Answer Set 3 & 4

OUT: 18 OCTOBER, 1994

EX2.1.3 A:

P	∧	Q	⇒	P
T	T	T	T	T
T	F	F	T	T
F	F	T	T	F
F	F	F	T	F
1	3	2	5	4

EX2.1.3 B:

(¬ P	⇒	(P	∧	Q))	≡	P
F	T	T	T	T	T	T
F	T	T	F	F	T	T
T	F	F	F	T	T	F
T	F	F	F	F	T	F
4	5	1	3	2	7	6

EX2.1.3 C:

P	∧	(P	⇒	Q)	⇒	Q
T	T	T	T	T	T	T
T	F	T	F	F	T	F
F	F	F	T	T	T	T
F	F	F	T	F	T	F
4	5	1	3	2	7	6

EX2.1.3 D:

(P	∧	Q)	∨	(¬P	∧	R)	⇒	Q	∨	R
T	T	T	T	F	F	T	T	T	T	T
T	F	F	F	F	F	T	T	F	T	T
F	F	T	T	T	T	T	T	T	T	T
F	F	F	T	T	T	T	T	F	T	T
T	T	T	T	F	F	F	T	T	T	F
T	F	F	F	F	F	F	T	F	F	F
F	F	T	F	T	F	F	T	T	T	F
F	F	F	F	T	F	F	T	F	F	F
1	3	2	7	4	6	5	11	8	10	9

EX2.1.3 E:

(P	∨	Q)	∧	(¬P	∨	R)	⇒	Q	∨	R
T	T	T	T	F	T	T	T	T	T	T
T	T	F	T	F	T	T	T	F	T	T
F	T	T	T	T	T	T	T	T	T	T
F	F	F	F	T	T	T	T	F	T	T
T	T	T	F	F	F	F	T	T	T	F
T	T	F	F	F	F	F	T	F	F	F
F	T	T	T	T	T	F	T	T	T	F
F	F	F	F	T	T	F	T	F	F	F
1	3	2	7	4	6	5	11	8	10	9

EX2.2.3: Given $0 \leq i < j \leq N$: (a) the smallest i is 0; (b) the largest i is $N-1$; (c) the smallest j is 1; (d) the largest j is N; (e) the smallest N is 1.

EXERCISE 3 If F and G are tautologically equivalent, then for whatever the program's state, F is true if and only if G is true. Hence, whatever branch is taken in,

```
if F then S else T
```

will be taken in,

```
if G then S else T.
```

Neither F nor G have side effects. In each statement, the world is affected and transformed only by S or T, whichever is run. Hence, under any circumstance, either of these two statements causes the same code to be run to affect the world. Hence they affect the world exactly the same.

EXERCISE 2.5.2

(a) $s = A + B$.

(b) $z = (x + y)/2$.

(c) $k|x$.

(d) $f|N$.

(e) $x = 2^k$, some integer k.

(f) $(D|X) \wedge (D|Y)$.

(g) $X = qD + R$, for integers q, r.

EXERCISE 2.5.3

(a) $z = \min(X, Y)$.

(b) $z < \min(X, Y)$.

(c) $(k \leq N) \wedge (k = 2^l) \wedge (N < 2^{l+1})$, for some integer l.

(d) $(N < k) \wedge (k = 2^l) \wedge (2^{l-1} \leq N)$, for some integer l.

(e) $(a^2 \leq N) \wedge (N < (a + 1)^2)$.

(f) $(1 \leq x \leq 100) \wedge (y|x)$.

EXERCISE 2.5.4

(a) $a = \min(A, B) \wedge b = \max(A, B)$.

(b) $(a \le b \le c) \wedge \big(((a = X) \wedge (b = Y) \wedge (c = Z)) \vee ((a = X) \wedge (c = Y) \wedge (b = Z)) \vee ((b = X) \wedge (a = Y) \wedge (c = Z)) \vee ((b = X) \wedge (c = Y) \wedge (a = Z)) \vee ((c = X) \wedge (a = Y) \wedge (b = Z)) \vee ((c = X) \wedge (b = Y) \wedge (a = Z)) \big)$.

(c) $(X = \min(X, Y, Z) \Rightarrow p = 1) \wedge (Y = \min(X, Y, Z) \Rightarrow p = 2) \wedge (Z = \min(X, Y, Z) \Rightarrow p = 3)$. What if $X = Y$?

----------ANSWER4.PAS----------

```
[ENVIRONMENT,INHERIT('mystring2')]

module answer4(input,output) ;

(* Answer to problem set 4,
   move-to-front heuristic for the unique words program
   (c) Burton Rosenberg 1994
*)
type
(* a mylist has a dummy header. *)
   MyList = ^MyListElement ;
   MyListElement = record
                   theMs : MyString ;
                   count : integer ;
                   next : MyList ;
               end ;
   MyListCoupon = MyList ;

procedure create_MyList(var ml : MyList ) ;
(* Create dummy header for an empty list and return pointer in ml. *)
begin
   new(ml) ;
   ml^.count := 0 ;
   ml^.next := NIL ;
   ml^.theMS.length := 0 ; (* WARNING: breakdown of mystring abstraction *)
end ;

procedure print_MyList( ml : MyList ) ;
(* Print mylist ml. No side effects. *)
begin
   ml := ml^.next ; (* skip dummy header *)
   while (ml<>NIL) do begin
       write(ml^.count:3,' ') ;
       writeln_MyString( ml^.theMs ) ;
```

```
       ml := ml^.next
   end
end ;

procedure add_MyList( var ml: Mylist; data: MyString; count: integer ) ;
(* Add to the front of MyList ml a record containing MyString data and
   count count. *)
var p : MyList ;
begin
   new(p) ;
   p^.count := count ;
   copy_MyString( p^.theMs, data ) ;
   p^.next := ml^.next ;
   ml^.next := p
end ;

function find_MyList( ml: MyList; toFind: MyString ) : MyListCoupon ;
(* Search for MyString tofind on MyList ml. Return non-NIL coupon if
   found, else return NIL *)
var p : MyList ;
begin
   p := NIL ;
   (*L.I. toFind not found on list up till predecessor of ml^.
          ml^ to check *)
   while ( p<>ml^.next ) do
      if compareNC_MyString(ml^.next^.theMs, toFind) then p := ml^.next
      else ml := ml^.next ;
   if (ml^.next=NIL) then find_MyList := NIL
   else find_MyList := ml ;
end ;

procedure Increment_MyList( ml: Mylist; coup: MyListCoupon ) ;
begin  coup^.next^.count := coup^.next^.count+1   end ;

procedure Statistics_MyList( ml: MyList; var len: integer; var sum: integer ) ;
(* Calculate length and sum of count fields of MyList ml. *)
begin
   len := 0 ;
   sum := 0 ;
   ml := ml^.next ; (* skip dummy header *)
   (* L.I. len is number of list items up till, not including ml^,
           sum is the sum of count fields for list items up till,
           not including ml^.count. *)
   while (ml<>NIL) do begin
```

```
      len := len+1 ;
      sum := sum + ml^.count ;
      ml := ml^.next
   end
end ;

procedure MoveToFront_MyList( ml: MyList; coup: MyListCoupon ) ;
(* Move list item indicated by MyListCoupon coup to front of MyList ml. *)
var p: MyList ;
begin
   (*ASSERT: coup<>NIL and coup^.next<>NIL *)
   p := coup^.next ;
   coup^.next := p^.next ;
   p^.next := ml^.next ;
   ml^.next := p
end ;
end.
```

----------DRIVER4.PAS----------

```
[INHERIT('MYSTRING2','ANSWER4')]
program driver1( input, output ) ;

(*  driver program to test answer4.pas
   (c) Burton Rosenberg, U. of Miami 1994
*)
var
   ml : MyList  ;
   aLine, aWord : MyString ;
   p : MyListCoupon ;
   i : integer ;
   numWords, totalCount : integer ;

begin
   create_MyList( ml ) ;
   i := readln_MyString( aLine ) ;

   (* L.I. aLine has a line of text yet to process *)
   while ( i<>MYEOF ) do begin
      parse_MyString( aLine, aWord ) ;

      (* L.I. aWord has a word to process *)
      while ( length_MyString( aWord ) > 0 ) do begin
         p := find_MyList( ml, aWord ) ;
```

```
         if (p=NIL) then
            (* word not found, add to list w/ count=1 *)
            add_MyList( ml, aWord, 1 )
         else begin
            (* word found, increment the count then move to front *)
            increment_MyList( ml, p ) ;
            moveToFront_MyList( ml, p ) ;
         end ;
         parse_MyString( aLine, aWord ) ;
      end {while length ...} ;
      i := readln_MyString( aLine ) ;
   end { while not MYEOF } ;

   (* Calculate and print results *)
   Statistics_MyList( ml, numWords, totalCount ) ;
   writeln(totalCount:5,' words in the file,') ;
   writeln(numWords:5,' unique words. They are:') ;
   print_MyList( ml ) ;
end .

----------------- SAMPLE OUTPUT -------------------

$ run driver4 /input=constitution.dat
$ type /page sys$output.dat
7760 words in the file,
 1250 unique words. They are:
  2 others
205 to
 40 on
  1 MESSAGE
 35 this
  3 pass
  4 liberty
  4 our
 16 is
  3 secure
 12 more
741 the
 34 Constitution
  3 know
 17 who
    ..............  et cetera
```

NOTE: CPU Time for driver4.pas on this file was 50.53 seconds. For driver2.pas, CPU Time was 2:50.38.