

プログラミング言語レポート

氏名: 赤松 佑哉 (akamatsu, YUYA)

学生番号: 09B23595

出題日: 2024 年 5 月 9 日

提出日: 2024 年 6 月 4 日

締切日: 2024 年 6 月 4 日

1 概要

講義を通して学んだ関数型言語 SML 言語を実際の問題解決を通して実践した。今回、C 言語の標準ライブラリに存在する文字列操作関数、`strcat`、`strcmp`、`strcpy`、`strex`、`strlen`、`sort` と同等の操作をリストに行う SML プログラムを作成した。いかに各関数の概要を示す。

- `listcat` : `strcat` に相当し、2 つのリストを連結する。
- `listcmp` : `strcmp` に相当し、2 つのリストが要素・順序ともに等しいか判定する。
- `listcpy` : `strcpy` に相当し、リストの複製（コピー）を行う。
- `listex` : `strex` のように、条件を満たす要素がリスト中に存在するかを確認する（高階関数を用いる）。
- `listlen` : `strlen` に相当し、リストの長さ（要素数）を返す。
- `listsort` : `sort` に対応し、マージソートによりリストを昇順に並べ替える。

作成したプログラムは第 9 章に添付している。≡

2 `listcat.sml` の実装

二つのリストを引数として受け取り、それらを結合した一つのリストを返す `listcat` 関数を実装した。パターンマッチングを用いて引数の構造に応じた場合分けを行い、異なる処理を実現している。

片方が空リストであれば、もう片方のリストをそのまま返す。両方のリストに要素がある場合は、第一引数の先頭要素を取り出し、それを再帰的に呼び出された `listcat` の戻り値の先頭に加えることでリストを構築する。

この再帰処理は、第一引数が空リストとなった時点で終了し、第二引数そのまま返される。このとき、再帰は打ち切れ、結合されたリストが得られる。

以下のテスト入力により動作確認を行った。

```
val test1 = listcat([1,2,3], [4,5])
val test2 = listcat([1,2,3], [])
```

```
val test3 = listcat([], [1,2,3])
val test4 = listcat([], [])
```

実行結果は次のようになった .

```
$ sml listcat.sml
Standard ML of New Jersey v110.79 [built: Sat Oct 26 12:27:04 2019]
[opening listcat.sml]
val listcat = fn : int list * int list -> int list
val test1 = [1,2,3,4,5] : int list
val test2 = [1,2,3] : int list
val test3 = [1,2,3] : int list
val test4 = [] : int list
```

結果を見るときちんと二つのリストを合併し一つのリストの作成に成功している . また空リストの時もきちんと動作してるのがわかる . 以上が二つのリストの合併 `listcat` の実装 , 解説である . 作成した SML ソースコードは第 9.1 章に添付している .

3 listcmp.sml の実装

二つのリストを引数として受け取り , 対応する要素の差をすべて合計し返す `listcmp` 関数を実装した . `strcmp` 関数のような等価判定ではなく , 数値的な差異を合計してリスト間の「違いの度合い」を測ることができる .

関数内部ではパターンマッチングを用い , 両方が空リストなら 0 を返す . 一方が空リストで他方に要素が残っている場合には , 残りの要素をすべて加算して返す . 両方に要素がある場合には , 先頭要素の差を取り , 再帰的に残りのリストに対して同様の処理を行っている .

以下の入力を引数として渡し , 動作確認を行った .

```
val test1 = listcmp([1,2,3], [1,2,3])
val test2 = listcmp([1,2,3], [1,2,2])
val test3 = listcmp([1,2,3], [1,2,4])
val test4 = listcmp([1,2,3], [1,2])
val test5 = listcmp([1,2], [1,2,3])
val test6 = listcmp([], [])
```

実行結果は次のようになった .

```
$ sml listcmp.sml
Standard ML of New Jersey v110.79 [built: Sat Oct 26 12:27:04 2019]
[opening listcmp.sml]
val listcmp = fn : int list * int list -> int
val test1 = 0 : int
val test2 = 1 : int
val test3 = ~1 : int
```

```
val test4 = 3 : int
val test5 = 3 : int
val test6 = 0 : int
```

結果を見るときちんと二つのリストの辞書的な差異を数値として出力できている．リストどおしの要素数が違う場合もきちんと動作している．以上がリストを比較する `listcmp` 関数の実装，解説である．作成した SML ソースコードは第 9.2 章に添付している．

4 `listcpy.sml` の実装

リストの内容をそのままコピーした新しいリストを返す関数 `listcpy` を実装した．この関数は，C 言語における `strcpy` のように元のリストを変更せずに，その要素を順にコピーして新しいリストを構築する機能を持つ．

SML ではリストはイミュータブル（不変）であるため，コピー処理は新しいリストを再帰的に構築する形で実現される．具体的には，パターンマッチングにより，空リストに到達した場合には空リストを返す終了条件とし，非空の場合には先頭要素を新しいリストの先頭に追加しながら再帰呼び出しを行っている．

以下のテストコードにより動作確認を行った．

```
val test1 : int list = listcpy[]
val test2 = listcpy[1,2,3,4]
```

実行結果は次のようになった．

```
$ sml listcpy.sml
Standard ML of New Jersey v110.79 [built: Sat Oct 26 12:27:04 2019]
[opening listcpy.sml]
val listcpy = fn : 'a list -> 'a list
val test1 = [] : int list
val test2 = [1,2,3,4] : int list
```

この結果より，空リストに対しても，任意の整数リストに対しても正しくコピーされたリストを返すことが確認できた．このように，再帰を用いた基本的なリスト操作を通じて，SML の関数型プログラミングの特性を活かした実装を行った．作成した SML ソースコードは第 9.3 章に添付している．

5 `listexi.sml` の実装

`listexi` 関数は，整数とリストを引数に取り，その整数がリスト中に何回出現するかをカウントして返す関数である．パターンマッチングを用いて，リストの構造に応じた再帰処理を行うことで実現している．

まずリストが空である場合（終了条件）には 0 を返す．それ以外の場合には，リストの先頭要素と比較し，一致していれば 1 を加算し，再帰的に残りのリストで同様の判定を行う．この処理により，リスト中の該当要素の合計出現回数を求めることができる．

以下にテスト入力とその実行結果を示す．

```
val test1 = listexi(1, [1,2,3])
val test2 = listexi(0, [1,2,3])
val test3 = listexi(1, [1,2,1])
val test4 = listexi(1, [])
```

実行結果は次のようになった。

```
L$ sml listexi.sml
Standard ML of New Jersey v110.79 [built: Sat Oct 26 12:27:04 2019]
[opening listexi.sml]
listexi.sml:4.27 Warning: calling polyEqual
val listexi = fn : ''a * ''a list -> int
val test1 = 1 : int
val test2 = 0 : int
val test3 = 2 : int
val test4 = 0 : int
```

実行結果より，listexi 関数は指定した要素の出現回数を正しくカウントしていることが確認できる．この関数はリスト探索と条件分岐，再帰処理の組み合わせにより実装されており，基本的なリスト操作の理解に適した例である．作成した SML ソースコードは第 9.4 章に添付している．

6 listlen.sml の実装

listlen 関数は，引数として与えられたリストの長さ（要素数）を求めて返す関数である．この関数は，パターンマッチングと再帰処理を用いてシンプルに実装されており，リスト操作の基礎的な理解に適している．

リストが空であれば長さは 0 を返す．そうでない場合（リストが少なくとも 1 つの要素を持つ場合），リストの先頭を除いた残りのリストに対して再帰的に listlen を呼び出し，その結果に 1 を加えることで全体の長さを求めている．

以下にテスト入力とその実行結果を示す．

```
val test1 = listlen([])
val test2 = listlen([1])
val test3 = listlen([1, 2])
val test4 = listlen([1,2,3])
```

実行結果は次のようになった．

```
val test1 = 0 : int
val test2 = 1 : int
val test3 = 2 : int
val test4 = 3 : int
```

実行結果から，空リストを含むさまざまな長さのリストに対して，listlen 関数が正しく要素数を返していることが確認できる．再帰を通じてリストを 1 つずつ処理しながら合計を求める構造は，

SML における関数型プログラミングの基本的な設計手法を反映している．作成した SML ソースコードは第 9.5 章に添付している．また，以下に畳み込み関数を用いたリストの長さを求めるプログラムをも別途作成した．

```
fun listlen2 L =  
  foldr (fn (x, y) => y + 1) 0 L;  
val test1 = listlen2([])  
val test2 = listlen2([1])  
val test3 = listlen2([1, 2])  
val test4 = listlen2([1,2,3])
```

実行結果は次のようになった．

```
val listlen2 = fn : 'a list -> int  
val test1 = 0 : int  
val test2 = 1 : int  
val test3 = 2 : int  
val test4 = 3 : int
```

このように，畳み込み関数を使用した際もきちんと動作していることがわかる．プログラムの説明をしておくと，まず畳み込み関数のリストの各要素に対する処理として $y+1$ と定義している．その理由が，リストの各要素ごとにリストの要素とは関係なしに 0 から 1 加算しているためである．こうすることで，リストの持っている要素の分だけ 1 が加算され結局それはリストの長さとも一致するということだ．

7 listsort.sml

整数リストを昇順に整列する関数 `listsort` を実装した．この関数はマージソート (Merge Sort) アルゴリズムに基づいており，以下の 3 つの関数から構成される

- `merge`: 2 つの整列済みリストを結合して新たな整列済みリストを生成する補助関数
- `listsplit`: リストをおよそ半分に分割する補助関数
- `listsort`: 与えられたリストを再帰的に分割し，整列しながら結合するメイン関数

各関数を以下の第 7.1 章，7.2 章に詳細を記述している以下はテスト入力とその実行結果である．

```
val test1 = listsort [5,4,3,2,1]  
val test2 = listsort [2,3,1,2]  
val test3 = listsort []  
val test4 = listsort [4,3,2,1]
```

実行結果は次のようになった．

```
$ sml listsort.sml  
Standard ML of New Jersey v110.79 [built: Sat Oct 26 12:27:04 2019]  
[opening listsort.sml]
```

```

val merge = fn : int list * int list -> int list
val listsplit = fn : 'a list -> 'a list * 'a list
val listsort = fn : int list -> int list
val test1 = [1,2,3,4,5] : int list
val test2 = [1,2,2,3] : int list
val test3 = [] : int list
val test4 = [1,2,3,4] : int list

```

結果より，`listsort` 関数が再帰的な分割・統合処理を通じて，任意の順序のリストを正しく昇順に整列していることが確認できた．このように，関数型言語 SML においても効率的な整列アルゴリズムを簡潔に表現できることが示された．ソースコードは第 9.6 章に示す．

7.1 listsplit.sml

与えられたリストを約半分ずつの二つのリストに分割する `listsplit` 関数を実装した．この関数は，マージソートなどの分割統治法を用いるアルゴリズムの前処理として有用である．

関数の内部には，分割処理を再帰的に行う補助関数 `loop` を定義しており，`lst` を先頭から 2 つずつ取り出しながら左右のリスト `left` , `right` に交互に振り分けていく方式を採用している．処理終了後に `rev` を用いてリストを元の順序に戻している点が特徴である．`rev` (リストの反転) を使う利点は，リストの末尾に要素を追加するコストを回避しつつ，意図した順序のリストを得られるという点からだ．以下はテスト入力とその実行結果である．

```

val test1 = listsplit([1,2,3,4])
val test2 = listsplit([1,2,3,4,5])
val test3 = listsplit([])
val test4 = listsplit([1])

```

実行結果は次のようになった．

```

val test1 = ([1,3],[2,4]) : int list * int list
val test2 = ([1,3,5],[2,4]) : int list * int list
val test3 = ([],[ ]) : int list * int list
val test4 = ([1],[ ]) : int list * int list

```

このように，リストが空，1 要素，偶数個，奇数個の場合にも対応しており，安定して左右に要素を交互に分けることができる実装であることが確認できた．作成したソースコードは第 9.7 章に添付している．

7.2 マージソートの実装

`listsplit` 関数により，リストを 2 要素ずつに分割する．分割し終わったらその中で，大小関係の条件分岐によりソートを行う．ソートを終わると，ソート済みのリストどうしをさらに前の要素から大小関係で場合分けを行いながら，マージしていく．この動作を再帰の終了条件はリストの分割終了と呼び出し，それぞれについてソート，マージをすると最終的にはソート済みの一つのリスト

となるという手続きである．

今回，クイックソートでなくマージソートを用いた理由は，再帰的なリスト操作が今回の講義の内容に沿っており，またマージソートの持つ計算量の特性にもある．

マージソートは，入力の形に関係なく常に 最悪計算量が $\mathcal{O}(n \log n)$ である という点で非常に安定したソートアルゴリズムである．この安定性は，たとえ入力ですでにソートされている場合や逆順である場合でも計算量が大きく悪化することがないことを意味している．

分割回数は最大で $\log_2 n$ 回に達し，各段階で全体の n 要素に対するマージ操作を行うため，トータルの計算量は常に $n \log n$ に抑えられる．そのため，大規模なデータや最悪ケースの発生が懸念される状況でも予測可能な処理時間を維持することができる．

8 SML や講義に関する所感

今回関数型言語を初めて本格的に学び，新たな発見，関数型言語の強みについて知ることができた．実際，初めは C 言語に備わっているような様々な便利な関数が使えず，記述規則も目新しいもので苦労した．しかし，条件分岐，再帰呼び出しだけで想像よりたくさんのことが実装できていて驚いた．何とんでも副作用がないというだけでこんなにもデバッグ，拡張，が容易になるとは思わなかった．引数を固定するだけで新たな関数として扱えるのも今回の課題では扱わなかったが，拡張性，柔軟性の面でとても強みとなると感じた．講義ではあまり深いところまでは触れなかったが，Fortran をはじめとする命令型言語の歴史もとても面白かった．すべての言語は作られた背景があり，今の便利な環境は今までの言語の改良の積み重ねによるものであると再認識した．

9 作成したプログラムのソースコード

9.1 listcat.sml のソースコード

```
1: fun listcat([], x) = x : int list
2:   |listcat(x, []) = x : int list
3:   |listcat(x::xs, y) = x :: listcat(xs, y)
```

9.2 listcmp.sml のソースコード

```
1: fun listcmp([], []) = 0
2:   |listcmp([], x::xs) = x + listcmp([], xs)
3:   |listcmp(x::xs, []) = x + listcmp(xs, [])
4:   |listcmp(x::xs, y::ys) =
5:     let
6:       val diff = x - y
7:     in
8:       diff + listcmp(xs, ys)
9:     end;
```

9.3 listcpy.sml のソースコード

```
1: fun listcpy [] = []
2:   |listcpy (x::xs) = x :: listcpy (xs);
```

9.4 listexi.sml のソースコード

```
1: fun listexi(x, []) = 0
2:   | listexi(x, y::ys) =
3:     let
4:       val check = if (x = y) then 1 else 0
5:     in
6:       check + listexi(x, ys)
7:     end;
```

9.5 listlen.sml のソースコード

```
1: fun listlen([]) = 0
2:   | listlen(x::xs) = 1 + listlen(xs)
```

9.6 listsort.sml のソースコード

```
1: fun merge([], ys) = ys
2:   | merge(xs, []) = xs
3:   | merge(x::xs, y::ys) =
4:     if x <= y then x :: merge(xs, y::ys)
5:     else y :: merge(x::xs, ys);
6:
7: fun listsplit lst =
8:   let
9:     fun loop([], left, right) = (rev left, rev right)
10:    | loop([x], left, right) = (rev (x::left), rev right)
11:    | loop(x::y::rest, left, right) = loop(rest, x::left, y::right)
12:   in
13:     loop(lst, [], [])
14:   end;
15:
16: fun listsort([]) = []
17:   | listsort([x]) = [x]
18:   | listsort L =
19:     let
20:       val (left, right) = listsplit L
21:     in
22:       merge(listsort left, listsort right)
23:     end;
```

9.7 listsplit.sml のソースコード

```
1: fun listsplit lst =
2:   let
3:     fun loop([], left, right) = (rev left, rev right)
4:    | loop([x], left, right) = (rev (x::left), rev right)
5:    | loop(x::y::rest, left, right) = loop(rest, x::left, y::right)
6:   in
7:     loop(lst, [], [])
8:   end;
```