MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Cambridge, Massachusetts
Project MAC

Artificial Intelligence Project
Memo 80

Memorandum MAC-M-242
June 28, 1965

PDP-6 LISP Input-Output for the Display

by William A. Martin

ABSTRACT

An intermediate level language for display programming has been embedded in LISP 1.5. The language is intended as a basis for higher level display languages and includes facilities for both generation and analysis of display information. Through the construction of a heirarchy of LISP functions it will be possible to assign a complicated meaning to a series of simple light pen motions, or to construct a complex picture. The intermediate level language should abstract from the light pen trajectory the information which these LISP functions require and provide a basis for picture construction which offers alternatives of expenditure of space, time, and programming effort. The first section of this memo discusses the system and gives programming examples. The details of the examples can be understood by reading the second section which discusses the implementation and the LISP functions available.

I. A System for Display Language Construction

The system provides a language macrosal for the generation of picture parts called objects. An object can be any combination of points, lines, and characters. An object is generated by calling the function macrosal [NAME; DESCRIPTION] described in the second section. The most common way to describe an object is to establish a set point. The set point is established utilizing PARAMETER, LOCY, and LOCX statements. The exact format of these statements is discussed below. An object description is terminated with a STOP statement. If NAME is T, the current description will be appended to the description of the last object generated. The first example, disp, generates a large object in this manner.

The user communicates with a display through a light pen. As the light pen sweeps across the screen, its trajectory can be used in many different ways. For example, it may be used to determine a point, a set of points, or a line. Or if a subpart of the display has been defined as an object, the trajectory may be simply interpreted as a pointer to this object or a point on the object. The LISP functions embedded in the display language facilitate acquiring the data needed to make these different levels of interpretation.

One problem in utilizing the light pen is to determine when it is near the screen and not just being moved into place. This is solved by using the

width of the field of view of the pen as measured by a tracking cross. This width decreases as the pen approaches the screen and a center dot is displayed whenever the field of view is less than a certain prescribable limit. This pen distance is available to LISP.

The approach of the pen to an object is considered significant. Just how close the pen must come before being noticed is a program variable. Its most recent position within this distance is recorded; in addition, so are the last 5 such positions, each at least a prescribed distance from the preceding one. This represents a crude way of gradually forgetting the details of the past. It is also possible to get the current coordinates of the pen, obtain a list of all objects currently seen by the pen, to report when the pen sees an object with a name other than a given name, or to require an object to move with the pen.

The example function sketch is a LISP function using several of these features. The function uses a subroutine to display five different light buttons. If the light pen is held near one of these buttons, the tracking cross will be centered about the point where the pen is seen. The program interprets pointing at these buttons to mean 1) draw a line, 2) move a line, 3) delete a line, 4) suppress the cross and 5) return control to the teletype. To draw lines touch the first button, a new line will then be drawn whenever the pen leaves the screen and then returns. This process is terminated whenever the pen returns near one of the light buttons. Additional LISP functions could be written to expand sketch into a program similar to

SKETCH PAD.

Often one wants to communicate to the display certain basic forms which are to be used in constructing larger units. These inputs might be a set of letters which are to be parsed into a sentence; or they might be a set of circuit elements or music symbols. In SKETCH PAD this is done by indicating the type of form and then moving the light pen so that the parameters of the form, the end points of a line segment for instance, can be abstracted from the trajectory. Alternatively, one could abstract both the type and the parameters from the trajectory. The example program argus uses a method developed by Teitelman which enables the user to teach the machine to replace a single line by a known form. A line is a single movement (however complex) on the surface of the display without lifting the pen. The LISP data structure is convenient for storing properties of the forms to be recognized.

The parsing of very large displayed expressions, such as LISP S-expressions for example, can be difficult for people. Furthermore, there may be alternative parsings. People can be aided by intensifying, upon request, grammatical subexpressions or sub-objects containing referenced segments or by providing additional displays meaningfully related to the first display. These might be rotated views of an object or shaded objects. Further development of the system is needed in this area.

It is the task of the programmer to organize a program and data base in such a way that the most needed inputs to a machine will have short representations and the most needed computations will be efficient. The

combinatorial aspects are such that this must be done through a series of levels of concepts. An important point is that it is not possible to complete an entire level at a time. The most useful concepts at a given level only become clear with the exploration of higher levels. The exploration of higher levels without intermediate concepts is, however, almost impossibly tedious. It is important in an experimental situation to have a system where one can make changes to any desired depth and provide for the irregular growth and reorganization of the data base.

In the present display system, experimentation will probably indicate that new statements for macrosal are needed, or that certain objects occur so often that more programming effort could well be spent in generating them efficiently. To provide for these possibilities macrosal has been programmed as a syntactic extension of the scope assembly language, sal. Sal is a LISP function which creates objects from lists of octal numbers. It is described in detail in part II. Provision has also been made for the addition of machine language subroutines which alter the objects as they are displayed. Furthermore, the system is organized so that no statements need be made about features of the display language which are not needed.

By embedding these display facilities in LISP one makes available a wealth of mechanisms which have proved useful in the analysis and generation of language and in the development of systems which can be increased incrementally in complexity.

II. Implementation

The use of LISP in this system has two distinct disadvantages. First, it is not possible to interrupt the LISP system at any point in time and immediately employ its full power. It may be in the midst of garbage collection. Garbage collection with the current version of PDP-6 LISP requires a noticable time. Second, the data types are too limited. It is not convenient to set up the type of list structure used in SKETCH PAD, but this can be approximated. A serious problem is the inability to set aside blocks of registers to contain display instructions and to store information about light pen actions.

To get around these problems a fixed buffer of 2048 words has been set aside for description of the display. All communication between LISP and the display goes through this buffer. This buffer contains two kinds of data structures; display lists and headers. One header is associated with each display list, which is a list of half word commands for the display. The headers build down from the top of the buffer and the display lists build up from the bottom.

During display an interrupt routine cycles through a dispatch table. Dispatches can occur to a pen track routine, a routine which displays the contents of the display buffer, a line drawing routine, and a routine which terminates the display.

The pen track routine displays a cross as was described earlier. In a crude effort to give the routine enough display time each cycle, it is called

between display of each object in the display buffer. Use of the clock would be better.

The format of the headers in the display buffer is as follows:

A = ON-OFF bit

B = Tracking Cross jump bit

C = Move with pen bit

NAME (Pointer to an atom)		
-(display list length	pointer to start of display list - 1
Pen hit distance	Pen hit count
Subr address	A B C
Most recent Y	Most recent X
Y	X
Y	X
Y	X
Y	X
Y	X

Figure 1 Header Format

Each display list has a name which is kept as the first word of its header. When an object is referenced by a LISP function, the headers are searched for one with the name mentioned.

The interrupt program cycles through the headers. It picks up a pointer for a BLKO instruction from the right half of the second word of each header. This BLKO is terminated by a STOP instruction at the end of each display list. If the subr address is not 0, then the subr at this address will be executed when the STOP is reached.

If the light pen is seen during display of some display list, control goes immediately to an interrupt program. Several conditional branchings can occur within this program. The interrupt program first re-displays the display list as a check against light pen noise. If the pen is seen a second time the pen cross movable bit is checked. If this bit is a 1 the pen tracking cross is centered about the point where the light pen was seen and the rest of the display list is displayed. Otherwise, a check is made to see if the center of the pen cross is within a specified minimum distance of the point seen. If not, the display list is continued with the light pen reenabled. If the pen is close enough, the coordinates of the point seen are stored in the fifth word of the header. The last five words of the header contain the coordinates of points seen by the pen in the past. Each of these history points is at least a specified distance from the preceding one. This distance is in the left half of the third word of the header. When a new point is seen, it is added to the history points if it is far enough away from the one most recently stored or if there are none. The number of history points is kept in the right half of the third word of the header. This number can be set to zero by LISP. Whenever there are more than five history points the oldest one is lost. After the point seen has been appropriately stored, a check is made to see if the object should move with the pen. If bit C is set, a pointer to the display list is transmitted to the pen track routine. The display list is then finished.

Display lists are put into the buffer by the sal language. If a line is to be drawn with the pen. A set point for the line is created with the sal language. Its display list is then incremented by the line drawing routine. This incrementong is terminated when the pen leaves the screen.

DISPLAY FUNCTIONS

macrosal[X;Y]

macrosal interprets its arguments and then calls sal on the result. X is the first argument for sal. Y is a description of a display list for sal. Y is a list of lists, each of which is a macro. The first word of each macro is an atom which has under the property MACROSAL a function of one argument. macrosal gets this function and applies it to the remainder of the macro list. The result of this function is a list for sal beginning and ending in mode 1. This list is prefaced by two integers which give respectively the mode that the display will end up in and the mode in which it must begin, if it is to interpret this list correctly. macrosal appends the successive macro expansions, using the prefacing integers to create the proper linking of modes.

The following macro word formats are in the system:

1. (PARAMETER penenable scale intensity)

Normally the first word in a set point, this statement creates a parameter word. If penenable, scale, or intensity is NIL, the corresponding field of the parameter word is not enabled.

2. (LOCY n)

Creates a non-displaying Y point word which sets the scope Y coordinate to n. This is normally the second statement in a set point.

3. (LOCX n)

Creates a non-displaying X point word which sets the scope X coordinate to N. This is normally the third and last statement in a set point.

4. (STOP)

Creates a parameter word with stop enabled. This is normally the last statement in a display list.

5. (ISTOP)

Creates a parameter word with stop enabled and a flag that the previous word has a breakout bit. The use of ISTOP is explained in the description of sal which follows.

6. (LOCYD n)

Creates a displaying Y point word which sets the display Y coordinate to n and displays a point.

7. (LOCXD n)

Creates a displaying X point word which sets the display X coordinate to n and displays a point.

8. (LINE X_1 Y_1 --- X_n Y_n)

Creates a sequence of non-displaying line segments from vector words. The display starts at the last point displayed or set by LOCX, LOCY. Each increment has size $X = X_{i+1} - X_i$, $Y = Y_{i+1} - Y_i$.

9. (LINED X_1 Y_1 --- X_n Y_n)

Creates a line like LINE, but displays it.

10. (LONGLINE X_1 Y_1 X_2 Y_2)

Creates a non-displaying vector continue word. A line will be drawn from the current display coordinates to the edge of the display. The slope of the line is $(Y_2-Y_1)/(X_2-X_1)$.

11. (LONGLINED X_1 Y_1 X_2 Y_2)

Creates a line like LONGLINE, but displays

12. (CHAR X_1 ----- X_n)

Creates a display of character made words which are the PNAME's of the atoms X_i. No spaces are inserted between the PNAME's.

sal[X;Y;Z]

X is the name of the display list to be created, or T. If X is T, then this list is appended to the last one created. If the previous list ends in STOP (3000_8), the STOP is removed. If it ends in ESTOP (403000_8), it not only removes this but zeros the breakout bit (400000_8) in the previous half word. Y is a description of the list to be created. There are two forms for the elements of Y corresponding to two modes for the assembly function sal. The function is initially in mode 0. In mode 1 sal removes lists of atoms, two at a time, from Y. Each atom is a number or is bound

In mode 1 sal removes lists of atoms, two at a time, from Y. Each atom is a number or is bound to one on the dotted pair list Z. For each pair of lists sal forms one display half word instruction by shifting the numbers on the second list the number of places specified by the corresponding number on the first list. When it encounters NIL on Y, it goes into mode 2. In mode 2 sal takes numbers or non-numerical atoms one at a time from Y. Each number is a display half word. If a non-numerical atom is encountered, sal looks at the previous number to see if it puts the display into increment or character mode. If in character mode, sal assembles the PNAME of the atom as characters. If in increment mode, sal gets a list of full words off the atom's property list with the indicator SCHAR. It assembles these as increment mode half words. Consecutive non-numerical atoms are assembled together in the same mode. When sal finds a number, the breakout bit is set. If sal finds NIL, it returns to mode 1. Mode 1 is more flexible and mode 2 is more economical. NIL is not a legal object name.

MORE FUNCTIONS

In the description below S stands for the name of an object with a set point. W stands for the name of an object with or without a set point.

sx[S]	Returns the current x coordinate of S.
sy[S]	Returns the current y coordinate of S.
smv[S]	Makes object S follow the light pen whenever S sees it. Returns S.
sunmv[S]	Negates smv. Returns S.
sxyinc[S;X;Y]	Increments the set point coordinates of S by X,Y. Returns S.
sclr[ ]	Clears the display buffer. Returns NIL.
sdlt[W]	Deletes object W from the display buffer and returns W.
ptrk[X;Y]	Starts the pen tracking cross at X,Y and returns NIL.
puntrk[ ]	Stops the display of the pen track cross. Returns NIL.
px[ ]	Returns the x coordinate of the pen.
py[ ]	Returns the y coordinate of the pen.

ptchp[ ]	Returns NIL if the pen is far from the screen.
srn[ ]	Starts the display build up in the buffer. Display will continue until sunrn[ ] is called.
sunrn[ ]	Stops the display and returns NIL.
pldw[X;Y]	To use this command, use macrosal to set up a set point at the current pen position followed by a zero length relative line and terminated with ISTOP. Then call pldw. Increments of length at least X and scale Y will be added to this object until the pen leaves the screen. Returns NIL.
phclr[W]	Clears the pen approach history points of object W and returns W.
ph[W]	Returns a list of the pen approach history points for object W.
phc1[W;X]	Sets to X the minimum distance between pen approach history points for object W. X is an integer between 0 and 1024.
phc2[X]	Sets to X the minimum distance between pen approach history points which is assumed for newly created objects.

phc2[X]	Sets to X the maximum distance on the screen at which the pen can see an object. X is an intege between 0 and 40.
sint[W;Y]	Sets to Y the intensity of object W. Returns W. W is an integer between 0 and 7.
scopy[W1, W2]	Copies object W1 and names the copy W2. Returns
plsh[ ]	Returns a list of all objects currently being seen by the pen.
sscl[W;Y]	Sets the scale of object W to Y. Returns W. Y is an integer between 0 and 3.
plhw[W]	Returns the name of the first object other than W seen by the pen.
ptchw[ ]	Waits until the pen is near the screen and then returns NIL.
puntchw[ ]	Waits until the pen is not near the screen and then returns NIL.
pjp[W]	If the tracking cross is not following the pen and pjp[W] has been executed, then when the pen sees object W, the tracking cross will be center about the point of W seen. Returns W.
pumjp[W]	Negates pjp and returns NIL.

poh[ ] — Returns the name of the most recent object to be seen by the pen or NIL if none have been seen since pohc was executed.

pohc[ ] — Sets the last object seen by the pen to NIL. Returns NIL.

sline[S] — Returns a list of the coordinates of the end points of the line segments which make up object S.

ssubr[W;Y] — Causes the subroutine beginning at location Y to be executed each time object W is displayed.

sloc[W] — Intensifies a subexpression of object W which is indicated by the light pen. Subexpressions are marked in the object by pseudo parentheses which are not displayed. A pseudo left paren is indicated by the parameter word 600001_8 and a pseudo right paren by the parameter word 600002_8.

sadd[W] — Counts the number of pseudo left parens to the intensified subpart of W.

```
(PRINDEF DISP MV EXPOND)
(DEFLIST ((DISP (LAMBDA (N) (PROG (U V) (MACROSAL (QUOTE DISP) (
QUOTE ((PARAMETER 1 2 3) (LOCY 1) (LOCX 1) (LINED 0 0 0 0) (ISTO
P)))) (SETO U (EXPAND N NIL)) A (COND ((NULL U) (RETURN NIL))) (
MACROSAL T (LIST (COND ((EQUAL (CAR U) (QUOTE U)) (QUOTE (LINED
0 0 0 4))) ((EQUAL (CAR U) (QUOTE D)) (QUOTE (LINED 0 0 0 777777
777774))) ((EQUAL (CAR U) (QUOTE L)) (QUOTE (LINED 0 0 777777777
774 0))) (T (QUOTE (LINED 0 0 4 0)))) (QUOTE (ISTOP)))) (SETO U
(CDR U)) (GO A))))) EXPR)

(DEFLIST ((MV (LAMBDA (X) (COND ((EQ X (QUOTE U)) (QUOTE D)) ((E
Q X (QUOTE D)) (QUOTE U)) ((EQ X (QUOTE L)) (QUOTE R)) (T (QUOTE
L)))))) EXPR)

3
(PRINDEF EXPAND)
(DEFLIST ((EXPAND (LAMBDA (N EXP) (PROG (A B C D E) (SETQ EXP (Q
UOTE (U R D R))) START (COND ((ZEROP N) (RETURN EXP))) (SETQ N (
PLUS N -1.0)) LOOP (SETQ A (CAR EXP)) (SETQ EXP (CDR EXP)) (SETO
B (CAR EXP)) (SETQ EXP (CDR EXP)) (SETQ C (CAR EXP)) (SETQ E (NCON
C E (LIST B A (MV B) A A B C B (MV C) B C C (MV B) C B D))) (CON
D (EXP (GO LOOP))) (SETQ EXP E) (SETO E NIL) (GO START))))) EXPR
)

3
```

```
(PRINDEF SKETCH)
(DEFLIST ((SKETCH (LAMBDA (N X) (PROG (U W) (COND (X (GO A))) (P
JP (BUTTON (QUOTE S) 100 1700)) (PJP (BUTTON (QUOTE D) 300 1700)
) (PJP (BUTTON (QUOTE M) 500 1700)) (BUTTON (QUOTE P) 700 1700)
(PJP (BUTTON (QUOTE R) 1100 1700)) A (POHC) (SETQ W (PLHW NIL))
B (COND ((MEMBER W (QUOTE (D M))) (GO C)) ((EQUAL W (QUOTE S)) (
GO G)) (( EQUAL W (QUOTE R)) (GO H)) ((EQUAL W (QUOTE P)) (GO I))
((EQUAL U (QUOTE M)) (GO E)) ((EQUAL U (QUOTE D)) (SDLT W))) (G
O A) C (SDLT U) (SETQ U W) (BUTTONON W) (SETQ W (PLHW W)) (GO B)
 E (SMV W) (PUNTCHW) (SUMMV W) (GO A) G (SDLT U) (BUTTONON W) (S
ETQ U W) D (PUNTCHW) (PTCHW) (COND ((GREATERP (PY) 1600) (GO A))
) (LINEDRAW (GENSYM) N 1) (GO D) H (SDLT U) (RETURN NIL) I (SDLT
U) (SETQ U W) (BUTTONON W) (PUNTRK) (GO A))))) EXPR)

3
(PRINDEF LINEDRAW BUTTON BUTTONON BUTTONON1)
(DEFLIST ((LINEDRAW (LAMBDA (NAME QUALITY SCALE) (PROG NIL (PTCH
W) (SAL NAME (APPEND (QUOTE ((0) (0) (0 4))) (CONS (LIST 34114 S
CALE) (CONS (LIST 0) (APPEND (CONS (LIST (PLUS 220900 (PY))) (CO
NS (QUOTE (0)) (LIST (LIST (PLUS 114000 (PX)))))) (QUOTE ((0) (4
00000) (0) (30000)))))) NIL) (PLDW QUALITY SCALE) (RETURN NAME))
))) EXPR)

(DEFLIST ((BUTTON (LAMBDA (NAME X Y) (SXYING (SAL NAME (QUOTE ((
```

```
Ø) (Ø) (Ø) (34114) (Ø) (22ØØØØ) (Ø) (114ØØØ) (Ø) (2ØØØ1Ø) (Ø) (3
Ø4ØØØ) (Ø) (2ØØ21Ø) (Ø) (6Ø4ØØØ) (Ø) (3ØØØ))) NIL) X Y)))) EXPR)

(DEFLIST ((BUTTONON (LAMBDA (NAME) (BUTTONON1 NAME (SX NAME) (SY
 NAME ))))) EXPR)

(DEFLIST ((BUTTONON1 (LAMBDA (NAME X Y) (SXYING (SAL NAME (QUOTE
((Ø) (Ø) (Ø)  (34114) (Ø) (22ØØØØ) (Ø) (114ØØØ) (Ø) (7Ø4Ø1Ø) (Ø)
(3ØØØ))) NIL) X Y)))) ESPR)
3
```

```
(PRINDEF ARGUS MAKSTRING MAKSTRING1 MAKSTRING2
MAKSTRING3)
DEFLIST ((ARGUS (LAMBDA NIL (PROG (U V R Q MERGELIST NQ) (SETQ
U (LINEDRAW (QUOTE ARGUS) 2 1)) (PUNTCHW) (SETQ V (MAKSTRING (SL
INE U))) (SETQ R (MAP2 V ATREES (QUOTE GETC))) (MAP2 AWEIGHTS R
(FUNCTION (LAMBDA (WEIGHT LIST) (MAP LIST (FUNCTION (LAMBDA (CAN
DIDATE) (SETQ MERGELIST (MERGE3 MERGELIST))))))) (COND ((NULL M
ERGELIST) (GO A))) (SETQ NQ (CDAR MERGELIST)) (SETQ Q (CAAR MERG
ELIST)) (MAP MERGELIST (FUNCTION MERGE1)) A (SDLT U) (MACROSAL (
QUOTE LAST) (LIST (QUOTE (PARAMETER 1 3 3)) (LIST (QUOTE LOCY) A
YMIN) (LIST (QUOTE LOCX) AXMIN) (LIST (COND ((GET Q (QUOTE SCHAR
)) (QUOTE SCHAR)) (T(QUOTE CHAR))) (COND ((NULL Q) (QUOTE ?)) (
T Q))) (QUOTE (STOP)))) (COND ((NOT (EQUAL AMODE (QUOTE TRAIN)))
 (RETURN Q))) (PRINT R) (SETQ U (READ)) (COND ((NOT (EQUAL Q U))
 (GO B))) (CSETQ AWEIGHTS (MAP2 AWEIGHTS R (FUNCTION (LAMBDA (X
Y) (COND ((MEMBER U Y) (PLUS X 1)) (T X)))))) B (CSETQ ATREES (M
AP2 V ATREES (QUOTE PUTC))) (RETURN U))))) EXPR)

(DEFLIST ((MAKSTRING (LAMBDA (LINE) (PROG (U V MIN MAX V1 V2) (M
AKSTRING2 (EVERYOTHER LINE)) (CSETQ AXMIN MIN( *MAKSTRING2 (EVER
YOTHER (CDR LINE))) (CSETQ AYMIN MIN) (RETURN U))))) EXPR)

(DEFLIST ((MAKSTRING1 (LAMBDA (X) (COND ((GREATERP (CAR X) V2) (
LIST NIL (MAKSTRING3 X NIL))) (T (LIST (MAKSTRING3 X NIL) NIL)))
))) EXPR)

(DEFLIST ((MAKSTRING2 (LAMBDA (X) (PROG NIL (SETQ MIN (CAR X)) (
SETQ MAX (CAR X)) (MAP X (QUOTE MINMAX)) (SETQ V (QUOTIENT (PLUS
 MAX (MINUS MIN)) 3)) (SETQ V1 (PLUS MIN V)) (SETQ V2 (PLUS V1 (
QUOTIENT V 6))) (SETQ U (APPEND (MAKSTRING1 X) U)) (SETQ V1 (PLU
S V1 V)) (SETQ V2 (PLUS V2 V)) (SETQ U (APPEND (MAKSTRING1 X) U)
) (RETURN NIL))))) EXPR)

(DEFLIST ((MAKSTRING3 (LAMBDA (X LEFT) (COND ((NULL X) 1) ((OR (
AND LEFT (GREATERP (CAR X) V2)) (AND (NOT LEFT) (GREATERP V1 (CA
R X)))) (PLUS 1 (MAKSTRING3 (CDR X) (NULL LEFT)))) (T (MAKSTRING
3 (CDR X) LEFT))))))) EXPR)

3
```

```
 (PRINDEF MAP MAP2 SUB1 EVERYOTHER MINMAX MERGE1
MERGES GETC PUTC)
(DEFLIST ((MAP (LAMBDA (X FN) (COND ((NUL X) NIL) (T (CONS (FN
(CAR X)) (MAP (CDR X) FN)))))))) EXPR)

(DEFLIST ((MAP2 (LAMBDA (X Y GN) (COND ((NULL X) NIL) (T (CONS (
FN (CAR X) (CAR Y)) (MAP2 (CDR X) (CDR Y) FN)))))))) EXPR)

(DEFLIST ((SUB1 (LAMBDA (X) (PLUS 68719476735 X)))) EXPR)

(DEFLIST ((EVERYOTHER (LAMBDA (X) (COND ((OR (NULL (CDR X)) (NUL
L (CDDR X))) (LIST (CAR X))) (T (CONS (CAR X) (EVERYOTHER (CDDR
X))))))))) EXPR)

(DEFLIST ((MINMAX (LAMBDA (X) (COND ((GREATERP X MAX) (SETQ MAX
X)) ((GREATERP MIN X) (SETQ MIN X)) (T NIL))))) EXPR)

(DEFLIST ((MERGE1 (LAMBDA (LIST) (COND ((GREATERP (CDR LIST) NO)
 (PROG2 (SETQ NO (CDR LIST)) (SETQ ) (CAR LIST)))) (T NIL))))) E
XPR)

(DEFLIST ((MERGE3 (LAMBDA (MERGELIST) (COND ((NULL MERGELIST) (L
IST (CONS CANDIDATE WEIGHT))) ((EQUAL CANDIDATE (CAAR MERGELIST)
) (CONS (CONS CANDIDATE (PLUS WEIGHT (CDAR MERGELIST))) (CDR MER
GELIST))) (T (CONS (CAR MERGELIST) (MERGE3 (CDR MERGELIST)))))))
) EXPR)

(DEFLIST ((GETC (LAMBDA (PSTRING TREE) (COND ((NULL PSTRING) NIL
) ((NULL TREE) NIL) ((ZEROP PSTRING) (CAR TREE)) (T (GETC (SUB1
PST:ING) (CDR TREE)))))))) EXPR)

(DEFLIST ((PUTC (LAMBDA (PSTRING TREE) (COND ((NULL PSTRING) TRE
E) ((NULL TREE) (COND ((ZEROP PSTRING) (LIST (LIST U))) (T (CONS
 NIL (PUTC (SUB1 PSTRING) NIL))))) ((ZEROP PSTRING) (COND ((MEMB
ER U (CAR TREE)) TREE) (T (RPLACA TREE (CONS U (CAR TREE)))))) (
T (CONS (CAR TREE) (PUTC (SUB1 PSTRING) (CDR TREE))))))))) EXPR)
```