Programming Challenge: M&C Interactive Problem Solver

Code:

```
( defun mc ()
        ( establish-world )
        ( init-move-list )
        ( make-moves )
)

( defun make-moves ()
  ( display-world )
  ( cond
    ( ( goalp )
      ( write-line "Good work!" )
          nil
        )
    ( ( feast-state-p )
          ( write-line "Yummy yummy yummy, I got Good in my tummy!!" )
          nil
        )
    ( t
          ( let ( m )
            ( format t ">>> " ) ( setf m ( read ) )
            ( if ( applicable-p m )
                  ( let () ( perform-move m ) ( make-moves ) )
                  ( let () ( write-line "Move inapplicable" ) nil )
                )
          )
        )
  )
)


( defun perform-move ( move )
  ( setf *move-list* ( snoc move *move-list* ) )
  ( if ( equal ( current-bank ) *left-bank* )
    ( move-lr move )
        ( move-rl move )
  )
)
```

```
( defun move-lr ( ml )
  ( if ( null ml ) ( return-from move-lr ) )
  ( move-lr-1 ( first ml ) )
  ( move-lr ( rest ml ) )
)


( defun move-rl ( ml )
  (if ( null ml ) ( return-from move-rl ) )
  ( move-rl-1 ( first ml ) )
  ( move-rl ( rest ml ) )
)

( defun establish-world ()
        ( setf *left-bank* '(M M M C C C B) )
        ( setf *right-bank* '() )
)

( defun init-move-list ()
        ( setf *move-list* '() )
)

( defun display-world ()
        ( format t "*left-bank* ~a~%" *left-bank* )
        ( format t "*right-bank* ~a~%" *right-bank* )
        ( format t "~a~%" *move-list* )
)

( defun goalp ()
        ( cond
                ( ( and ( =  ( count 'M *right-bank* ) 3 ) ( = ( count 'C *right-bank*) 3 ) )
                t
                )
                ( t
                        nil
                )
        )
)
```

```
( defun feast-state-p ()
  ( cond
        ( ( and ( > ( count 'C *left-bank* ) ( count 'M *left-bank* ) ) ( > ( count 'M
*left-bank* ) 0 ) )
          t
        )
        ( t
          nil
        )
        ( ( and ( > ( count 'C *right-bank* ) ( count 'M *right-bank* ) ) ( >  ( count 'M
*right-bank* ) 0 ) )
          t
        )
        ( t
          nil
        )
  )
)

( defun applicable-p ( m )
  ( cond
    ( ( and ( <= ( count 'M m ) ( count 'M ( current-bank ) ) )
            ( <= ( count 'C m ) ( count 'C ( current-bank ) ) )
                ( <= ( count 'B m ) ( count 'b ( current-bank ) ) )
                ( = ( count 'B m ) 1 )
                ( or ( > ( count 'M m ) 0 ) ( > ( count 'C m ) 0 ) )
          )
            t
        )
        ( t
          nil
        )
  )
)

( defun current-bank ()
  ( cond
    ( ( = ( count 'B *right-bank* ) 1 )
          *right-bank*
        )
```

```
        ( ( = ( count 'B *left-bank* ) 1 )
          *left-bank*
        )
  )
)

( defun snoc ( obj current-list )
  ( append current-list ( list obj ) )
)

( defun move-lr-1 ( ml )
  ( setf *left-bank* ( remove ml *left-bank* :count 1 ) )
  ( setf *right-bank* ( append ( list ml ) *right-bank* ) )
)

( defun move-rl-1 ( ml )
  ( setf *right-bank* ( remove ml *right-bank* :count 1 ) )
  ( setf *left-bank* ( append ( list ml ) *left-bank* ) )
)
```

Demo:

```
CL-USER> (mc)
*left-bank* (M M M C C C B)
*right-bank* NIL
NIL
>>> ( m b)
*left-bank* (M M C C C)
*right-bank* (B M)
((M B))
Yummy yummy yummy, I got Good in my tummy!!
NIL
CL-USER> (mc)
*left-bank* (M M M C C C B)
*right-bank* NIL
NIL
>>> (c c b)
*left-bank* (M M M C)
*right-bank* (B C C)
((C C B))
>>> (c c b)
*left-bank* (B C C M M M C)
*right-bank* NIL
((C C B) (C C B))
>>> (m c b)
*left-bank* (C M M C)
*right-bank* (B C M)
((C C B) (C C B) (M C B))
>>> ( c c b)
Move inapplicable
NIL
CL-USER> (mc)
*left-bank* (M M M C C C B)
*right-bank* NIL
NIL
>>> ( c c b)
*left-bank* (M M M C)
*right-bank* (B C C)
((C C B))
>>> (c b)
*left-bank* (B C M M M C)
```

right-bank (C)
((C C B) (C B))
>>> (c c b)
left-bank (M M M)
right-bank (B C C C)
((C C B) (C B) (C C B))
>>> (c b)
left-bank (B C M M M)
right-bank (C C)
((C C B) (C B) (C C B) (C B))
>>> (m m b)
left-bank (C M)
right-bank (B M M C C)
((C C B) (C B) (C C B) (C B) (M M B))
>>> (m c b)
left-bank (B C M C M)
right-bank (M C)
((C C B) (C B) (C C B) (C B) (M M B) (M C B))
>>> (m m b)
left-bank (C C)
right-bank (B M M M C)
((C C B) (C B) (C C B) (C B) (M M B) (M C B) (M M B))
>>> (c b)
left-bank (B C C C)
right-bank (M M M)
((C C B) (C B) (C C B) (C B) (M M B) (M C B) (M M B) (C B))
>>> (c c b)
left-bank (C)
right-bank (B C C M M M)
((C C B) (C B) (C C B) (C B) (M M B) (M C B) (M M B) (C B) (C C B))
>>> (c b)
left-bank (B C C)
right-bank (C M M M)
((C C B) (C B) (C C B) (C B) (M M B) (M C B) (M M B) (C B) (C C B) (C B))
>>> (c c b)
left-bank NIL
right-bank (B C C C M M M)
((C C B) (C B) (C C B) (C B) (M M B) (M C B) (M M B) (C B) (C C B) (C B)
 (C C B))
Good work!

NIL