4x4x4 Last Layer Wing Edges Solver

By: Christopher Mowla

What is the Purpose of this Program?

- This program only solves the last layer wing edges on a 4x4x4 cube, but also can be used to solve the last layer wing edges on an NxNxN one orbit at a time.
- This program solves the last layer wings only with 3-cycle algorithms and a 2-cycle algorithm, if the cube has odd parity. *Take note also that is uses the minimum number of 3-cycles of wings for any given case.*

Examples:

The last layer wings are numbered like so when they are solved:

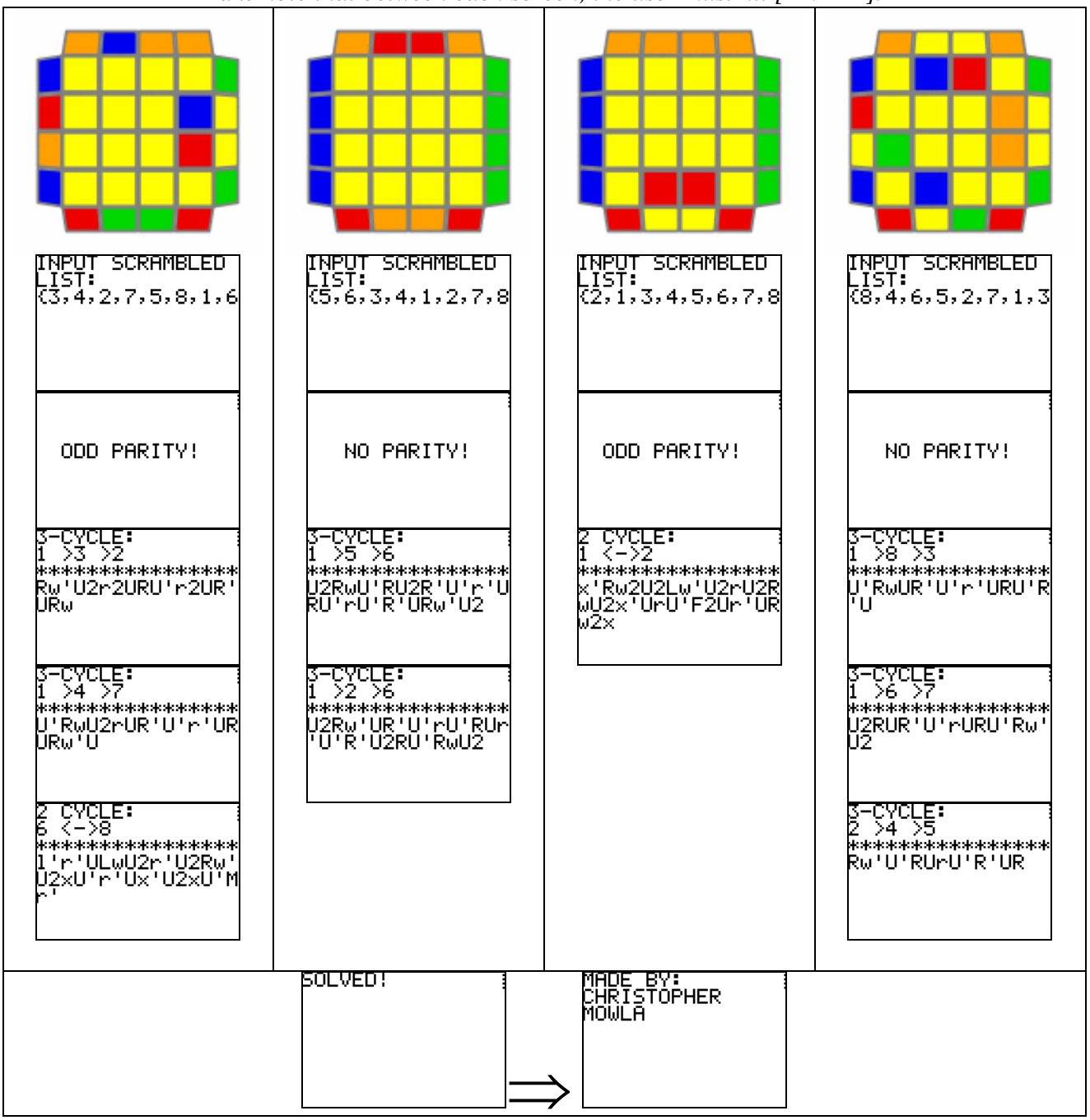
[6] [5]

[7] [4]

[8] [3]

[1] [2]

Take note that between each screen, the user must hit [ENTER].



The cube case images were created with visualcube, by Conrad Rider (Cride5).

Program Outline:

```
prgmA4X4SLVR
     prgmA2CYCLES
     prgmA2CYCMOV
          prgmLSTCUTM
               prgmLSTINSRP
          prgmLSTINSRP
     prgmA3CYCGRP
          prgmLSTINSRP
     prgmPERMTEST
     prgmAMOVES3
     prgmAMOVES2
```

The Programs:

This attachment consists of 10 programs:

prgmA4X4SLVR (Main/home program)

prgmA2CYCLES (Generates a 2-cycle list to solve the given scramble). Note that this program may be used to "solve" even lists with a large amount of elements with extreme efficiency.

prgmA2CYCMOV (Moves disjoint 2-cycles to the right of the 2-cycle list (for minimizing the number of required 3-cycles).

prgmA3CYCGRP Merges two non-disjoint 2-cycles as one 3-cycle and converts two disjoint 2-cycles into two disjoint 3-cycles.

prgmLSTCUTM A list subroutine which deletes a portion of consecutive arguments out of a list.

prgmLSTINSRP A list subroutine which inserts one list into another at the specified location in the list. It has the option to "insert" or "overright" arguments in the list to be inserted to.

prgmPERMTEST A brief subroutine to convert numbers 1-8 into strings "1" through "8".

prgmAMOVES3 The program which contains all algorithms for 3-cycle cases.

prgmAMOVES2 The program which contains all algorithms for 2-cycle cases.

prgmSCRAMBLE A program not integrated into the main program which scrambles a list with N arguments, either a full random scramble or a derangement scramble (no numbers are in their correct locations).

The Code: (With Comments Highlighted in Yellow)

```
PROGRAM A4X4SLVR:
This is the main program that calls all of the subroutines.

Clear the home screen
ClrHome
Prompt to input scrambled list, where, the numbers indicate specific wing
edges, and the slots in the list indicate the slots on the cube where they
are.  (For more information on how to input the correct list to represent
the last layer of your 4x4x4 cube, as well as show you examples, see the
readme document).
Disp "INPUT SCRAMBLED","LIST:"
Input "",L2
Clear the home screen
ClrHome
Call the subroutine to solve this list by swapping two numbers at a time.
It will yield a recording of the numbers swapped in a list L4.
prgmA2CYCLES
This program will locate disjoint 2-cycles in list L4 and move them to the
end of the list.  The resulting list will be list L4 (updated).
prgmA2CYCMOV
This program will merge all pairs of two non-disjoint 2-cycles as a 3-cycle
and convert two disjoint 2-cycles as two non-disjoint 3-cycles.  If there
is one 2-cycle left over (that is, the cube has odd parity), then it is
just left at the very end of the list.  The final list will be list L4
(updated again).
prgmA3CYCGRP
See the programs for more in-depth information about them.

Check to see if the cube has odd parity by seeing if the 3-cycle list is
evenly divisible by 4.  If it is, then there is no odd parity, otherwise,
there is.
If fPart(dim(L4)/4)≠0
Then
     Output(4,3,"ODD PARITY!
     Pause
Else
     Output(4,4,"NO PARITY!
End

Let Z be our counter for grabbing algorithms to represent the 3-cycles in
our final 3-cycle list L4.  Of course it starts out to be 1, since we start
from the beginning of the list.
1→Z
While we still have a 3-cycle left to link to an algorithm,
While (Z+3)≤dim(L4)
Take note that at least 4 arguments must be left in list L4. This is
because each 3-cycle in the list L4 is 4 arguments long: {(b,a,a,c)}, where
the 3-cycle is a->b->c.  This way to represent the 3-cycles was my personal
preference.  But the point is, we need 4 arguments to be left in list L4 if
we are to continue grabbing algorithms that correspond to them, before we
either:
   1) Terminate (that is, the cube is solved)
```

2) Move on to grab an algorithm which corresponds to the remaining 2-cycle (if there is one), that is, the cube has parity.

Clear the screen between each set of moves shown.

```
ClrHome
```

For fun, we can tell the user immediately whether or not the cube has an odd permutation (parity). We can tell simply by seeing if the 3-cycle list L_4 is evenly divisible by four (the number of arguments which represents each 3-cycle). If not, then there is one extra 2-cycle (and hence parity).

```
Output(1,1,"3 CYCLE:
```

Display the cycle in the form a>b>c (as mentioned above).

```
Disp "","
Output(2,1,L4(Z+1
Output(2,3,">
Output(2,4,L4(Z
Output(2,6,">
Output(2,7,L4(Z+3
```

Let the user know that it is going to take a few seconds to retrieve the corresponding algorithm from the 3-cycle table (prgmAMOVES3).

```
Output(3,1,"PLEASE WAIT...
```

We now define a string Str1. We are going to convert the following arguments of list L_4 to strings and add each onto string Str1 to have a string of the form: "ABC", where A<B and A<C.

```
" →Str1
```

The subroutine we are going to use takes an integer X, whose values are 1-8 and converts them to "1"-"8" strings. It adds each onto string Str1 each time.

```
L4(Z+1)→X
prgmPERMTEST
L4(Z)→X
prgmPERMTEST
L4(Z+3)→X
prgmPERMTEST
```

Now we delete the first space of string Str1 (we had to have its length be non-zero to add other strings to it).

```
sub(Str1,2,length(Str1)-1)→Str1
```

Now we access the 3-cycle move table subroutine to retrieve the corresponding algorithm as string Str0.

```
prgmAMOVES3
```

Output the string (moves) to the user below the location where we showed what 3-cyle of pieces we are doing with these moves.

```
Output(3,1,"****************
Output(4,1,Str0
Pause
Z+4→Z
End
```

Now the 3-cycle algorithms have been shown, and now either of two things will happen when the user hits the enter key:

1) The cube will be solved and therefore we can skip the code below by the condition.
2) Access the corresponding 2-cycle needed to solve the cube (odd parity fix).

If our counter Z is not equal to Z at this point, but less than it, we obvious must have more numbers in the 3-cycle list L_4, but obviously less

```
than 4.  So, because list L4 has an even number of numbers, we obviously
have a 2-cycle we need to take care of.
If Z<dim(L4)
Then
Clear the home screen
      ClrHome
      Output(2,1,"           "
      Output to the user the 2-cycle which we will retrieve the
      corresponding algorithm for.  Also, for consistency on displaying the
      2-cycle, we will always display the lower number to the left.  To save
      code, we can just use the sort ascending list function.
      Output(1,1,"2 CYCLE:
      {L4(Z+1),L4(Z)}→L3
      SortA(L3
      Disp "","
      Output(2,1,L3(1
      Output(2,3,"<->
      Output(2,6,L3(2
      Let the user know that it is going to take a few seconds to retrieve
      the algorithm to finish solving the cube.
      Output(3,1,"PLEASE WAIT...
      We do the same process as we did for 3-cycles to link the
      corresponding algorithm to the cycle by use of strings.
      " →Str1
      L3(1)→X
      prgmPERMTEST
      L3(2)→X
      prgmPERMTEST

      sub(Str1,2,length(Str1)-1)→Str1
      prgmAMOVES2
      We have the corresponding algorithm know in string Str0.  Let's
      display it to the user under the location where we showed the cycle we
      are doing with these moves.
      Output(3,1,"****************
      Output(4,1,Str0
      Pause
End

Now all conditions have been satisfied.  The program will exit.  But first,
why not let them know that the cube should be solved, as well as give
credit where it is due.
ClrHome
Pause "SOLVED!
ClrHome
Disp "MADE BY:","CHRISTOPHER","MOWLA"
Pause
ClrHome
```

PROGRAM A2CYCLES:

```
If any elements in list L2 are equal to their corresponding slot numbers,
then make that argument zero as a preliminary marker.
For(A,1,dim(L2),1)
      If A=L2(A)
            Ø→L2(A)
End
Let the 2-cycle list to be formed be L4.
Ø→dim(L4)
Let Z be a counter in the following loop.
1→Z
We can start an infinite loop condition so that the loop will terminate
only when a condition within becomes false.
While 1
      Since we could have possibly made the first argument in list L2 zero
      (if the piece is already solved in that slot), we need to increase our
      starting point until we get to the first unsolved slot.
      While L2(Z)=Ø
            Z+1→Z
      End
      Add two arguments to the right of the forming 2-cycle list L4 in each
      iteration.
      2+dim(L4)→dim(L4)
      We swap the number in slot Z of L2, that is, L2(Z) with the number
      that is in the L2(Z)^th argument of list L2,  L2(L2(Z)).
      L2(Z)→B
      L2(L2(Z))→A
      Now, it is always true that the location B is put in is its
      corresponding slot number.  So let us simply store it to be zero (to
      mark it 'solved').
      Ø→L2(L2(Z
      But for the slot L2(Z) where A is placed, it may or may not be the
      corresponding slot for letter A (.e.g. "solved").
      If it is the corresponding slot (if it is "solved"), let's mark it as
      a zero.
      If A=Z:Then
            Ø→L2(Z)
            Otherwise, we just store the value of A there, so that it is
            involved in a 2-cycle swap later.
      Else
            A→L2(Z)
      End
      Now, by the algorithm for 2-cycle formation, each 2-cycle pair is of
      the following
      form: (B, Z)  or  (The piece that was solved back,the location it was in before the swap).
      B→L4(dim(L4)-1)
      Z→L4(dim(L4))
      The terminating condition is when all slots have been "solved", that
      is, when all arguments of list L2 are zero.
      If sum(L2)=Ø
            Return
End
```

PROGRAM A2CYCMOV:

This program compares all 2-cycles and sees whether or not a 2-cycle is independent from the rest (disjoint). If it is, it is moved to the end (far right) of list L_4 so that as many dependent pairs can be next to each other as possible (by this, prgmA3CYCGRP will generate the least amount of 3-cycles as possible for the given scramble).

$$\{(p_i, s_j), (p_{i+1}, s_{j+1})\}$$

Let X represent the argument in list L_4 which contains the first element of the four being compared (p_i).

```
1→X
```

Let J represent the number of 2-cycles (which have two elements each, obviously) that have been compared. Then the length of list L_4 divided by two is the number of comparisons necessary to compare all 2-cycles in L_4.

```
For(J,1,dim(L4)/2,1)
```

If there are no longer two 2-cycle pairs left to be compared, exit this subroutine.

```
If (X+3)>dim(L4)
    Return
```

The above scenario will occur if there are an odd number of 2-cycles that solve the scramble/key.

If there is an even number of 2-cycles, then the condition of the loop suffices.

If the 2nd and 4th numbers are the same (i.e. $s_j = s_{j+1}$), then we have a dependent pair. Hence we do not need to move either of these 2-cycles to the end of list L_4.

```
If L4(X+1)=L4(X+3)
Then
```

We move on to the next 4 arguments in L_4.

```
    X+4→X
```

In addition to the regular counter in the for-loop condition, we must add an additional one to J since we have analyzed two 2-cycles instead of one.

```
    J+1→J
Else
```

If the 2nd and 4th numbers are NOT the same (i.e. $s_j \neq s_{j+1}$), the 2-cycles (p_i, s_j) and (p_{i+1}, s_{j+1}) are disjoint 2-cycles. We will move the first 2-cycle (p_i, s_j) to the end of the list, with the hope that the remaining one will not be disjoint with the next 2-cycle.

But if the current two 2-cycles being compared are the last two, then there is no reason to move the first one to the right of the second, as they will still be independent. Hence, exit the subroutine.

```
    If J=dim(L4)/2
        Return
```

```
        Since that condition is taken care of, we can go ahead and apply
        our algorithm.

        Store the first 2-cycle as a temporary list ᴸL6.
        {L4(X),L4(X+1)}→ᴸL6
        We delete the first 2-cycle (pi,sj) (of the two currently being
        compared) from the list L4 using a subroutine.

        Let A be the slot location in list L4 to begin deleting and B be
        the last slot in list L4 to delete.
        X→A:X+1→B
        We store list L4 as a temporary list ᴸL1 for the subroutine to
        edit:
        L4→ᴸL1
        Execute the subroutine
        prgmLSTCUTM
        Replace list L4 with the new adjusted list (which does not
        contain the 2-cycle (pi,sj)).
        ᴸL5→L4

        Now we will take the 2-cycle (pi,sj) (stored as list ᴸL6) and
        insert it into list L4 with another subroutine.

        We let the slot location in list L4 which is to the left of where
        we need to insert the 2-cycle to be A (which is a new slot to the
        right of the last argument current in list L4).
        dim(L4)+1→A
        By the way the subroutine works, to insert arguments into a list
        without replacing them, the terminating location Z needs to be
        less than the beginning location A.  Hence, we can let Z = A-1.
        A-1→Z
        By the subroutine, the list piece being inserted is list ᴸL2, so
        let's store it as list ᴸL6 (our 2-cycle (pi,sj) ).
        ᴸL6→ᴸL2
        The list which the subroutine inserts a list ᴸL2 into is list
        ᴸL1.  So, let's make it our developing 3-cycle list L4.
        L4→ᴸL1
        Execute subroutine
        prgmLSTINSRP
        Now store the resulting list ᴸL5 as our developing 3-cycle list,
        L4.
        ᴸL5→L4
    End
End
```

PROGRAM A3CYCGRP:

Let C be p_i in the current two 2-cycles being merged $\{(p_i, s_j), (p_{i+1}, s_{j+1})\}$.

1→C

While there is are least two 2-cycles left to be merged/converted to 3-cycles,

While (C+3)≤dim(L4)

Let us store s_j as a temporary variable U.

L4(C+1)→U

and Let us store p_{i+1} as a temporary variable V.

L4(C+2)→V

and Let us store s_{j+1} as a temporary variable W.

L4(C+3)→W

If $s_j = s_{j+1}$,

If U=W:Then

Make this $\{(p_i, s_j), (p_{i+1}, s_j)\}$ into this $\{(p_i, s_j, s_j, p_{i+1})\}$.

V→L4(C+3

W→L4(C+2

And we move on to the next four numbers in list L4.

C+4→C

Otherwise, we must

1) Convert them to $\{(p_i, s_j, s_j, p_{i+1}), (s_{j+1}, s_j, s_j, p_{i+1})\}$.

2) Replace $\{(p_i, s_j), (p_{i+1}, s_j)\}$ with it.

Else

Define a new list

∅→dim(L5

Make it have 8 arguments

dim(L5)+8→dim(L5

*The following code, before we convert these disjoint 2 2-cycles into 2 3-cycles, is just to rewrite the permutation representation $\{(p_i, s_j), (p_{i+1}, s_{j+1})\}$ so that, when the 3-cycles are made, the smallest number is the left number (because 3-cycles can be represented 3 different ways, and we don't want to have an algorithm data program be 3 times the size it needs to be). Of course, this does not happen every time, so we have a condition. *The reason we do not need to do this for non-disjoint 2-cycles is because by the algorithm of prgmA2CYCYLES, the number Z in the 2-cycle* (B, Z) *is always going to be less than B. However, when we reorder the 2-cycles by moving one 2-cycle to the end, this may not be true anymore.*

If $s_j > s_{j+1}$, then switch the order of the two 2-cycles: $\{(p_{i+1}, s_{j+1}), (p_i, s_j)\}$.

If L4(C+1)>L4(C+3)

Then

```
        {L4(C),L4(C+1),L4(C+2),L4(C+3)}→LTEMP
        LTEMP(1)→L4(C+2)
        LTEMP(2)→L4(C+3)
        LTEMP(3)→L4(C
        LTEMP(4)→L4(C+1
    End
```

Now we Convert them to $\{(P_i,S_j,S_j,P_{i+1}),(S_{j+1},S_j,S_j,P_{i+1})\}$, where the P_i maybe equal to p_{i+1}, etc., from the code above.

```
L4(C→L5(1
L4(C+1→L5(2
L4(C+1→L5(3
L4(C+2→L5(4
L4(C+3→L5(5
L4(C+1→L5(6
L4(C+1→L5(7
L4(C+2→L5(8
```

By the subroutine, list LL2 is inserted into list LL1. If the value of Z is less than A, the subroutine will insert. If not, then the subroutine will both insert and overwrite arguments from A to Z.

Hence, we can start out by letting list L5 be our insertion list and list LL1 be our currently developing 3-cycle list L4.

```
L5→LL2
L4→LL1
```

We store the current location in the main list of 3-cycles as the starting point to insert list L5 of 8 arguments (2 3-cycles) in the place of the original location of the pair of 2 2-cycles (4 arguments in the developing 3-cycle list L4).

By the subroutine, this location is A.

```
C→A
```

The end location of arguments to replace is obviously just after 3 more slots where we begin to replace.

```
C+3→Z
```

Execute the list insert/replace subroutine:

```
prgmLSTINSRP
```

We replace our old 3-cycle list as the new 3-cycle list generated by the subroutine .

```
LL5→L4
```

Since we ultimately inserted 4 additional arguments into list L4 (our 3-cycle list), we need to add 4 to the counter C. In addition, we add another 4 to indicate we have analyzed 4 more spaces in the list (just as we add 4 if we do not have two disjoint 2-cycles in the developing 3-cycle list L4).

Since we inserted 4 extra elements into list L4, and we wish to indicate that we have analyzed this section of 4 arguments, we add a total of 8 to our counter, C.

```
        C+8→C
    End
End
```

PROGRAM LSTINSRP:

This subroutine can both insert a list ʟL2 into list ʟL1, and it can insert list ʟL2 into list ʟL1 + overwrite any amount of list ʟL1 (the user must specify how much, if any).

A is the starting location and Z is the ending locations of where in list ʟL1 that we wish to insert list ʟL2.

[Step 1] The left portion of list ʟL1 needs to be stored as another list ʟL5 (which is also the start of the final list).

```
Ø→dim(ʟL5
For(X,1,A-1,1)
     1+dim(ʟL5→dim(ʟL5
     ʟL1(X)→ʟL5(X
End
```

[Step 2] The right half of the list subtracted from the ending increment Z will be stored as another list ʟL4.

```
Z-dim(ʟL5)+1→B
Ø→dim(ʟL4)
For(X,1,dim(ʟL1)-Z,1)
     1+dim(ʟL4)→dim(ʟL4)
     ʟL1(dim(ʟL5)+B→ʟL4(X)
     B+1→B
End
```

For memory purposes, let's delete list ʟL1, since we don't need it anymore.

```
DelVar ʟL1
```

[Step 3] Now we add on what we wanted to insert, ʟL2, to our final list ʟL5.

```
dim(ʟL5)→D
For(X,1,dim(ʟL2),1)
     1+dim(ʟL5)→dim(ʟL5)
     D+1→D
     ʟL2(X)→ʟL5(D)
End
```

For memory purposes, let's delete list ʟL2, since we don't need it anymore.

```
DelVar ʟL2
```

[Step 4] Finally, we add on the right half we previously stored in list ʟL4, on to the final list ʟL5.

```
dim(ʟL5→D
For(X,1,dim(ʟL4
     D+1→D
     1+dim(ʟL5→dim(ʟL5
     ʟL4(X→ʟL5(D
End
```

For memory purposes, let's delete list ʟL4, since we don't need it anymore.

```
DelVar ʟL4
```

PROGRAM LSTCUTM:

This subroutine takes in a list ₗL1 and has a portion of it, ₗL1(A) to ₗL1(B), cut out from it.

[Step 1] We store all of ₗL1 which is to the left of what we want to cut as a temporary list ₗL2.

```
Ø→dim(ₗL2
Ø→dim(ₗL3
For(Z,1,A-1,1)
     dim(ₗL2)+1→dim(ₗL2
     ₗL1(Z)→ₗL2(Z)
End
```

[Step 2] Now we store all of list ₗL1 which is to the right of where we want to cut as temporary list ₗL3.

```
1→G
For(Z,B+1,dim(ₗL1),1)
     dim(ₗL3)+1→dim(ₗL3
     ₗL1(Z)→ₗL3(G)
     G+1→G
End
```

[Step 3] Finally, we add the right half (list ₗL3) to the left half (ₗL2). To save code, instead of writing a loop here, we can instead just use the subroutine LSTINSRP to add the lists.

In that subroutine, the main list is ₗL1 and the list being inserted is list ₗL2. So we store the left and right halves as these two lists respectively.

```
ₗL2→ₗL1
ₗL3→ₗL2
```

To insert at the location one argument larger than a given list can be interpreted as adding two separate lists together. So, let the location we insert list ₗL2 into list ₗL1 be the dimension of list ₗL1 plus one.

```
dim(ₗL1)+1→A
```

As mentioned before in program A2CYCMOV (on its second page), if we want to use this subroutine to insert and not overwrite, we must store Z as a lesser integer value than A :let it be one less.

```
A-1→Z
```

Accessing the previous subroutine, we are in business.

For memory purposes, let's delete list ₗL3, since we don't need it anymore.

```
DelVar ₗL3

prgmLSTINSRP
```

PROGRAM PERMTEST:

This program is just a subroutine to the main program which takes the current value of the variable X and adds that into the string Str1, which is ultimately used to request algorithms for corresponding 3-cycles and/or 2-cycles.

```
If X=1
Str1+"1→Str1
If X=2
Str1+"2→Str1
If X=3
Str1+"3→Str1
If X=4
Str1+"4→Str1
If X=5
Str1+"5→Str1
If X=6
Str1+"6→Str1
If X=7
Str1+"7→Str1
If X=8
Str1+"8→Str1
```

PROGRAM AMOVES2:

This program consists of a search function to retrieve the corresponding 2-cycle odd parity algorithm for a 2-cycle list L_3 of the form {a,b}, where a<b. By having a always less than b (by using the sort ascending list function in the main program prior to displaying it to the user), we save 50% space in this program because we can represent the same 2-cycle in 2 different ways (and we would have to have 2 searches for the same algorithm instead of one). *Guaranteeing this space reduction for 3-cycles is not so easy, but we have already done it (see the comments for program pgrmAMOVES3 for more information).*

```
If Str1="12
"x'Rw2U2Lw'U2rU2RwU2x'UrU'F2Ur'URw2x
If Str1="34
"Ux'Rw2U2Lw'U2rU2RwU2x'UrU'F2Ur'URw2xU'
If Str1="56
"U2x'Rw2U2Lw'U2rU2RwU2x'UrU'F2Ur'URw2xU2
If Str1="78
"U'x'Rw2U2Lw'U2rU2RwU2x'UrU'F2Ur'URw2xU

If Str1="25
"U2rU2rU2M'U2rU2r'U2lU2r2U2
If Str1="47
"U'rU2rU2M'U2rU2r'U2lU2r2U
If Str1="16
"rU2rU2M'U2rU2r'U2lU2r2
If Str1="38
"UrU2rU2M'U2rU2r'U2lU2r2U'

If Str1="23
"Ul'2ULwU2r'U2Rw'U2xU'r'Ux'U2xU'M'l'U'
If Str1="45
"U2l'2ULwU2r'U2Rw'U2xU'r'Ux'U2xU'M'l'U2
If Str1="67
"U'l'2ULwU2r'U2Rw'U2xU'r'Ux'U2xU'M'l'U
If Str1="18
"l'2ULwU2r'U2Rw'U2xU'r'Ux'U2xU'M'l'

If Str1="24
"U2l'r'ULwU2r'U2Rw'U2xU'r'Ux'U2xU'Mr'U2
If Str1="46
"U'l'r'ULwU2r'U2Rw'U2xU'r'Ux'U2xU'Mr'U
If Str1="68
"l'r'ULwU2r'U2Rw'U2xU'r'Ux'U2xU'Mr'
If Str1="28
"Ul'r'ULwU2r'U2Rw'U2xU'r'Ux'U2xU'Mr'U'

If Str1="13
"U'rlU'Rw'U2lU2LwU2xUlU'x'U2xUMlU
If Str1="35
"rlU'Rw'U2lU2LwU2xUlU'x'U2xUMl
If Str1="57
"UrlU'Rw'U2lU2LwU2xUlU'x'U2xUMlU'
If Str1="17
```

```
"U2rlU'Rw'U2lU2LwU2xUlU'x'U2xUMlU2

If Str1="27
"U'l'2U'LwU2r'U2Rw'U2xUr'U'x'U2xUM'l'U
If Str1="14
"l'2U'LwU2r'U2Rw'U2xUr'U'x'U2xUM'l'
If Str1="36
"Ul'2U'LwU2r'U2Rw'U2xUr'U'x'U2xUM'l'U'
If Str1="58
"U2l'2U'LwU2r'U2Rw'U2xUr'U'x'U2xUM'l'U2


If Str1="15
"rU2r'U2r'U2lU2r'U2rU2M'U2r2U2r'
If Str1="37
"U'rU2r'U2r'U2lU2r'U2rU2M'U2r2U2r'UIf Str1="26
If Str1="26
"l'U2lU2lU2r'U2lU2l'U2M'U2l'2U2l
If Str1="48
"U'l'U2lU2lU2r'U2lU2l'U2M'U2l'2U2lU

The algorithm corresponding to the needed 2-cycle is stored as string Str0.
Ans→Str0
```

PROGRAM AMOVES3:

This program consists of a search function to retrieve the corresponding 3-cycle algorithms for a 3-cycle list L_3 of the form {a,b,c}, where a<b and a<c (not necessarily b<c). By having a<b and a<c, we save 66% space in this program because we can represent the same 3-cycle in 3 different ways (and we would have to have 3 searches for the same algorithm instead of one). How program prgmA3CYCGRP formed the 3-cycle list, this is guaranteed, even with optimizing the number of 3-cycles need to solve a given scramble. (To see where it guarantees this, find the * in the comments in that program).

```
"→Str0

If Str1="186
"U2R'U'RUr'U'R'URwU2
If Str1="283
"U'R'U'RUr'U'R'URwU
If Str1="254
"R'U'RUr'U'R'URw
If Str1="476
"UR'U'RUr'U'R'URwU'

If Str1="168
"U2Rw'U'RUrU'R'URU2
If Str1="238
"U'Rw'U'RUrU'R'URU
If Str1="245
"Rw'U'RUrU'R'UR
If Str1="467
"URw'U'RUrU'R'URU'

If Str1="253
"RwUR'U'r'URU'R'
If Str1="475
"URwUR'U'r'URU'R'U'
If Str1="176
"U2RwUR'U'r'URU'R'U2
If Str1="183
"U'RwUR'U'r'URU'R'U

If Str1="235
"RUR'U'rURU'Rw'
If Str1="457
"URUR'U'rURU'Rw'U'
If Str1="167
"U2RUR'U'rURU'Rw'U2
If Str1="138
"U'RUR'U'rURU'Rw'U

If Str1="163
"RwU'R'U'rURU'r'U2Rw'
If Str1="385
"URwU'R'U'rURU'r'U2Rw'U'
If Str1="275
```

```
"U2RwU'R'U'rURU'r'U2Rw'U2
If Str1="174
"U'RwU'R'U'rURU'r'U2Rw'U

If Str1="136
"RwU2rUR'U'r'URURw'
If Str1="358
"URwU2rUR'U'r'URURw'U'
If Str1="257
"U2RwU2rUR'U'r'URURw'U2
If Str1="147
"U'RwU2rUR'U'r'URURw'U

If Str1="285
"U2Rw'U2r'U'RUrU'R'U'RwU2
If Str1="274
"U'Rw'U2r'U'RUrU'R'U'RwU
If Str1="164
"Rw'U2r'U'RUrU'R'U'Rw
If Str1="386
"URw'U2r'U'RUrU'R'U'RwU'

If Str1="258
"U2Rw'URUr'U'R'UrU2RwU2
If Str1="247
"U'Rw'URUr'U'R'UrU2RwU
If Str1="146
"Rw'URUr'U'R'UrU2Rw
If Str1="368
"URw'URUr'U'R'UrU2RwU'

If Str1="185
"U'Rw'Ur2URU'r2UR'U2RwU
If Str1="273
"Rw'Ur2URU'r2UR'U2Rw
If Str1="154
"URw'Ur2URU'r2UR'U2RwU'
If Str1="376
"U2Rw'Ur2URU'r2UR'U2RwU2

If Str1="158
"U'Rw'U2RU'r2UR'U'r2U'RwU
If Str1="237
"Rw'U2RU'r2UR'U'r2U'Rw
If Str1="145
"URw'U2RU'r2UR'U'r2U'RwU'
If Str1="367
"U2Rw'U2RU'r2UR'U'r2U'RwU2

If Str1="263
"U'RwU2R'Ur2U'RUr2URw'U
If Str1="485
"RwU2R'Ur2U'RUr2URw'
If Str1="276
"URwU2R'Ur2U'RUr2URw'U'
If Str1="184
```

```
"U2RwU2R'Ur2U'RUr2URw'U2

If Str1="236
"U'RwU'r2U'R'Ur2U'RU2Rw'U
If Str1="458
"RwU'r2U'R'Ur2U'RU2Rw'
If Str1="267
"URwU'r2U'R'Ur2U'RU2Rw'U'
If Str1="148
"U2RwU'r2U'R'Ur2U'RU2Rw'U2

If Str1="175
"U2RwU'R'U'r'URU'rU2Rw'U2
If Str1="173
"U'RwU'R'U'r'URU'rU2Rw'U
If Str1="153
"RwU'R'U'r'URU'rU2Rw'
If Str1="375
"URwU'R'U'r'URU'rU2Rw'U'

If Str1="157
"U2RwU2r'UR'U'rURURw'U2
If Str1="137
"U'RwU2r'UR'U'rURURw'U
If Str1="135
"RwU2r'UR'U'rURURw'
If Str1="357
"URwU2r'UR'U'rURURw'U'

If Str1="264
"Rw'U2rU'RUr'U'R'U'Rw
If Str1="486
"URw'U2rU'RUr'U'R'U'RwU'
If Str1="286
"U2Rw'U2rU'RUr'U'R'U'RwU2
If Str1="284
"U'Rw'U2rU'RUr'U'R'U'RwU

If Str1="246
"Rw'URUrU'R'Ur'U2Rw
If Str1="468
"URw'URUrU'R'Ur'U2RwU'
If Str1="268
"U2Rw'URUrU'R'Ur'U2RwU2
If Str1="248
"U'Rw'URUrU'R'Ur'U2RwU

If Str1="165
"U2RwU'RUr'UR'U'rURU2R'URw'U2
If Str1="387
"U'RwU'RUr'UR'U'rURU2R'URw'U
If Str1="152
"RwU'RUr'UR'U'rURU2R'URw'
If Str1="374
"URwU'RUr'UR'U'rURU2R'URw'U'
```

```
If Str1="156
"U2RwU'RU2R'U'r'URU'rU'R'URw'U2
If Str1="378
"U'RwU'RU2R'U'r'URU'rU'R'URw'U
If Str1="125
"RwU'RU2R'U'r'URU'rU'R'URw'
If Str1="347
"URwU'RU2R'U'r'URU'rU'R'URw'U'

If Str1="265
"Rw'UR'U2RUrU'R'Ur'URU'Rw
If Str1="487
"URw'UR'U2RUrU'R'Ur'URU'RwU'
If Str1="162
"U2Rw'UR'U2RUrU'R'Ur'URU'RwU2
If Str1="384
"U'Rw'UR'U2RUrU'R'Ur'URU'RwU

If Str1="256
"Rw'UR'U'rU'RUr'U'R'U2RU'Rw
If Str1="478
"URw'UR'U'rU'RUr'U'R'U2RU'RwU'
If Str1="126
"U2Rw'UR'U'rU'RUr'U'R'U2RU'RwU2
If Str1="348
"U'Rw'UR'U'rU'RUr'U'R'U2RU'RwU

If Str1="182
"U2RwUR'Ur2U'RUr2U2Rw'U2
If Str1="243
"U'RwUR'Ur2U'RUr2U2Rw'U
If Str1="465
"RwUR'Ur2U'RUr2U2Rw'
If Str1="687
"URwUR'Ur2U'RUr2U2Rw'U'

If Str1="128
"U2RwU2r2U'R'Ur2U'RU'Rw'U2
If Str1="234
"U'RwU2r2U'R'Ur2U'RU'Rw'U
If Str1="456
"RwU2r2U'R'Ur2U'RU'Rw'
If Str1="678
"URwU2r2U'R'Ur2U'RU'Rw'U'

If Str1="132
"Rw'U2r2URU'r2UR'URw
If Str1="354
"URw'U2r2URU'r2UR'URwU'
If Str1="576
"U2Rw'U2r2URU'r2UR'URwU2
If Str1="187
"U'Rw'U2r2URU'r2UR'URwU

If Str1="123
"Rw'U'RU'r2UR'U'r2U2Rw
```

```
If Str1="345
"URw'U'RU'r2UR'U'r2U2RwU'
If Str1="567
"U2Rw'U'RU'r2UR'U'r2U2RwU2
If Str1="178
"U'Rw'U'RU'r2UR'U'r2U2RwU

If Str1="172
"U2RwU2rUR'U'r2URU'rU2Rw'U2
If Str1="143
"U'RwU2rUR'U'r2URU'rU2Rw'U
If Str1="365
"RwU2rUR'U'r2URU'rU2Rw'
If Str1="587
"URwU2rUR'U'r2URU'rU2Rw'U'

If Str1="127
"U2RwU2r'UR'U'r2URU'r'U2Rw'U2
If Str1="134
"U'RwU2r'UR'U'r2URU'r'U2Rw'U
If Str1="356
"RwU2r'UR'U'r2URU'r'U2Rw'
If Str1="578
"URwU2r'UR'U'r2URU'r'U2Rw'U'

If Str1="142
"Rw'U2rU'RUr2U'R'UrU2Rw
If Str1="364
"URw'U2rU'RUr2U'R'UrU2RwU'
If Str1="586
"U2Rw'U2rU'RUr2U'R'UrU2RwU2
If Str1="287
"U'Rw'U2rU'RUr2U'R'UrU2RwU

If Str1="124
"Rw'U2r'U'RUr2U'R'Ur'U2Rw
If Str1="346
"URw'U2r'U'RUr2U'R'Ur'U2RwU'
If Str1="568
"U2Rw'U2r'U'RUr2U'R'Ur'U2RwU2
If Str1="278
"U'Rw'U2r'U'RUr2U'R'Ur'U2RwU

The algorithm corresponding to the needed 3-cycle is stored as string StrØ.
Ans→StrØ
```

PROGRAM SCRAMBLE:

I have included a scramble program as well, to generate random scrambles of a list of a length of your choice (well, as much as the calculator can handle).

There are two items required for input:

1) The value of N. It is the length of the list you want a random scramble from. For example, N = 8 for just the last layer wing edges of the 4x4x4, but you can choose a larger number such as 24 (total wing edges on a 4x4x4), if you liked to retrieve the optimal sequence to solve all wings on a scrambled 4x4x4. Unfortunately, I don't have the space for algorithm tables in the mini memory of the Ti-83/84p calculators, even if I were to expand this program to solve all wing edges on a 4x4x4 (or even expand that to all X-center pieces).
2) If you want a scramble which does not have any pieces correct, let I=Ø. Otherwise, you can let I be any real value you wish.

This program is probably useful to generate last layer scrambles by doing the inverse (the move to correct the scramble by the solver program) to your cube.

(Form a list of the form $\{1,2,3,...,N\}$).

Initialize list L_2 to have zero dimensions.
Ø→dim(L_2)
Since the length of string StrØ was stored as N, then from 1 to N
For(A,1,N,1)
Add one dimension to list L_2.
1+dim(L_2)→dim(L_2
Store the current number to its corresponding/current slot number.
A→L_2(dim(L_2
End

Note: The following code is not my own algorithm. It is Alex Beresovsky's (a member of ti-calc.org) code. He programmed this random scramble routine for the ti-89, and I translated it to the ti-83 plus. A link to the source of his ti-89 code is below. However, I did modify his scrambler so that it could generate derangement scrambles needed for our purposes. I have made that my code and the comments to that inserted code in red font.
http://www.ticalc.org/archives/files/fileinfo/321/32103.html

Let the length of the list to be scrambled be S.
dim(L_2)→S
For a for-loop variable Y, from 1 to the length of the list being scrambled,
For(Y,1,S,1)
Lbl A
Seek out a random integer which is in the subset of the integers $\{1,2,...,S\}$.
The first argument is the lower bound and the second argument is the upper

bound (interestingly though, ti-83p basic does not seem to care if these two numbers are interchanged...). Store this randomly chosen integer as T.

```
randInt(1,S)→T
```

Now we swap the number in slot T with the number in slot M of list L_2.

```
L2 (T)→Z
L2 (Y)→M
```

If either Z or M are equal to the location in list L_2 that they are about to be stored in, then this contradicts a derangement scramble. Hence, we need to return to the point where a random integer is computed and try again until condition is met. We will index this location point as label A.

$$\begin{cases} \text{if} \quad I = 0, & \text{apply condition} \\ \text{if} \quad I \neq 0, & \text{do not apply condition} \end{cases}$$

```
If (Z=Y or M=T) and I=Ø
Goto A
Z→L2 (Y)
M→L2 (T)
End
```

Note: Any scrambling routine may be used here, although this is sufficient.

The Code: (Without Comments, but still code is indented for loops and if-then statements)

(The Algorithm Table Programs prgmAMOVES3 and prgmAMOVES2 are not shown here because they already had very little comments already).

PROGRAM A4X4SLVR:

```
ClrHome
Disp "INPUT SCRAMBLED","LIST:"
Input "",L2
ClrHome
prgmA2CYCLES
prgmA2CYCMOV
prgmA3CYCGRP
If fPart(dim(L4)/4)≠Ø
Then
     Output(4,3,"ODD PARITY!
     Pause
Else
     Output(4,4,"NO PARITY!
End
1→Z
While (Z+3)≤dim(L4)
ClrHome
Output(1,1,"3 CYCLE:
Disp "","
Output(2,1,L4(Z+1
Output(2,3,">
Output(2,4,L4(Z
Output(2,6,">
Output(2,7,L4(Z+3
Output(3,1,"PLEASE WAIT...
" →Str1
L4(Z+1)→X
prgmPERMTEST
L4(Z)→X
prgmPERMTEST
L4(Z+3)→X
prgmPERMTEST
sub(Str1,2,length(Str1)-1)→Str1
prgmAMOVES3
Output(3,1,"***************
Output(4,1,StrØ
Pause
Z+4→Z
End
If Z<dim(L4)
Then
     ClrHome
     Output(2,1,"           "
     Output(1,1,"2 CYCLE:
     {L4(Z+1),L4(Z)}→L3
     SortA(L3
     Disp "","
```

```
      Output(2,1,L₃(1
      Output(2,3,"<->
      Output(2,6,L₃(2
      Output(3,1,"PLEASE WAIT...
      " →Str1
      L₃(1)→X
      prgmPERMTEST
      L₃(2)→X
      prgmPERMTEST
      sub(Str1,2,length(Str1)-1)→Str1
      prgmAMOVES2
      Output(3,1,"****************
      Output(4,1,Str0
      Pause
End
ClrHome
Pause "SOLVED!
ClrHome
Disp "MADE BY:","CHRISTOPHER","MOWLA"
Pause
ClrHome
```

PROGRAM A2CYCLES:

```
For(A,1,dim(L2),1)
      If A=L2(A)
            Ø→L2(A)
End
Ø→dim(L4)
1→Z
While 1
      While L2(Z)=Ø
            Z+1→Z
      End
      2+dim(L4)→dim(L4)
      L2(Z)→B
      L2(L2(Z))→A
      Ø→L2(L2(Z
      If A=Z:Then
            Ø→L2(Z)
      Else
            A→L2(Z)
      End
      B→L4(dim(L4)-1)
      Z→L4(dim(L4))
      If sum(L2)=Ø
            Return
End
```

```
PROGRAM A2CYCMOV:

1→X

For(J,1,dim(L4)/2,1)
     If (X+3)>dim(L4)
          Return
     If L4(X+1)=L4(X+3)
     Then
          X+4→X
          J+1→J
     Else
          If J=dim(L4)/2
               Return

          {L4(X),L4(X+1)}→L L6
          X→A:X+1→B
          L4→L L1
          prgmLSTCUTM
          L L5→L4
          dim(L4)+1→A
          A-1→Z
          L L6→L L2
          L4→L L1
          prgmLSTINSRP
          L L5→L4
     End
End
```

```
PROGRAM A3CYCGRP:

1→C
While (C+3)≤dim(L4)
      L4(C+1)→U
      L4(C+2)→V
      L4(C+3)→W

      If U=W:Then
            V→L4(C+3
            W→L4(C+2
            C+4→C
      Else
            Ø→dim(L5
            dim(L5)+8→dim(L5
            If L4(C+1)>L4(C+3)
            Then
                  {L4(C),L4(C+1),L4(C+2),L4(C+3)}→LTEMP
                  LTEMP(1)→L4(C+2)
                  LTEMP(2)→L4(C+3)
                  LTEMP(3)→L4(C
                  LTEMP(4)→L4(C+1
            End

            L4(C→L5(1
            L4(C+1→L5(2
            L4(C+1→L5(3
            L4(C+2→L5(4
            L4(C+3→L5(5
            L4(C+1→L5(6
            L4(C+1→L5(7
            L4(C+2→L5(8

            L5→LL2
            L4→LL1
            C→A
            C+3→Z
            prgmLSTINSRP
            LL5→L4
            C+8→C
      End
End
```

PROGRAM LSTINSRP:

```
Ø→dim(ʟL5
For(X,1,A-1,1)
      1+dim(ʟL5→dim(ʟL5
      ʟL1(X)→ʟL5(X
End
Z-dim(ʟL5)+1→B
Ø→dim(ʟL4)
For(X,1,dim(ʟL1)-Z,1)
      1+dim(ʟL4)→dim(ʟL4)
      ʟL1(dim(ʟL5)+B→ʟL4(X)
      B+1→B
End
DelVar ʟL1
dim(ʟL5)→D
For(X,1,dim(ʟL2),1)
      1+dim(ʟL5)→dim(ʟL5)
      D+1→D
      ʟL2(X)→ʟL5(D)
End
DelVar ʟL2
dim(ʟL5→D
For(X,1,dim(ʟL4
      D+1→D
      1+dim(ʟL5→dim(ʟL5
      ʟL4(X→ʟL5(D
End
DelVar ʟL4
```

PROGRAM LSTCUTM:

```
Ø→dim(ʟL2
Ø→dim(ʟL3
For(Z,1,A-1,1)
     dim(ʟL2)+1→dim(ʟL2
     ʟL1(Z)→ʟL2(Z)
End
1→G
For(Z,B+1,dim(ʟL1),1)
     dim(ʟL3)+1→dim(ʟL3
     ʟL1(Z)→ʟL3(G)
     G+1→G
End
ʟL2→ʟL1
ʟL3→ʟL2
dim(ʟL1)+1→A
A-1→Z
DelVar ʟL3
prgmLSTINSRP
```

PROGRAM PERMTEST:

```
If X=1
Str1+"1→Str1
If X=2
Str1+"2→Str1
If X=3
Str1+"3→Str1
If X=4
Str1+"4→Str1
If X=5
Str1+"5→Str1
If X=6
Str1+"6→Str1
If X=7
Str1+"7→Str1
If X=8
Str1+"8→Str1
```

PROGRAM SCRAMBLE:

```
Ø→dim(L2)
For(A,1,N,1)
1+dim(L2)→dim(L2
A→L2(dim(L2
End
dim(L2→S
For(Y,1,S,1)
Lbl A
randInt(1,S)→T
L2(T)→Z
L2(Y)→M
If (Z=Y or M=T) and I=Ø
Goto A
Z→L2(Y)
M→L2(T)
End
```