プログラミング言語論・テスト問題用紙

('00 年 7 月 27 日・ 8:50 〜 10:20)

解答上、その他の注意事項

I. 問題は、問 I 〜 VI までである。
II. 解答用紙の右上の欄に学籍番号・名前を記入すること。
III. 解答欄を間違えないよう注意すること。
IV. 選択式でない問で解答欄がマス目になっている場合は、1 字に 1 マスを用いること。特に空白にも必ず 1 マスを用いること
V. 解答中の文字 (特に a と d) がはっきりと区別できるよう注意すること。
VI. ノート・プリント・参考書などは持ち込み可である。
VII. テストの配点は 75 点である。合格はレポートの得点を加点して、100 点満点中 60 点以上とする。

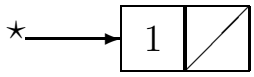
このテストでの表記上の注意について:
(最初に必ず読んで下さい。)
Schemeでは、$(a_1\ a_2\ \ldots a_n)$という括弧を使った式は、ユーザが入力するときは関数適用の意味に、処理系が出力するときはリストの意味になる。ユーザがリストを入力する時は「'」(クウォート記号)をつけて '(1 2 3 4) のように書く必要がある。これでは、計算の途中結果を示すなど時に不便なので、このテストではまぎらわしい時はリストを表す括弧はクウォート記号を使わずに大括弧(square bracket、「[」と「]」)で表すことにする。
例: [1 2 3 4], [[1 2] [4 5]] など

I. 以下の通常の数学の記法を Scheme の記法にせよ。

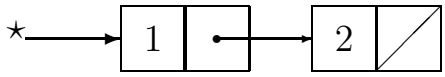
(1) $1-2\times 3+4$

(2) $\dfrac{1}{2-\sqrt{5}}$ (ヒント — 平方根を求める関数は Scheme では sqrt)

II. 箱矢印記法(box-pointer notation)では、cons セルを 2 つのつながった箱(box)で表す。左側の箱には car の内容が、右側の箱には cdr の内容が書かれる。
リストは、このような箱への矢印(pointer)として表される。また、空リストは特別に斜線で表す。
例えば、carが 1で cdrが空リストであるようなリストは、



[1 2] というリストは



の $\star$ のついている矢印で表される。

(1) [1 2 3 4] というリストと、

(2) [[1 2] [3 4]] というリストを

箱矢印記法で表し、その矢印に $\star$ 印をつけよ。

III. 次のように関数 rev-append を定義する。

```
(define (rev-append xs ys)
   (if (null? xs) ;  下線部を 式1とする
       ys         ;  下線部を 式2とする
       (rev-append (cdr xs) (cons (car xs) ys))))   ;  下線部を 式3とする
```

例えば、(rev-append [1 2] [4 5]) は次のように計算される。ここで下線部は、計算が行なわれる部分を示している。

```
(rev-append [1 2] [4 5]))
⇒ (rev-append [2] (cons 1 [4 5]))    ; 式1が偽なので式3を選択する。
⇒ (rev-append [2] [1 4 5])           ; cons の計算
⇒ (rev-append [] (cons 2 [1 4 5]))   ; 式1が偽なので式3を選択する。
⇒ (rev-append [] [2 1 4 5])          ; cons の計算
⇒ [2 1 4 5]                          ; 式1が真なので式2を選択する。
```

この例にならって、次のように定義された関数 append

```
(define (append xs ys)
   (if (null? xs)      ;  下線部を 式 1 とする
       ys              ;  下線部を 式 2 とする
       (cons (car xs) (append (cdr xs) ys))));  下線部を 式 3 とする
```

について、(append [1 2] [4 5]) の評価（計算）の様子を書き下せ。リストを表す括弧には、上の rev-append の例のように大括弧（square bracket）を使え。

IV. 2 つの昇順に並んだ数のリストを、1 つの昇順のリストに併合する関数 merge を次のように定義する。例えば、(merge [1 3 5] [2 4]) は [1 2 3 4 5] に、(merge [1 4 7] [2 4 6]) は、[1 2 4 4 6 7] になる。空欄を埋め merge の定義を完成させよ。

```
(define (merge xs ys)
   (if (null? xs) ys
      (if (null? ys) xs
          (if (< (car xs) (car ys))
              [          (1)          ]
              [          (2)          ])))))
```

V. 次のような関数 exist?, filter を定義する（filter は配布プリントに定義してあったものと同一である。）

```
(define (exist? p xs)
   (if (null? xs) #f
       (or (p (car xs)) (exist? p (cdr xs)))))

(define (filter p xs)
   (if (null? xs) '()
       (if (p (car xs))
           (cons (car xs) (filter p (cdr xs)))
           (filter p (cdr xs)))))
```

これらの関数を用いて、x がリスト xs に要素として含まれているかどうかを判定する述語 member?、2 つのリストに共通に含まれている要素のリストを返す関数 intersection を定義する。例えば、

- (member? 4 [1 4 7]) ⇒#t
- (member? 3 [1 4 7]) ⇒#f
- (intersection [2 3 4] [1 4 7]) ⇒[4]
- (intersection [2 5 8] [1 4 7]) ⇒[]

である。

以下の空欄に適切な lambda 式 を入れて定義を完成させよ。

ただし 2 つの要素が等しいかどうかを判定する述語は eqv?を使用せよ。

eqv?の使用例:

- (eqv? 1 2) ⇒#f
- (eqv? #t #t) ⇒#t

```
(define (member? x xs) (exist? [     (1)     ] xs))

(define (intersection xs ys)
   (filter [     (2)     ] ys))
```

VI. リスト型を C の構造体として、次のように定義する。（ただし、リストの要素としては int 型のみを想定する。）

```
struct _List {
  int car;
  struct _List* cdr;
};

typedef struct _List* List;
```

また空リストは NULL で表す。
次の Scheme の関数とほぼ同等の動きをする（つまり、リスト中の最大値を求める）C の関数 mymaximum を定義せよ。

```
(define (mymaximum xs)
  (if (null? (cdr xs))
     (car xs)
     (max (car xs) (mymaximum (cdr xs)))))
```

ただし、次のような補助関数は定義されているものとして使用して良い。

```
int max(int x, int y) {
  if (x>y) { return x; } else { return y; }
}
```

プログラミング言語論・テスト解答用紙('00年 7月 27日)

学部	学籍番号		氏名	

I. (5点 ×2)

(1).	
(2).	

II. (6点 ×2)

(1).	
(2).	

III. (8点)

```
(append [1 2] [4 5])
```

⇒

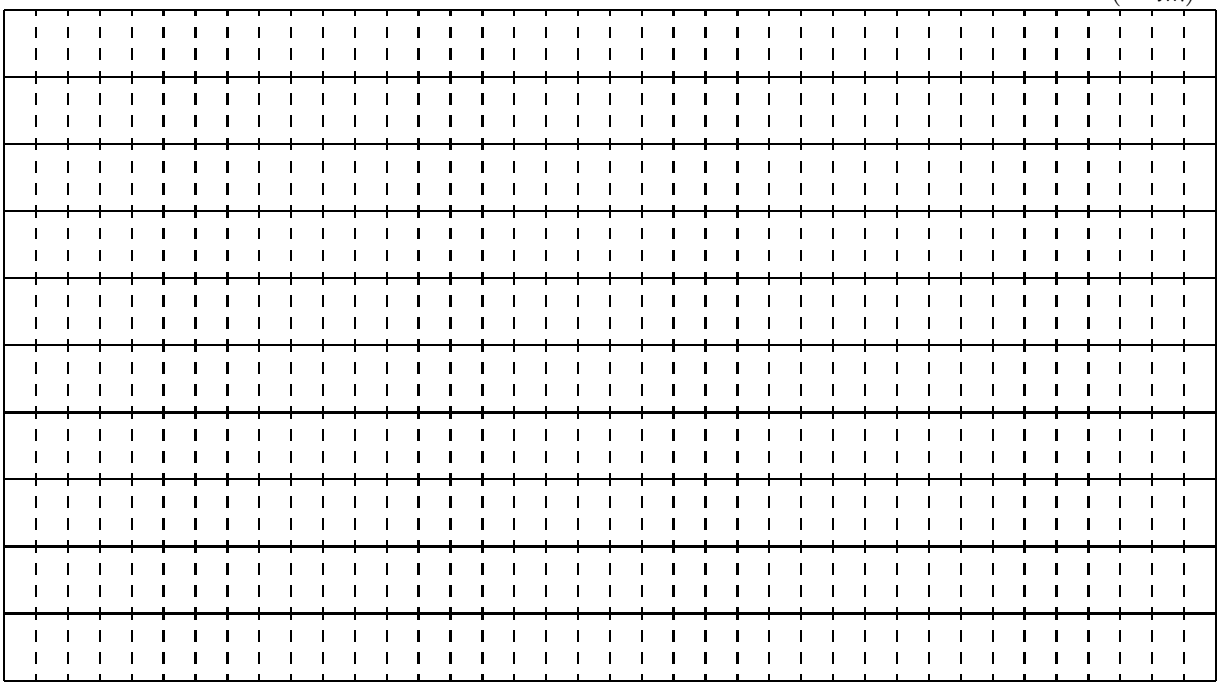
IV. (7点 ×2)

(1).	
(2).	

V. (8 点 ×2)

(1).	
(2).	

VI. (15 点)



授業・テストの感想