Programming Challenge: Three Card Flush

Task 1:

Demo:

```
CL-USER> ( demo--make-deck )
>>> Testing: make-deck
--- deck = ((2 . CLUB) (3 . CLUB) (4 . CLUB) (5 . CLUB) (6 . CLUB) (7 . CLUB)
        (8 . CLUB) (9 . CLUB) (10 . CLUB) (JACK . CLUB) (QUEEN . CLUB)
        (KING . CLUB) (ACE . CLUB) (2 . DIAMOND) (3 . DIAMOND)
        (4 . DIAMOND) (5 . DIAMOND) (6 . DIAMOND) (7 . DIAMOND)
        (8 . DIAMOND) (9 . DIAMOND) (10 . DIAMOND) (JACK . DIAMOND)
        (QUEEN . DIAMOND) (KING . DIAMOND) (ACE . DIAMOND) (2 . HEART)
        (3 . HEART) (4 . HEART) (5 . HEART) (6 . HEART) (7 . HEART)
        (8 . HEART) (9 . HEART) (10 . HEART) (JACK . HEART) (QUEEN . HEART)
        (KING . HEART) (ACE . HEART) (2 . SPADE) (3 . SPADE) (4 . SPADE)
        (5 . SPADE) (6 . SPADE) (7 . SPADE) (8 . SPADE) (9 . SPADE)
        (10 . SPADE) (JACK . SPADE) (QUEEN . SPADE) (KING . SPADE)
        (ACE . SPADE))
--- number of cards in deck = 52
NIL

CL-USER> ( demo--establish-shuffled-deck )
>>> Testing: shuffle-deck
--- *deck* ... ((3 . CLUB) (QUEEN . DIAMOND) (5 . DIAMOND) (QUEEN . HEART)
            (9 . SPADE) (ACE . SPADE) (6 . HEART) (KING . CLUB) (5 . CLUB)
            (8 . CLUB) (10 . SPADE) (3 . DIAMOND) (8 . SPADE) (4 . DIAMOND)
            (7 . CLUB) (KING . HEART) (7 . HEART) (4 . SPADE)
            (KING . SPADE) (7 . DIAMOND) (7 . SPADE) (JACK . CLUB)
            (2 . DIAMOND) (9 . HEART) (3 . SPADE) (3 . HEART)
            (10 . DIAMOND) (6 . SPADE) (8 . DIAMOND) (2 . SPADE) (6 . CLUB)
            (ACE . HEART) (JACK . HEART) (2 . CLUB) (ACE . CLUB) (9 . CLUB)
            (6 . DIAMOND) (5 . SPADE) (9 . DIAMOND) (4 . CLUB) (2 . HEART)
            (JACK . DIAMOND) (10 . CLUB) (10 . HEART) (KING . DIAMOND)
            (5 . HEART) (QUEEN . CLUB) (QUEEN . SPADE) (4 . HEART)
            (JACK . SPADE) (ACE . DIAMOND) (8 . HEART))
--- number of cards in *deck* = 52
NIL
```

Code:

```
( defun make-deck ()
  ( mapcan #'make-cards '( club diamond heart spade ) )
)

( defun make-cards ( suit &aux ranks )
  ( setf ranks '( 2 3 4 5 6 7 8 9 10 jack queen king ace ) )
  ( setf suit-duplicates ( duplicate ( length ranks ) suit ) )
  ( mapcar #'cons ranks suit-duplicates )
)

( defun demo--make-deck()
  ( format t ">>> Testing: make-deck~%" )
  ( setf deck ( make-deck ) )
  ( format t "--- deck = ~A~%" deck )
  ( format t "--- number of cards in deck = ~A~%" ( length deck ) )
  nil
)

( defun duplicate ( num l )
  ( cond
    ( ( = 1 num )
              ( list l )
          )
          ( t
            ( cons l ( duplicate ( - num 1 ) l ) )
          )
  )
)

( defun establish-shuffled-deck ()
  ( setf *deck* ( shuffle ( make-deck ) ) )
  nil
)

( defun shuffle ( deck )
  ( cond
    ( ( null deck )
            deck
          )
```

```
        ( t
          ( setf card ( nth ( random ( length deck ) ) deck ) )
          ( setf deck ( remove card deck ) )
          ( cons card ( shuffle deck ) )
        )
  )
)

( defun demo--establish-shuffled-deck ()
  ( format t ">>> Testing: shuffle-deck ~%" )
  ( establish-shuffled-deck )
  ( format t "--- *deck* ... ~A~%" *deck* )
  ( format t "--- number of cards in *deck* = ~A~%" ( length *deck* ) )
  nil
)
```

Task 2

Demo:

```
CL-USER> ( demo--deal-hands )
>>> Testing: deal-hands
--- *hand1* = ((JACK . HEART) (4 . SPADE) (9 . SPADE))
--- *hand2* = ((5 . CLUB) (ACE . CLUB) (5 . HEART))
--- number of cards in *deck* = 46
NIL
CL-USER> ( demo--deal-hands )
>>> Testing: deal-hands
--- *hand1* = ((KING . HEART) (8 . DIAMOND) (6 . DIAMOND))
--- *hand2* = ((QUEEN . DIAMOND) (4 . DIAMOND) (5 . CLUB))
--- number of cards in *deck* = 46
NIL
CL-USER>
```

```
CL-USER> ( demo--randomly-discard-cards )
--- *hand1* = ((7 . CLUB) (4 . CLUB) (9 . HEART))
--- *hand2* = ((8 . HEART) (9 . DIAMOND) (9 . CLUB))
--- *hand1* = ((7 . CLUB) NIL (9 . HEART))
--- *hand2* = ((8 . HEART) NIL (9 . CLUB))
NIL
CL-USER> ( demo--randomly-discard-cards )
--- *hand1* = ((4 . CLUB) (QUEEN . DIAMOND) (QUEEN . HEART))
--- *hand2* = ((2 . DIAMOND) (8 . HEART) (KING . CLUB))
--- *hand1* = (NIL (QUEEN . DIAMOND) (QUEEN . HEART))
--- *hand2* = (NIL (8 . HEART) (KING . CLUB))
NIL
CL-USER>
```

Code:

```
( defun deal-hands()
  ( establish-shuffled-deck )
  ( setf *hand1* () )
  ( setf *hand2* () )
```

```
  ( deal-card-to-hand1 )
  ( deal-card-to-hand2 )
  ( deal-card-to-hand1 )
  ( deal-card-to-hand2 )
  ( deal-card-to-hand1 )
  ( deal-card-to-hand2 )
  nil
)

( defun deal-card-to-hand1 ()
  ( setf card ( car *deck* ) )
  ( setf *deck* ( remove card *deck* ) )
  ( setf *hand1* ( append *hand1* ( list card ) ) )
)

( defun deal-card-to-hand2 ()
  ( setf card ( car *deck* ) )
  ( setf *deck* ( remove card *deck* ) )
  ( setf *hand2* ( append *hand2* ( list card ) ) )
)

( defun demo--deal-hands ()
  ( format t ">>> Testing: deal-hands~%" )
  ( deal-hands )
  ( format t "--- *hand1* = ~A~%" *hand1* )
  ( format t "--- *hand2* = ~A~%" *hand2* )
  ( format t "--- number of cards in *deck* = ~A~%" ( length *deck* ) )
  nil
)

( defun randomly-discard-cards ()
  ( randomly-discard-card-from-hand1 )
  ( randomly-discard-card-from-hand2 )
  nil
)

( defun randomly-discard-card-from-hand1 ()
  ( setf card-placement ( random ( length *hand1* ) ) )
  ( setf ( nth card-placement *hand1* ) () )
  nil
```

```
)

( defun randomly-discard-card-from-hand2 ()
  ( setf card-placement ( random ( length *hand2* ) ) )
  ( setf ( nth card-placement *hand2* ) () )
  nil
)


( defun demo--randomly-discard-cards ()
  ( deal-hands )
  ( format t "--- *hand1* = ~A~%" *hand1* )
  ( format t "--- *hand2* = ~A~&" *hand2* )
  ( randomly-discard-cards )
  ( format t "--- *hand1* = ~A~%" *hand1* )
  ( format t "--- *hand2* = ~A~%" *hand2* )
  nil
)
```

Task 3:

Demo:

```
CL-USER> (demo--replace-cards )
>>> Testing: replace-cards
--- *hand1* = ((4 . HEART) (8 . SPADE) (7 . CLUB))
--- *hand2* = ((10 . SPADE) (8 . HEART) (QUEEN . HEART))
--- *hand1* = (NIL (8 . SPADE) (7 . CLUB))
--- *hand2* = (NIL (8 . HEART) (QUEEN . HEART))
--- *hand1* = ((QUEEN . SPADE) (8 . SPADE) (7 . CLUB))
--- *hand2* = ((2 . SPADE) (8 . HEART) (QUEEN . HEART))
NIL
CL-USER> ( demo--replace-cards )
>>> Testing: replace-cards
--- *hand1* = ((JACK . CLUB) (QUEEN . CLUB) (KING . HEART))
--- *hand2* = ((2 . HEART) (5 . HEART) (9 . CLUB))
--- *hand1* = ((JACK . CLUB) (QUEEN . CLUB) NIL)
--- *hand2* = ((2 . HEART) (5 . HEART) NIL)
--- *hand1* = ((JACK . CLUB) (QUEEN . CLUB) (3 . CLUB))
--- *hand2* = ((2 . HEART) (5 . HEART) (ACE . CLUB))
NIL
CL-USER> ( demo--replace-cards )
>>> Testing: replace-cards
--- *hand1* = ((6 . CLUB) (2 . SPADE) (KING . CLUB))
--- *hand2* = ((9 . SPADE) (7 . SPADE) (ACE . HEART))
--- *hand1* = ((6 . CLUB) NIL (KING . CLUB))
--- *hand2* = (NIL (7 . SPADE) (ACE . HEART))
--- *hand1* = ((6 . CLUB) (ACE . CLUB) (KING . CLUB))
--- *hand2* = ((5 . CLUB) (7 . SPADE) (ACE . HEART))
NIL
```

```
CL-USER> ( demo--players-each-take-a-turn )
>>> Testing: players-each-take-a-turn
--- The hands ...
--- *hand1* = ((ACE . SPADE) (JACK . CLUB) (3 . DIAMOND))
--- *hand2* = ((2 . DIAMOND) (KING . HEART) (ACE . DIAMOND))
--- Each player takes a turn ...
```

```
--- *hand1* = ((ACE . SPADE) (4 . DIAMOND) (3 . DIAMOND))
--- *hand2* = ((5 . SPADE) (KING . HEART) (ACE . DIAMOND))
--- Each player takes a turn ...
--- *hand1* = ((ACE . SPADE) (8 . HEART) (3 . DIAMOND))
--- *hand2* = ((5 . SPADE) (2 . SPADE) (ACE . DIAMOND))
--- Each player takes a turn ...
--- *hand1* = ((ACE . SPADE) (8 . HEART) (6 . CLUB))
--- *hand2* = ((5 . SPADE) (2 . SPADE) (10 . SPADE))
--- Each player takes a turn ...
--- *hand1* = ((ACE . SPADE) (8 . HEART) (7 . DIAMOND))
--- *hand2* = ((10 . DIAMOND) (2 . SPADE) (10 . SPADE))
NIL
```

Code:

```
( defun replace-cards ()
  ( replace-card-in-hand1 )
  ( replace-card-in-hand2 )
  nil
)

( defun replace-card-in-hand1 ()
  ( setf first-card ( car *deck* ) )
  ( setf position-of-void ( position nil *hand1* ) )
  ( setf ( nth position-of-void *hand1* ) first-card )
  ( setf new-deck ( remove first-card *deck* :count 1 ) )
  ( setf *deck* new-deck )
  nil
)

( defun replace-card-in-hand2 ()
  ( setf first-card ( car *deck* ) )
  ( setf position-of-void ( position nil *hand2* ) )
  ( setf ( nth position-of-void *hand2* ) first-card )
  ( setf new-deck ( remove first-card *deck* :count 1 ) )
  ( setf *deck* new-deck )
  nil
)
```

```lisp
( defun demo--replace-cards ()
  ( format t ">>> Testing: replace-cards ~%" )
  ( deal-hands )
  ( format t "--- *hand1* = ~A~%" *hand1* )
  ( format t "--- *hand2* = ~A~%" *hand2* )
  ( randomly-discard-cards )
  ( format t "--- *hand1* = ~A~%" *hand1* )
  ( format t "--- *hand2* = ~A~%" *hand2* )
  ( replace-cards )
  ( format t "--- *hand1* = ~A~%" *hand1* )
  ( format t "--- *hand2* = ~A~%" *hand2* )
  nil
)

( defun players-each-take-a-turn ()
  ( randomly-discard-cards )
  ( replace-cards )
)

( defun demo--players-each-take-a-turn ()
  ( format t ">>> Testing: players-each-take-a-turn~%" )
  ( deal-hands )
  ( format t "--- The hands ...~%" )
  ( format t "--- *hand1* = ~A~%" *hand1* )
  ( format t "--- *hand2* = ~A~%" *hand2* )
  ( players-each-take-a-turn )
  ( format t "--- Each player takes a turn ...~%" )
  ( format t "--- *hand1* = ~A~%" *hand1* )
  ( format t "--- *hand2* = ~A~%" *hand2* )
  ( players-each-take-a-turn )
  ( format t "--- Each player takes a turn ...~%" )
  ( format t "--- *hand1* = ~A~%" *hand1* )
  ( format t "--- *hand2* = ~A~%" *hand2* )
  ( players-each-take-a-turn )
  ( format t "--- Each player takes a turn ...~%" )
  ( format t "--- *hand1* = ~A~%" *hand1* )
  ( format t "--- *hand2* = ~A~%" *hand2* )
  ( players-each-take-a-turn )
  ( format t "--- Each player takes a turn ...~%" )
  ( format t "--- *hand1* = ~A~%" *hand1* )
```

```
  ( format t "--- *hand2* = ~A~%" *hand2* )
  nil
)
```

Task 4:

Demo:

```
CL-USER> ( demo--flush-p )
>>> Testing: flush-p
((2 . CLUB) (ACE . CLUB) (10 . CLUB)) is a flush
((JACK . DIAMOND) (9 . DIAMOND) (5 . DIAMOND)) is a flush
((JACK . HEART) (10 . HEART) (9 . HEART)) is a flush
((2 . SPADE) (3 . SPADE) (ACE . SPADE)) is a flush
((10 . SPADE) (5 . DIAMOND) (ACE . SPADE)) is not a flush
((8 . CLUB) (9 . DIAMOND) (10 . HEART)) is not a flush
NIL
CL-USER> ( demo--high-card )
>>> Testing: high-card
(QUEEN . SPADE) is the high card of
 ((10 . HEART) (5 . CLUB) (QUEEN . SPADE) (7 . HEART))
(ACE . CLUB) is the high card of
 ((2 . DIAMOND) (2 . CLUB) (10 . HEART) (4 . DIAMOND) (ACE . CLUB))
(ACE . DIAMOND) is the high card of
 ((ACE . DIAMOND) (ACE . CLUB) (5 . SPADE))
NIL
CL-USER> ( demo-high-card )
{10037F2CE3}>.
CL-USER> ( demo--high-card )
>>> Testing: high-card
(QUEEN . SPADE) is the high card of
 ((10 . HEART) (5 . CLUB) (QUEEN . SPADE) (7 . HEART))
(ACE . CLUB) is the high card of
 ((2 . DIAMOND) (2 . CLUB) (10 . HEART) (4 . DIAMOND) (ACE . CLUB))
(ACE . DIAMOND) is the high card of
 ((ACE . DIAMOND) (ACE . CLUB) (5 . SPADE))
NIL
CL-USER> ( demo--straight-p )
>>> Testing: straight-p
((5 . SPADE) (3 . DIAMOND) (4 . SPADE) (6 . CLUB)) is a straight
((5 . SPADE) (7 . DIAMOND) (4 . SPADE) (8 . CLUB)) is not a straight
((KING . HEART) (QUEEN . DIAMOND) (ACE . SPADE) (10 . CLUB) (JACK .
DIAMOND)) is a straight
((ACE . CLUB) (2 . DIAMOND) (3 . SPADE)) is not a straight
```

```
NIL
CL-USER> ( demo--analyze-hand )
>>> Testing: analyze-hand
((5 . SPADE) (3 . DIAMOND) (4 . SPADE)) is a BUST
((5 . CLUB) (9 . CLUB) (4 . CLUB)) is a (9 HIGH CLUB FLUSH)
((QUEEN . HEART) (ACE . HEART) (KING . HEART)) is a (ACE HIGH HEART
STRAIGHT FLUSH)
```

Code:

```
( defun same-suit-p ( l )
  ( cond
    ( ( < ( length l ) 2 )
          t
        )
        ( ( eq ( car l ) ( cadr l ) )
          ( same-suit-p ( cdr l ) )
        )
        ( t
          nil
        )
  )
)

( defun flush-p ( hand &aux suits )
  ( setf suits ( mapcar #'cdr hand ) )
  ( same-suit-p suits )
)

( defun high-card ( hand )
  ( high-card-finder ( cdr hand ) ( car hand ) )
)

( defun high-card-finder ( hand high &aux values )
  ( setf values '( 2 3 4 5 6 7 8 9 10 jack queen king ace ) )
  ( cond
```

```
    ( ( = 0 ( length hand ) )
            high
          )
          ( ( < ( position ( car high ) values ) ( position ( car ( car hand ) ) values) )
            ( high-card-finder ( cdr hand ) ( car hand ) )
          )
          ( t
            ( high-card-finder ( cdr hand ) high )

          )
    )
)

( defun straight-p ( hand )
          ( setf hand ( sort-hand hand ) )
          ( check-straight hand )
)

( defun sort-hand ( hand &aux high )
  ( cond
    ( ( = ( length hand ) 1 )
            hand
          )
          ( t
            ( setf high ( high-card hand ) )
            ( setf hand ( remove high hand ) )
            ( snoc high ( sort-hand hand ) )
          )
  )
)

( defun check-straight ( hand &aux order order1 order2 difference )
  ( setf order '( 2 3 4 5 6 7 8 9 10 jack queen king ace ) )
  ( cond
    ( ( = ( length hand ) 1 )
          t
          )
          ( t
            ( setf order1 ( position ( car ( car hand ) ) order ) )
            ( setf order2 ( position ( car ( cadr hand ) ) order ) )
```

```
          ( setf difference ( - order2 order1 ) )
          ( cond
           ( ( = difference 1 )
                 ( check-straight ( cdr hand ) )
                )
                ( t
                  NIL
                )
            )
         )
    )
)

( defun analyze-hand ( hand &aux ret high )
  ( cond
    ( ( flush-p hand )
      ( setf high ( cons ( car ( high-card hand ) ) ' ( high ) ) )
           ( setf high ( snoc ( cdr ( car hand ) ) high ) )
           ( cond
            ( ( straight-p hand )
                  ( setf ret ( append high '(straight flush ) ) )
                )
                ( t
                  ( setf ret ( append high '( flush ) ) )
                )
     )
         )
         ( t
          ( setf ret 'bust )
         )
  )
  ret
)
```