if j <= i+1 then ()
          else
            let
              val pivot =
                let val (pi,pivot) = getPivot(i,j-1)
                in update(array,pi,sub(array,i));
                   update(array,i,pivot);
                    pivot
                end
              fun partition (a,b) =
                  if b < a then (a - 1)
                  else
                    let
                      fun scanRight a =
                          if a > b then a
                          else
                            case comp(sub(array,a),pivot) of
                              GREATER => a
                            | _ => scanRight (a+1)
                      val a = scanRight a
                      fun scanLeft b =
                          if b < a then b
                          else
                            case comp(sub(array,b),pivot) of
                              GREATER => scanLeft (b - 1)
                            | _ => b
                      val b = scanLeft b
                    in
```

```
                    if b < a then (a - 1)
                    else (swap(a,b);partition (a+1,b-1))
                  end
            val k = partition (i+1,j-1)
            val _ =  swap(i,k)
          in
            (qsort (i,k); qsort (k+1,j))
          end
    in
      qsort (0,Array.length array)
    end
end
end
```

問 13.4 以下の改良を加えよ.

1. 配列の大きさが小さい場合，クイックソートアルゴリズムは最適ではない．qsort 関数の冒頭で部分配列の大きさを判定し，大きさが 2 と 3 の場合を特別に処理するように変更せよ.
2. pivot 要素と先頭要素の入れ替え処理を伴う pivot の選択処理は，配列の大きさが小さいときかえって効率を低下させる．配列の大きさが 30 以下の場合，先頭要素を pivot とするように変更せよ.

解答例 以下に例を示す。この例では、pivot の選択も多少の洗練を試みている。ただし、効果があるかは計算実行環境にもより、一様に最適なものがあるかは、今後の探求によるところである。

```
structure ArrayQuickSortOpt : SORT  =
struct
local
  open Array
in
  fun sort (array,comp) =
      let
        fun swap (i,j) =
            let val temp = sub(array,i)
            in (update(array,i,sub(array,j)); update(array,j,temp))
            end
        fun sort3 i =
            case comp(sub(array,i),sub(array,i+1))
            of GREATER =>
              (case comp(sub(array,i+1),sub(array,i+2))
               of GREATER => (* 3 2 1 *)
                  swap(i,i+2)
```

```
            | _ => (case comp(sub(array,i),sub(array,i+2))
                    of GREATER => (* 3 1 2 *)
                       let val ei = sub(array,i) in
                         (update(array,i,sub(array,i+1));
                          update(array,i+1,sub(array,i+2));
                          update(array,i+2,ei))
                       end
                     | _ => (* 2 1 3 *)
                       (swap(i,i+1))
                  )
          )
          | _ =>
            (case comp(sub(array,i+1),sub(array,i+2))
              of GREATER =>
                 (case comp(sub(array,i),sub(array,i+2))
                   of GREATER => (* 2 3 1 *)
                      let val ei = sub(array,i) in
                        (update(array,i,sub(array,i+2));
                         update(array,i+2,sub(array,i+1));
                         update(array,i+1,ei))
                      end
                    | _ => (* 1 3 2 *)
                      (swap(i+1,i+2))
                 )
               | _ =>  (* 1 2 3 *)
                 ())
fun getPivot (i,j) =
    let
      val delta = (j-i) div 4
      val i = i + delta
      val j = i + delta * 3
      val mid = i + (j-i) div 2
      val ie = sub(array,i)
      val je = sub(array,j)
      val me = sub(array,mid)
    in
      case (comp(ie,me),comp(me, je))
       of (LESS, LESS) => (mid,me)
        | (LESS, _) => (case comp(ie, je) of LESS => (j,je) | _ => (i,ie))
```

```
            | (_, GREATER) => (mid,me)
            | _ => (case comp(ie, je) of LESS => (i,ie) | _ => (j,je))
        end
    fun qsort (i,j) =
        if j < i+2 then ()
        else if j = i+2 then case (comp(sub(array,i),sub(array,i+1)))
                             of GREATER => swap(i,i+1)
                              | _ => ()
        else if j = i + 3 then sort3  i
        else
          let
            val pivot = if (j-i) < 30 then sub(array,i)
                        else
                          let val (pi,pivot) = getPivot(i,j-1)
                          in update(array,pi,sub(array,i));
                             update(array,i,pivot);
                             pivot
                          end
            fun partition (a,b) =
                if b < a then a
                else
                  let
                    fun scanRight a =
                        if a > b then a
                        else
                          case comp(sub(array,a),pivot)
                          of GREATER => a
                           | _ => scanRight (a+1)
                    val a = scanRight a
                    fun scanLeft b =
                        if b < a then b
                        else
                          case comp(sub(array,b),pivot)
                          of GREATER => scanLeft (b - 1)
                           | _ => b
                    val b = scanLeft b
                  in
                    if b < a then a
                    else (swap(a,b);
```

```
                                partition (a+1,b-1))
                        end
                  val a = partition (i+1,j-1)
                  val _ =  swap(i,a-1)
                in
                  (qsort (i,a-1); qsort (a,j))
                end
        in
          qsort (0,Array.length array)
        end
  end
  end
```

第14章　システム時計の利用

14.1　Time と Timer ストラクチャ

問 14.1 日時の表現と操作は DATE シグネチャを持つ Date ストラクチャで提供されている．DATE シグネチャはその型から容易にその機能を推定できる．必要なら Date ストラクチャを使い，現在の日時と時刻を表す文字列を返す関数

```
currentTime : unit -> string
```

を定義せよ．

解答例 以下に定義と実行の例を示す。

```
# fun currentTime () = Date.toString (Date.fromTimeLocal (Time.now()));
val currentTime = fn : unit -> string
# currentTime ();
val it = "Sun Jul 19 10:17:42 2020" : string
```

問 14.2 関数 nlogn : int -> real と intcomp : int*int -> order を定義し，checkTime 関数を完成させよ．

解答例 以下に例を示す。以下では、intcomp は、Int ストラクチャに定義されている compare 関数を使用している。

```
fun log2 x = Math.log10 x / (Math.log10 2.0)
fun nlogn n = ((Real.fromInt n) * (log2 (Real.fromInt n)))
fun checkTime n =
    let
      val array = genArray n
      val tm = timeRun ArrayQuickSort.sort (array,Int.compare)
      val nlognRatio = tm / (nlogn n)
    in
      (n, tm div 1000, nlognRatio)
    end
```

問 14.3 関数 `printResult : int * int * real -> unit` は，n の値，時間，時間と $n/n\log(n)$ との比の値を受け取り，

```
size=10000, milli-secs=40, micro-secs/n log(n)=0.30103000
```

の形式でプリントする関数である．`printResult` 関数を定義し，`checkTime` 関数を完成させ，前節で作成したソートプログラムの評価を行え．

解答例 文脈から、完成させるべき関数は、`checkTime` ではなく、`testSort` 関数である。ここでは、SML#の動的型付け機構を使って定義されている以下の型の汎用の清書関数

```
# Dynamic.pp;
val it = fn : ['a#reify. 'a -> unit]
```

を使った定義例を以下に示す。

```
fun printResult (n, tm, ratio) =
    Dynamic.pp
     {"size" = n, "milli-secs" = tm, "micro-secs/n long(n)" = ratio}
fun testSort n = printResult (checkTime n)
```

以下は、SML#の対話型環境下での実行結果である。

```
# app testSort [1000, 10000, 1000000, 10000000];
{"micro-secs/n long(n)" = 0.0, "milli-secs" = 0, size = 1000}
{"micro-secs/n long(n)" = 0.0602059991328, "milli-secs" = 8, size = 10000}
{"micro-secs/n long(n)" = 0.0834856521308, "milli-secs" = 1664, size = 1000000}
{
  "micro-secs/n long(n)" = 0.0822069913873,
  "milli-secs" = 19116,
  size = 10000000
}
val it = () : unit
```

問 14.4 配列の大きさ n のリストを受け取り，各 n について n の値，計算時間，$n\log_2(n)$ の値との比を

```
- evalSort [10000,100000,1000000,10000000];
          array size          milli-sec.      micro s./n log(n)
               10000                  30          0.22577250
              100000                 460          0.27694760
             1000000                5740          0.28798536
            10000000               66630          0.28653755
 ------------------------------------------------------------
                                 avarage          0.26931075
```

のような形式で表示する関数 evalSort を以下の手順で作成せよ.

1. 20 文字以下の大きさの文字列を受け取り，その前に空白文字を付け加え，20 文字の大きさの文字列を返す関数

   ```
   padString : string -> string
   ```

   を定義せよ.

2. padString を使って，2 つの整数と 1 つの実数を受け取り，以下のような形式でプリントする関数 printLine を定義せよ.

   ```
   - printLine;
   val it = fn : int * int * real -> unit
   - printLine (10000,30,0.22577250);
                10000                    30           0.22577250
   val it = () : unit
   ```

3. 以上を使って evalSort を定義せよ.

解答例 evalSort の定義例を以下に示す。この例では、padString を定義する代わりに、StringCvt ストラクチャの padLeft 関数を使用している。

```
fun evalSort L =
    let
      val L' = map checkTime L
      val av = (foldr (fn ((_,_,x),y) => y+x) 0.0 L')/(Real.fromInt (List.length L'))
      val title = (StringCvt.padLeft #" " 20 "array size")
                  ^ (StringCvt.padLeft #" " 20 "milli-sec.")
                  ^ (StringCvt.padLeft #" " 20 "micro s./nlogn")
                  ^ "\n"
      fun formatReal a = StringCvt.padLeft #" " 20 (Real.fmt (StringCvt.FIX (SOME 8)) a)
      fun printLine (n,a,c) =
          let
            val ns =  StringCvt.padLeft #" " 20 (Int.toString n)
            val sa =  StringCvt.padLeft #" " 20 (Int.toString a)
            val sc = formatReal c
          in
            print (ns ^ sa ^ sc ^ "\n")
          end
    in
      (print title;
       map printLine L';
```

```
        print "-----------------------------------------------------------\n";
        print ("                                  avarage" ^ (formatReal av));
        print "\n")
    end
```

以下は、SML#の対話型環境下での実行結果である。現在の実行環境では、大きさ 10000 の配列では短時間に終了し、誤差が大きくなるとため、100000、1000000, 10000000、67108863（最大配列サイズ）のテスト行っている。

```
# evalSort [100000, 1000000, 10000000, Array.maxLen div 4];
          array size          milli-sec.      micro s./nlogn
              100000                 152          0.09151312
             1000000                1784          0.08950625
            10000000               20300          0.08729870
            67108863              144932          0.08306366
-------------------------------------------------------------
                                 avarage          0.08784543
val it = () : unit
```

問 14.5 関数 checkTime を一般化し，汎用のテスト関数

```
 eval : {prog:'a->'b, input:'a, size:'a-> int, base:int -> real}
        -> int * real * real
```

を定義せよ．ただし，引数レコードの各フィールドの意味は以下のとおりとする．

フィールド	値の意味
prog	評価すべきプログラム
input	プログラムへの入力
size	入力の大きさを返す関数
base	評価の基準となる関数

結果の型の意味は checkTime の場合と同様である．

解答例 checkTime と同様であれば、結果の型は int * int * real である。以下は定義例である。

```
fun eval {prog, input, size, base} =
    let
      val tm = timeRun prog input
      val n = size input
      val ratio = Real.fromInt tm / base n
    in
      (n, tm div 1000, ratio)
    end
```

問 14.6 自然数のリストを受け取り，各要素 n に対して，大きさ n の乱数配列を生成し，上記のような n 回の比較を行い，処理時間およびその n との比を

```
- evalCompare [500000,1000000,5000000];
          array size          milli-sec.          micro s./n
              500000                  40          0.08000000
             1000000                  70          0.07000000
             5000000                 370          0.07400000
------------------------------------------------------------
                                 avarage          0.07466667
```

のような形式で表示する関数 evalCompare を，eval を使って定義せよ．

解答例 以下に定義例を示す。

```
fun evalCompare L =
    let
      fun comp array =
          let
            val n = Array.length array
            val p = Array.sub(array, 0)
            val m = n div 2
            fun  loop x = if x <= m then ()
                          else (Int.compare (Array.sub(array, n - x), p);
                                Int.compare (Array.sub(array, x - 1), p);
                                loop (x -1))
          in
            loop n
          end
      fun evalN n = eval {prog = comp, input = genArray n, size = Array.length, base = real}
      val L' = map evalN L
      val av = (foldr (fn ((_,_,x),y) => y+x) 0.0 L')/(Real.fromInt (List.length L'))
      val title = (StringCvt.padLeft #" " 20 "array size")
                  ^ (StringCvt.padLeft #" " 20 "milli-sec.")
                  ^ (StringCvt.padLeft #" " 20 "micro s./n")
                  ^ "\n"
      fun formatReal a = StringCvt.padLeft #" " 20 (Real.fmt (StringCvt.FIX (SOME 8)) a)
      fun printLine (n,a,c) =
          let
            val ns =  StringCvt.padLeft #" " 20 (Int.toString n)
            val sa =  StringCvt.padLeft #" " 20 (Int.toString a)
            val sc = formatReal c
```

```
            in
              print (ns ^ sa ^ sc ^ "\n")
            end
      in
        (print title;
         map printLine L';
         print "-------------------------------------------------------------\n";
         print ("                                  avarage" ^ (formatReal av));
         print "\n")
      end
```

以下に SML#でのテスト例を示す。

```
# evalCompare [1000000, 10000000, Array.maxLen div 4];
          array size          milli-sec.          micro s./n
             1000000                   8          0.00800000
            10000000                  80          0.00800000
            67108863                 564          0.00840426
-------------------------------------------------------------
                               avarage          0.00813475
val it = () : unit
```

問 14.7 evalSort と evalCompare を組み合わせ，ソート関数の処理性能を

```
- normalEvalSort [500000,1000000,5000000,10000000];
          array size       time in cunit             T/n log(n)
              500000                2740          3.75926754
             1000000                5860          3.81825925
             5000000               33270          3.88323623
            10000000               68970          3.85195525
-------------------------------------------------------------
                                 avarage          3.82817957
The estimated sort time function: T(n) = 3.8 n log(n).
```

のような形式で表示をする関数 normalEvalSort を定義せよ.

解答例　以下に定義例を示す。

```
fun evalCompare L =
    let
      fun comp array =
          let
            val n = Array.length array
```

```
            val p = Array.sub(array, 0)
            val m = n div 2
            fun  loop x = if x <= m then ()
                          else (Int.compare (Array.sub(array, n - x), p);
                                Int.compare (Array.sub(array, x - 1), p);
                                loop (x -1))
          in
            loop n
          end
      fun evalN n =
          let val array = genArray n
          in eval {prog = comp, input = array, size = Array.length, base = real} end
      val L' = map evalN L
      val av = (foldr (fn ((_,_,x),y) => y+x) 0.0 L')/(Real.fromInt (List.length L'))
    in
      av
    end
  fun evalSortN c L =
    let
      fun sort array = ArrayQuickSort.sort (array, Int.compare)
      fun base n = c * nlogn n
      fun evalN n = eval {prog = sort, input = genArray n, size = Array.length, base = base}
      val L' = map evalN L
      val av = (foldr (fn ((_,_,x),y) => y+x) 0.0 L') / (Real.fromInt (List.length L'))
      val title = (StringCvt.padLeft #" " 20 "array size")
                  ^ (StringCvt.padLeft #" " 20 "time in cunit")
                ^ (StringCvt.padLeft #" " 20 "C/nlogn")
                ^ "\n"
      fun formatReal a = StringCvt.padLeft #" " 20 (Real.fmt (StringCvt.FIX (SOME 8)) a)
      fun printLine (n,a,c) =
       let
         val ns =  StringCvt.padLeft #" " 20 (Int.toString n)
         val sa =  StringCvt.padLeft #" " 20 (Int.toString (Real.floor (real a * 1000.0/ c)))
         val sc = formatReal c
       in
         print (ns ^ sa ^ sc ^ "\n")
       end
      val C = Real.fmt (StringCvt.FIX (SOME 2)) av
    in
```

```
        (print title;
         map printLine L';
         print "-----------------------------------------------------------------\n";
         print ("                                  avarage" ^ (formatReal av) ^ "\n");
         print ("The estimated sort time function: T(n) = " ^ C ^ " n log(n)\n"))
       end
  fun normalEvalSort L = evalSortN (evalCompare L) L
```

以下に SML#でのテスト例を示す。

```
  # normalEvalSort [1000000, 10000000, Array.maxLen div 4];
            array size       time in cunit             C/nlogn
               1000000              236268          7.43223684
              10000000             2756460          7.62717282
              67108863            20683151          7.75259017
  ------------------------------------------------------------
                                   avarage          7.60399994
  The estimated sort time function: T(n) = 7.60 n log(n)
  val it = () : unit
```

第15章　入出力処理

15.2　テキスト入出力

問 15.1 入力ファイル名のリストと出力ファイル名を受け取り，与えられたすべての入力ファイルの内容を連結した内容を出力ファイルに書き出す関数

```
cat : string list -> string -> unit.
```

を定義せよ.

解答例 以下に定義例を示す。

```
local
  open TextIO
in
  fun copyStream ins outs =
      if endOfStream ins then ()
      else case input1 ins of
             SOME c => (output1(outs,c);
                        copyStream ins outs)
           | NONE => copyStream ins outs
  fun cat L out =
      let
        val sources = map openIn L
        val sink = openOut out
      in
        (foldl (fn (x,_) => copyStream x sink) () sources;
         map closeIn sources;
         closeOut sink)
      end
end
```

問 15.2 指定されたファイルの文字数と行数をプリントする関数

```
wc : string -> unit
```

を定義せよ.

解答例　以下に定義例を示す。

```
local
  open TextIO
in
  fun wc file =
     let
       val ins = openIn file
       fun count (l,c) =
          if endOfStream ins then (l,c)
          else case input1 ins of
             SOME #"\n" => count (l+1, c+1)
           | SOME _ => count (l, c+1)
           | NONE => (l,c)
       val (l,c) = count (0,0)
       val _ = print (Int.toString l ^ " ")
       val _ = print (Int.toString c ^ "\n")
       val _ = closeIn ins
     in
       ()
     end
end
```

以下は SML#でのテスト実行例である

```
# wc "count.sml";
20 503
val it = () : unit
```

問 15.3 `filterFile` 関数を用いて，ファイルの中の文字をすべて小文字に変換したファイルを作成する関数

```
lowerFile : string -> string -> unit
```

を定義せよ.

解答例　以下に定義例を示す。

```
fun lowerFile inf outf = filterFile Char.toLower inf outf
```

問 15.4 stdIn と stdOut を使用し，プロンプト文字 ? を印字し，ユーザからの入力を受け取り，入力した文字をそのまま印字するプログラム

```
echo : unit -> unit
```

を書け．たとえば，以下のような動作をする．

```
- echo();
? abc
abc
? 1234;
1234;
? val it = () : unit
```

この例では，最後のプロンプトの後，ファイル終了文字を入力している．

解答例

```
fun echo () =
  let
    fun loop () =
        (print "? ";
         if TextIO.endOfStream TextIO.stdIn then ()
         else
           case TextIO.inputLine TextIO.stdIn of
             SOME string => (print string; loop ())
           | NONE => loop()
         )
  in
    loop()
  end
```

15.4 簡単な字句解析処理

問 15.5 getID を参考にして，数字列を読む込み関数

```
getNum : TextIO.instream -> string
```

を定義せよ．ただし数字か否かの判定には，ライブラリ関数 Char.isDigit を使用せよ．

解答例

```
fun getNum ins =
  let
    fun getRest s =
        case T.lookahead ins of
          NONE => s
        | SOME c =>
          if Char.isDigit c then
            getRest (s ^ T.inputN(ins,1))
          else s
  in
    DIGITS (getRest "")
  end
```

問 15.6 testLex の中で使われている toString は token 型データを文字列に変換する関数である．toString を定義せよ．

解答例

```
fun toString tok =
    case tok of
        EOF => "EOF"
      | ID s => "ID(" ^ s ^ ")"
      | DIGITS s => "DIGITS(" ^ s ^ ")"
      | SPECIAL c => "SPECIAL" ^ Char.toString c ^ ")"
      | BANG => "BANG"
      | DOUBLEQUOTE => "DOUBLEQUOTE"
      | HASH => "HASH"
      | DOLLAR => "DOLLAR"
      | PERCENT => "PERCENT"
      | AMPERSAND => "AMPERSAND"
      | QUOTE => "QUOTE"
      | LPAREN => "LPAREN"
      | RPAREN => "RPAREN"
      | TILDE => "TILDE"
      | EQUALSYM => "EQUALSYM"
      | HYPHEN => "HYPHEN"
      | HAT => "HAT"
      | UNDERBAR => "UNDERBAR"
```

```
    | SLASH => "SLASH"
    | BAR => "BAR"
    | AT => "AT"
    | BACKQUOTE => "BACKQUOTE"
    | LBRACKET => "LBRACKET"
    | LBRACE => "LBRACE"
    | SEMICOLON => "SEMICOLON"
    | PLUS => "PLUS"
    | COLON => "COLON"
    | ASTERISK => "ASTERISK"
    | RBRACKET => "RBRACKET"
    | RBRACE => "RBRACE"
    | COMMA => "COMMA"
    | LANGLE => "LANGLE"
    | PERIOD => "PERIOD"
    | RANGLE => "RANGLE"
    | BACKSLASH => "BACKSLASH"
    | QUESTION => "QUESTION"
```

問 15.7 lex 関数の特殊文字の処理を補い，lex を完成せよ.

解答例

```
signature LEX = sig
  datatype token
    =  EOF                          | ID of string
    | DIGITS of string              | SPECIAL of char
    | BANG              (* ! *)     | DOUBLEQUOTE     (* " *)
    | HASH              (* # *)     | DOLLAR          (* $ *)
    | PERCENT           (* % *)     | AMPERSAND       (* & *)
    | QUOTE             (* ' *)     | LPAREN          (* ( *)
    | RPAREN            (* ) *)     | TILDE           (* ~ *)
    | EQUALSYM          (* = *)     | HYPHEN          (* - *)
    | HAT               (* ^ *)     | UNDERBAR       (* _ *)
    | SLASH             (* \ *)     | BAR             (* | *)
    | AT                (* @ *)     | BACKQUOTE      (* ` *)
    | LBRACKET          (* [ *)     | LBRACE          (* { *)
    | SEMICOLON         (* ; *)     | PLUS            (* + *)
    | COLON             (* : *)     | ASTERISK        (* * *)
    | RBRACKET          (* ] *)     | RBRACE          (* } *)
    | COMMA             (* , *)     | LANGLE          (* < *)
```

```
    | PERIOD          (* . *)      | RANGLE          (* > *)
    | BACKSLASH       (* / *)      | QUESTION        (* ? *)
  val lex : TextIO.instream -> token
  val toString : token -> string
  val testLex : unit -> unit
end

structure Lex : LEX =
struct
  structure T = TextIO
      datatype token
        = EOF                          | ID of string
        | DIGITS of string             | SPECIAL of char
        | BANG            (* ! *)     | DOUBLEQUOTE     (* " *)
        | HASH            (* # *)     | DOLLAR          (* $ *)
        | PERCENT         (* % *)     | AMPERSAND       (* & *)
        | QUOTE           (* ' *)     | LPAREN          (* ( *)
        | RPAREN          (* ) *)     | TILDE           (* ~ *)
        | EQUALSYM        (* = *)     | HYPHEN          (* - *)
        | HAT             (* ^ *)     | UNDERBAR        (* _ *)
        | SLASH           (* \ *)     | BAR             (* | *)
        | AT              (* @ *)     | BACKQUOTE       (* ` *)
        | LBRACKET        (* [ *)     | LBRACE          (* { *)
        | SEMICOLON       (* ; *)     | PLUS            (* + *)
        | COLON           (* : *)     | ASTERISK        (* * *)
        | RBRACKET        (* ] *)     | RBRACE          (* } *)
        | COMMA           (* , *)     | LANGLE          (* < *)
        | PERIOD          (* . *)     | RANGLE          (* > *)
        | BACKSLASH       (* / *)     | QUESTION        (* ? *)
  fun skipSpaces ins =
      case T.lookahead ins of
        SOME c => if Char.isSpace c
                  then (T.input1 ins;skipSpaces ins)
                  else ()
      | _ => ()
  fun getID ins =
      let fun getRest s =
          case T.lookahead ins of
            SOME c => if Char.isAlphaNum c then
```

```
                        getRest (s ^ T.inputN(ins,1))
                    else s
          | _ => s
      in ID(getRest "")
      end
  fun getNum ins =
    let
      fun getRest s =
          case T.lookahead ins of
            NONE => s
          | SOME c =>
            if Char.isDigit c then
              getRest (s ^ T.inputN(ins,1))
            else s
    in
      DIGITS (getRest "")
    end
  fun lex ins =
      (skipSpaces ins;
       if T.endOfStream ins then EOF
       else
        let
          val c = valOf (T.lookahead ins)
        in
          if Char.isDigit c then getNum ins
          else if Char.isAlpha c then getID ins
          else case valOf (T.input1 ins) of
            #"!" => BANG
          | #"\"" => DOUBLEQUOTE
          | #"#" => HASH
          | #"$" => DOLLAR
          | #"%" => PERCENT
          | #"&" => AMPERSAND
          | #"'" => QUOTE
          | #"(" => LPAREN
          | #")" => RPAREN
          | #"~" => TILDE
          | #"=" => EQUALSYM
          | #"-" => HYPHEN
```

```
            | #"^" => HAT
            | #"_" => UNDERBAR
            | #"\\" => SLASH
            | #"|" => BAR
            | #"@" => AT
            | #"`" =>  BACKQUOTE
            | #"[" => LBRACKET
            | #"{" => LBRACE
            | #";" => SEMICOLON
            | #"+" => PLUS
            | #":" => COLON
            | #"*" => ASTERISK
            | #"]" => RBRACKET
            | #"}" => RBRACE
            | #"," => COMMA
            | #"<" => LANGLE
            | #"." => PERIOD
            | #">" => RANGLE
            | #"/" => BACKSLASH
            | #"?" => QUESTION
            | _ => SPECIAL c
          end)

    fun toString tok =
        case tok of
        EOF => "EOF"
          | ID s => "ID(" ^ s ^ ")"
          | DIGITS s => "DIGITS(" ^ s ^ ")"
          | SPECIAL c => "SPECIAL" ^ Char.toString c ^ ")"
          | BANG => "BANG"
          | DOUBLEQUOTE => "DOUBLEQUOTE"
          | HASH => "HASH"
          | DOLLAR => "DOLLAR"
          | PERCENT => "PERCENT"
          | AMPERSAND => "AMPERSAND"
          | QUOTE => "QUOTE"
          | LPAREN => "LPAREN"
          | RPAREN => "RPAREN"
          | TILDE => "TILDE"
```

```
          | EQUALSYM => "EQUALSYM"
          | HYPHEN => "HYPHEN"
          | HAT => "HAT"
          | UNDERBAR => "UNDERBAR"
          | SLASH => "SLASH"
          | BAR => "BAR"
          | AT => "AT"
          | BACKQUOTE => "BACKQUOTE"
          | LBRACKET => "LBRACKET"
          | LBRACE => "LBRACE"
          | SEMICOLON => "SEMICOLON"
          | PLUS => "PLUS"
          | COLON => "COLON"
          | ASTERISK => "ASTERISK"
          | RBRACKET => "RBRACKET"
          | RBRACE => "RBRACE"
          | COMMA => "COMMA"
          | LANGLE => "LANGLE"
          | PERIOD => "PERIOD"
          | RANGLE => "RANGLE"
          | BACKSLASH => "BACKSLASH"
          | QUESTION => "QUESTION"
    fun testLex () =
        let
          val token = lex (TextIO.stdIn)
        in
          case token of
            EOF => ()
          | _ => (print (toString token ^ "\n");
                  testLex ())
        end
end
```

問 15.8 字句解析プログラムに，以下のコマンドを解釈しファイルから読み込む処理を加えよ．

```
use fileName
```

字句解析処理は，この行を入力すると，*fileName* で指定されたファイルをオープンし，ファイルから入力を続行する．たとえば temp.txt ファイルの中身が `InFile` であれば，以下のような動作をする．

```
1
DIGITS(1)
use temp.txt
[opening file "temp.txt"]
ID(InFile)
[closing file "temp.txt"]
2
DIGITS(1)
```

さらにこの use 構文はネストして使用してもよいものとする.

解答例　この対応には、lex 関数ではなく、lex を使用するメイン関数、すなわち testLex の変更が必要である。以下に変更例を示す。

```
fun getFileName ins =
    let fun getRest s =
            case (T.lookahead ins) of
              SOME c =>
              if Char.isSpace c then s
              else getRest (s ^ T.inputN(ins,1))
            | NONE => s
    in getRest "" end
 fun testMain ins =
     let
       val token = lex ins
     in
       case token of
          EOF => ()
        | ID "use" =>
            let
              val fileName = (skipSpaces ins; getFileName ins)
              val newIns = TextIO.openIn fileName
            in
              (testMain newIns; TextIO.closeIn newIns; testMain ins)
            end
         | _ => (print (toString token ^ "\n");
                 testMain ins)
     end
 fun testLex () = testMain TextIO.stdIn
```

15.5 入出力エラーの処理

問 15.9 以前作成したファイル変換関数に上記の入出力エラー処理を加え,

```
- lowerFile "foo.txt";
IO Error : openIn failed. No such file or directory : foo.txt
val it = () : unit
```

のようにエラーメッセージを表示するように改良せよ.

解答例 教科書本文にある通り、例外 `IO.Io name:string, function:string, cause:exn` を以下のように `handle` すればよい。このエラー処理は `lowerFile` 関数に以下のように追加できる。

```
fun lowerFile inf outf =
    filterFile Char.toLower inf outf
    handle IO.Io {name,function,cause} =>
           (print ("IO Error : " ^ function ^ " faild. ");
            case cause of
               OS.SysErr (s,e) => print (s ^ ": ")
             | _ => print (exnMessage cause ^ ": ");
            print  (name ^ "\n"))
```

しかし、エラーを処理しプログラムを継続する場合には、オープンしているファイルのクローズ処理をするのが望ましい。そこで、ファイルをオープンする関数 `lowerFile` でエラー処理も行うのがより適当である。ただ、`lowerFile` は 2 つのファイルをオープンしており、クローズ処理が必要なオープン済みストリームはプログラムコードの位置に依存する。この問題を系統的に処理する一つの方法は、ファイナライザ (finalizer) 関数をエラー処理に渡す手法がある。以下はその例である。

```
exception Abort
fun IOhandler ({name,function,cause}, finalizer)  =
    (print ("IO Error : " ^ function ^ " faild. ");
     case cause of
       OS.SysErr (s,e) => print (s ^ ": ")
     | _ => print (exnMessage cause ^ ": ");
     print  (name ^ "\n");
     finalizer();
     raise Abort)
fun filterFile f inf outf =
      let val ins = openIn inf
              handle IO.Io param =>
                     IOhandler (param,fn () => ())
```

```
            val outs = openOut outf
                handle IO.Io param =>
                       IOhandler (param,fn () => closeIn ins)
        in
          (filterStream f ins outs
           handle IO.Io param =>
                  IOhandler
                    (param,
                    fn () => (TextIO.closeIn ins;
                              TextIO.closeOut outs));
            closeIn ins;
            closeOut outs)
        end
        handle Abort => ()
  fun lowerFile inf outf = filterFile Char.toLower inf outf
```

上記何れの場合も、SML#で以下のような実行結果となる。

```
# lowerFile "foo.txt";
val it = fn : string -> unit
# lowerFile "foo.txt" "bar.txt";
IO Error : openIn faild. No such file or directory: foo.txt
val it = () : unit
```

なお、例から分かる通り、`lowerFile` は２つの引数を取る関数であり、`lowerFile "foo.txt"`のみでは、実行されないため、エラーは発生しない。

第16章　データのフォーマッティング

16.2　文字列からのデータの読み込み

問 16.1 getc が（char，substring）reader 型を持つことを確かめよ．

解答例 Substring.getc の型は substring -> (char * substring) option である。一方 ('a, 'b) reader は 'b -> ('a * 'b) option であるから、substring -> (char * substring) option は、(char, substring) reader である。

問 16.2 上記の関数を使用して，空白で区切られた数字データの文字列を受け取り，呼び出されるごとに，そのデータから数字を 1 つずつ読み出す関数を返す関数

```
readInt : string -> unit -> int option
```

を定義せよ．たとえば

```
val f = readInt "123 345 abc";
val f = fn : unit -> int option
- f ();
val it = SOME 123 : int option
- f();
val it = SOME 345 : int option
- f();
val it = NONE : int option
```

のような振る舞いをする．

解答例　教科書にある decScan と intScan の定義も含めたコード例を以下に示す。

```
fun decScan x = Int.scan StringCvt.DEC x;
val intScan = decScan Substring.getc
fun readInt string =
  let
```

```
    val stream = ref (Substring.full string)
  in
    fn () =>
      case intScan (!stream) of
        SOME (i, substring) =>
        SOME i before stream := substring
      | NONE => NONE
end
```

問 16.3 real 型および bool 型に対してもそれぞれ以下の型を持つ関数が定義されている.

```
Real.scan : (char,'a) reader -> (real,'a) reader
Bool.scan : (char,'a) reader -> (bool,'a) reader
```

これらを用いて，ReadInt と同様の読み込み関数 readReal および readBool を定義せよ.

解答例　コード例を以下にしめす。これら関数はすべて同型であるため、以下の例では、高階の関数を定義し、それを各型の scan 関数に適用している。

```
fun makeRead scan string =
  let
    val reader = scan Substring.getc
    val stream = ref (Substring.full string)
  in
    fn () =>
      case reader (!stream) of
        SOME (i, substring) =>
        SOME i before stream := substring
      | NONE => NONE
 end
val readReal = makeRead Real.scan
val readBool = makeRead Bool.scan
```

以下はテスト実行例である。

```
# val f = readReal "1.2 3E10 abc";
val f = fn : unit -> real option
# f ();
val it = SOME 1.2 : real option
# f ();
val it = SOME 30000000000.0 : real option
# f ();
val it = NONE : real option
```

```
# val f = readBool "true false abc";
val f = fn : unit -> bool option
# f ();
val it = SOME true : bool option
# f ();
val it = SOME false : bool option
# f ();
val it = NONE : bool option
```

問 16.4 上記の各種の scan 関数は，対応するリーダに適用すれば substring 型以外のストリームデータに対してもそのまま使用することができる．これら種々のストリームに使用することができる以下の型を持つ汎用の読み込み関数

```
genericReadInt : (char,'a) reader -> 'a -> unit -> int option
```

を書け.

解答例

```
fun genericReadInt baseReader baseData =
  let
    val stream = ref baseData
    val reader = Int.scan StringCvt.DEC baseReader
  in
    fn () =>
       case reader (!stream) of
         SOME (i, data) =>
         SOME i before stream := data
       | NONE => NONE
  end
```

例えば、以下のように使用できる。

```
# fun readIntFromString string = genericReadInt Substring.getc (Substring.full string);
val readIntFromString = fn : string -> unit -> int option
# val f = readIntFromString  "123 456 abc";
val f = fn : unit -> int option
# f ();
val it = SOME 123 : int option
# f ();
val it = SOME 456 : int option
# f ();
val it = NONE : int option
```

問 16.5 ファイルの入力ストリームを受け取り，入力ストリームから数字データを 1 つずつ読み出す関数を返す関数

```
readIntFromStream : TextIO.instream -> unit -> int option
```

を genericReadInt を使って定義せよ．genericReadInt を使い，空白で区切られた数字列を内容とするファイル名を受け取り，ファイル内の各数字列を int 型データに変換し，そのリストを返す関数

```
readIntFromFile : string -> int list
```

を書け.

解答例 TextIO.StreamIO.input1 が、(char, TextIO.StreamIO.instream) reader を持つので、この関数を使って、以下のように定義できる。

```
fun readIntFromStream ins =
  let
    val stream = TextIO.getInstream ins
  in
    genericReadInt TextIO.StreamIO.input1 stream
  end
```

readIntFromFile の定義例を以下に示す。

```
fun readIntFromFile file =
  let
    val ins = TextIO.openIn file
    val reader = readIntFromStream ins
    fun loop L =
        case  reader () of
          SOME i => loop (L @ [i])
        | NONE =>  L
  in
    loop nil before TextIO.closeIn ins
  end
```

問 16.6 parseHttp を参考にして，ローカルファイルシステムのファイルパスを表す URL の解析処理関数 parseFile，およびスキーマ指定のない相対アドレス解析処理関数 parseRelative を書き ParseUrl モジュールを完成せよ.

解答例

```
structure ParseURL =
struct
local
  structure SS = Substring
```

```
in
  exception urlFormat
  datatype url
    = HTTP of {host : string list, path : string list option,
               anchor : string option}
    | FILE of {path : string list, anchor : string option}
    | RELATIVE of {path : string list, anchor : string option}
  fun parseHttp s =
      let val s = if SS.isPrefix "://" s then
                    SS.triml 3 s
                  else raise urlFormat
          fun neq c x = not (x = c)
          fun eq c x = c = x
          val (host,body) = SS.splitl (neq #"/") s
          val domain = map SS.string (SS.tokens (eq #".") host)
          val (path,anchor) =
              if SS.isEmpty body then (NONE,NONE)
              else
                let val (p,a) = SS.splitl (neq #"#") body
                in (SOME (map SS.string (SS.tokens (eq #"/") p)),
                    if SS.isEmpty a then NONE
                    else SOME (SS.string (SS.triml 1 a)))
                end
      in {host=domain, path=path, anchor=anchor}
      end
  fun parseRelative s =
      let val (path,anchor) =
              let val (p,a) = SS.splitl (fn #"#" => false | _ => true) s
              in (map SS.string (SS.fields (fn c => c = #"/") p),
                  if SS.isEmpty a then NONE else SOME (SS.string (SS.triml 1 a)))
           end
      in {path=path, anchor=anchor}
      end
  fun parseFile s =
      let val s = if SS.isPrefix ":/" s then SS.triml 2 s else raise urlFormat
          val (path,anchor) =
              let val (p,a) = SS.splitl (fn #"#" => false | _ => true) s
              in (map SS.string (SS.tokens (fn c => c = #"/") p),
                  if SS.isEmpty a then NONE else SOME (SS.string (SS.triml 1 a)))
```

```
                end
        in  {path=path,anchor=anchor}
        end
    fun parseUrl s =
        let val s = SS.full s
            val (scheme,body) = SS.splitl (fn c => not (c = #":")) s
        in
          if SS.isEmpty body then
             RELATIVE (parseRelative scheme)
          else case SS.string scheme of
                 "http" => HTTP (parseHttp body)
               | "file" => FILE (parseFile body)
               | _  => raise urlFormat
        end
end
end
```

16.3　書式付き書き出し処理

問 16.7 上で定義した formatData 関数を使って，string * string * string 型データと int * int * int list 型のデータを受け取り，

```
- printData ("first", "second", "third") [(1,2,3),(4,5,6)];
     first    second     third
         1         2         3
         4         5         6
```

のような形式でプリントする関数 printTriple を書け.

解答例

```
fun printTriple (s1, s2, s3) L =
    let
      val width = 10
      fun prS s =
          print
           (formatData
            {kind=STRING, width=SOME width, align=RIGHT} (S s))
      fun prI i =
          print
           (formatData
```

```
            {kind=INT StringCvt.DEC, width=SOME width, align=RIGHT} (I i))
      fun printLine (i1,i2,i3) = (prI i1; prI i2; prI i3; print "\n")
    in
      prS s1;
      prS s2;
      prS s3;
      print "\n";
      map printLine L
    end
```

問 16.8 第 16.2 節で定義した intScan を利用し，与えられた部分文字列の先頭が整数 n の表現であれば SOME n と残りの部分文字列を返し，数字表現でなければ NONE と与えられた部分文字列そのものを返す関数

```
scanInt : substring -> int option * substring
```

を書け.

解答例

```
fun scanInt s =
    case intScan s of
      SOME (i,s) => (SOME i, s)
    | NONE => (NONE, s)
```

問 16.9 以上の関数定義をまとめて，以下のシグネチャを持つストラクチャFormat を構築せよ.

```
signature FORMAT =
sig datatype kind =  INT of StringCvt.radix
                   | REAL of StringCvt.realfmt
                   | STRING
                   | BOOL
    datatype align = LEFT | RIGHT
    datatype format =
             LITERAL of string
           | SPEC of {kind:kind,width:int option,align:align}
    datatype argument = I of int
                      | R of real
                      | S of string
                      | B of bool
    exception formatError
    val format : string -> argument list -> string
    val printf : string -> argument list -> unit
end
```

ただし，printf 関数は，format 関数を使ってフォーマットした文字列を標準出力に印字する関数である．

解答例

```
structure Format : FORMAT =
struct
  exception formatError
  structure S = Substring
  datatype kind =  INT of StringCvt.radix
                 | REAL of StringCvt.realfmt
                 | STRING
                 | BOOL
  datatype align = LEFT | RIGHT
  datatype format =
           LITERAL of string
         | SPEC of {kind:kind,width:int option,align:align}
  datatype argument = I of int
                    | R of real
                    | S of string
                    | B of bool
  fun formatData {kind,width,align} data=
      let val body =
              case (kind,data) of
                (INT radix,I i) => Int.fmt radix i
              | (REAL fmt,R r) => Real.fmt fmt r
              | (STRING,S s) => s
              | (BOOL,B b) => Bool.toString b
              | _ => raise formatError
      in case width of
           NONE => body
         | SOME w => (case align of
                        LEFT => StringCvt.padRight #" " w body
                      | RIGHT => StringCvt.padLeft #" " w body)
      end
  fun scanInt s =
      let val r= Int.scan StringCvt.DEC S.getc s
      in case r of NONE => (NONE,s)
                 | SOME(n,s) => (SOME n,s)
      end
  fun oneFormat s =
      let val s = S.triml 1 s
```

```
    in if S.isPrefix "%" s then (LITERAL "%",S.triml 1 s)
       else
         let val (a,s) = if S.isPrefix "-" s
                         then (LEFT,S.triml 1 s)
                         else (RIGHT,s)
             val (w,s) = scanInt s
             val (c,s) = case S.getc s of NONE => raise formatError
                                        | SOME s  => s
         in (SPEC {width=w,align=a,
                   kind=case c of
                          #"d" => INT StringCvt.DEC
                        | #"s" => STRING
                        | #"f" => REAL (StringCvt.FIX NONE)
                        | #"e" => REAL (StringCvt.SCI NONE)
                        | #"g" => REAL (StringCvt.GEN NONE)
                        | _ => raise formatError},
             s)
         end
    end
fun parse s =
    let
      val (s1,s) = StringCvt.splitl (fn c => c <> #"%") S.getc s
      val prefix = if s1 = "" then nil
                   else [LITERAL s1]
    in if S.isEmpty s then prefix
       else let val (f,s) = oneFormat s
                val L = parse s
            in prefix@(f::L)
            end
    end
fun format s L =
    let val FL = parse (S.full s)
        fun splice (h::t) L =
            (case h of
               LITERAL s => s ^ (splice t L)
             | SPEC s => (formatData s (List.hd L) ^ (splice t (List.tl L))))
          | splice nil l = ""
    in
      (splice FL L)
```

```
        end
    fun printf s L = print (format s L)
end
```

問 16.10 日時を表す書式を以下のように定める．

%H	24 時間制の時間（00 から 23）
%I	12 時間制の時間（01 から 12）
%k	24 時間制の時間（0 から 23）
%M	分
%S	秒
%d	日（01 から 31)
%m	月（01 から 12）
%Y	年

format の定義を参考にして，日時に関する以下の埋め込み書式指定を含む文字列を受け取り，日時をプリントする関数

```
showTime : string -> unit
```

を書け．たとえば

```
- showTime "The time is %H hour %M minutes on %m/%d/%Y.\n";
The time is 21 hour 04 minutes on 9/9/2000.
val it = () : unit
```

のように動作をする．

解答例

```
structure ShowTime =
struct
local
  structure S = Substring
in
  exception FormatError
  fun dH t = StringCvt.padLeft #"0" 2 (Int.toString (Date.hour (Date.fromTimeLocal t)))
  fun dI t =
      let val h = Date.hour (Date.fromTimeLocal t)
          val i = if h = 0 then 12 else if h > 12 then h - 12 else h
      in StringCvt.padLeft #"0" 2 (Int.toString i)
      end
  fun dk t = Int.toString (Date.hour (Date.fromTimeLocal t))
  fun dM t = Int.toString (Date.minute (Date.fromTimeLocal t))
```

```
fun dS t = Int.toString (Date.second (Date.fromTimeLocal t))
fun dd t= StringCvt.padLeft #"0" 2
            (Int.toString (Date.day (Date.fromTimeLocal t)))
fun dm t = case Date.month (Date.fromTimeLocal t) of
             Date.Jan => "01"
           | Date.Feb => "02"
           | Date.Mar => "03"
           | Date.Apr => "04"
           | Date.May => "05"
           | Date.Jun => "06"
           | Date.Jul => "07"
           | Date.Aug => "08"
           | Date.Sep => "09"
           | Date.Oct => "10"
           | Date.Nov => "11"
           | Date.Dec => "12"
fun dY t = Int.toString (Date.year (Date.fromTimeLocal t))
datatype spec = EMBED of Time.time -> string | LITERAL of string
fun oneFormat s =
    let val s = S.triml 1 s
    in if S.isPrefix "%" s then (LITERAL "%",S.triml 1 s)
       else
         let
           val (c,s) = case S.getc s of NONE => raise FormatError
                                      | SOME s  => s
         in (EMBED (case c of
                     #"H" => dH
                   | #"I" => dI
                   | #"k" => dk
                   | #"M" => dM
                   | #"S" => dS
                   | #"d" => dd
                   | #"m" => dm
                   | #"Y" => dY
                   | _ => raise FormatError),
             s)
         end
    end
fun parse s =
```

```
        let
          val (s1,s) = StringCvt.splitl (fn c => c <> #"%") S.getc s
          val prefix = if s1 = "" then nil
                       else [LITERAL s1]
        in if S.isEmpty s then prefix
           else let val (f,s) = oneFormat s
                    val L = parse s
                in prefix@(f::L)
                end
        end
    fun format s tm =
        let val FL = parse (S.full s)
            fun splice (h::t) =
                (case h of
                   LITERAL s => s ^ (splice t)
                 | EMBED f => f tm ^ (splice t))
              | splice nil = ""
        in
          splice FL
        end
    fun showTime s =
        print (format s (Time.now()))
end
end
```

問 16.11 Format ストラクチャを用いて，第 13 章で作成したソート関数の評価プログラムの印字処理部分を書き直せ.

解答例 以下に変更部分の例を示す。

```
fun evalSort L =
    let
      val L' = map checkTime L
      val av = (foldr (fn ((_,_,x),y) => y+x) 0.0 L')/(Real.fromInt (List.length L'))
      fun printLine (n,a,c) =
          Format.printf  "%20d%20d%20f\n" [Format.I n, Format.I a, Format.R c]
     in
       (Format.printf
          "%20s%20s%20s\n"
           [Format.S "array size", Format.S "milli-sec.", Format.S "micro s./nlogn"];
        map printLine L';
```

```
    print "------------------------------------------------------------\n";
    Format.printf "%40s%20f\n" [Format.S "avarage", Format.R av]
  )
end
```

第17章　OSとのインターフェイス

17.2　ディレクトリとファイルの操作

問 17.1 上記の ls 関数を改良し，各ファイルごとに，ファイルの種類（ディレクトリ，リンク，通常ファイル），ファイルのアクセス（読み出し可，書き込み可，実行可），最終更新年月日，ファイルサイズを以下のような形式で表示するようにせよ.

```
- ls();
dlrwx            file size       last modified            file name
d-rwx                 4096        Mar 19 10:50             examples
--rw-                18666        Apr  8 09:39               ml.tex
--rw-               257094        Mar 18 16:40            part1.tex
--rw-               186111        Apr 16 10:58            part2.tex
```

先頭の dlrwx はそれぞれ，ディレクトリ，リンク，読み込み可，書き込み可，実行可を表す.

解答例

```
fun ls () =
    let
      val d = F.openDir (F.getDir())
      fun printRest () =
          case F.readDir d of
            NONE => F.closeDir d
          | SOME f =>
            let
           val size = F.fileSize f
           val time = Date.toString (Date.fromTimeLocal (F.modTime f))
              val modString = implode
                                [if F.isDir f then #"d" else #"-",
                                 if F.isLink f then #"l" else #"-",
                                 if F.access (f, [F.A_READ]) then #"r" else #"-",
                                 if F.access (f, [F.A_WRITE]) then #"w" else #"-",
                                 if F.access (f, [F.A_EXEC]) then #"x" else #"-"]
        in
             (Format.printf
```

```
                  "%10s%15d%30s%30s\n"
                  [Format.S modString,
                   Format.I size,
                   Format.S time,
                   Format.S f
                  ];
            printRest())
        end
    in
      (Format.printf
         "%10s%15s%30s%30s\n"
         [Format.S "dlrwx",
          Format.S "file size",
          Format.S "last modified",
          Format.S "file name"
         ];
       printRest()
      )
    end
```

問 17.2 copy 関数が，ファイルおよびディレクトリの更新日付を保存するように変更せよ.

解答例 copy 関数に F.setTime コマンドを追加すればよい。変更例を以下に示す。

```
fun copy a b =
    if not (F.isDir a) then
      (copyFile a b;
       F.setTime (b,SOME (F.modTime a)))
    else
      let val d = F.openDir a
          fun copyDirStream d b =
              case F.readDir d of
                NONE => F.closeDir d
              | SOME item =>
                let val from = P.concat (a,item)
                    val to = P.concat (b,item)
                in (copy from to;copyDirStream d b)
                end
      in
        (F.mkDir b;
         copyDirStream d b;
```

```
      F.setTime (b,SOME (F.modTime a)))
  end
```