PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G09G 3/02	A1	(11) International Publication Number: WO 93/08559
		(43) International Publication Date: 29 April 1993 (29.04.93)

(21) International Application Number: PCT/US92/09123

(22) International Filing Date: 22 October 1992 (22.10.92)

(30) Priority data:
781,747 23 October 1991 (23.10.91) US

(60) Parent Application or Grant
(63) Related by Continuation
US 781,747 (CIP)
Filed on 23 October 1991 (23.10.91)

(71) Applicant *(for all designated States except US)*: ELECTRONIC INK [US/US]; 1119 Park Hill Road, Berkeley, CA 94708 (US).

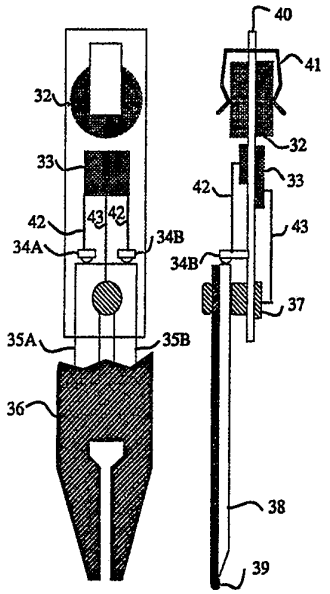
(72) Inventor; and
(75) Inventor/Applicant *(for US only)* : KLEY, Vic [US/US]; 1119 Park Hill Road, Berkeley, CA 94708 (US).

(74) Agent: LARWOOD, David, J.; Crosby, Heafey, Roach & May, 1999 Harrison Street, Oakland, CA 94612 (US).

(81) Designated States: AT, AU, BB, BG, BR, CA, CH, CS, DE, DK, ES, FI, GB, HU, JP, KP, KR, LK, LU, MG, MN, MW, NL, NO, PL, RO, RU, SD, SE, US, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, SN, TD, TG).

Published
With international search report.

(54) Title: COMPUTER INPUT SYSTEM



(57) Abstract

A computer input system (30) consists of hardware for determining the position of a drawing tool such as a pen (30) in relation to a display incorporated in a writing tablet (50) and software components for identifying the pen (30) user and for accelerating information entry be expanding stroke characteristics or word abbreviations. The writing tablet (50) includes a writing surface with a pixel array (100) connected to a host processor. The pen (30) includes an optical sensor (34A, 34B) connected to a controller and to an output device coupled to a remote receiver. One form of receiver is a transparent, resistive sheet (53) essentially covering the writing tablet (50). The host processor can control the pixel display to illuminate pixels (100) near the pen (30) and can receive information from the pen (30). When the user brings the pen (30) to the writing surface (50), the host can display pixels (100) under the pen (30), leaving a trail of electronic ink.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria
AU	Australia
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CS	Czechoslovakia
CZ	Czech Republic
DE	Germany
DK	Denmark
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IT	Italy
JP	Japan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
LI	Liechtenstein
LK	Sri Lanka
LU	Luxembourg
MC	Monaco
MG	Madagascar
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SK	Slovak Republic
SN	Senegal
SU	Soviet Union
TD	Chad
TG	Togo
UA	Ukraine
US	United States of America
VN	Viet Nam

COMPUTER INPUT SYSTEM

Field of the Invention

The present invention relates to pen and brush input for electronic tablets or displays and software for controlling and speeding up input through pen, brush and keyboard text input systems, user identification and point of sale/teller systems.

Background of the Invention

The computer industry has developed a number of writing tablets and light pens to allow a user to interact with a computer in a rough approximation of putting pen to paper. Such devices generally give a line of fixed width in response to user input. Some newer devices include a pressure sensitivity, but no system has been developed that emulates a pen or brush in a natural way. No one has found a natural way to increase input speed for either pens or keyboard input. In addition, the user must be identified in order to operate most stroke recognition, point of sale and automatic teller systems.

Summary Of The Invention

The invention consists of hardware embodiments for determining the position of a drawing tool in relation to a display incorporated in a writing tablet. The invention includes software components for identifying the pen user including kernal information for complete or partial stroke recognition and communication, particularly useful for completion of point of sale or debit transactions. A preferred drawing tool takes the form of a pen or brush and herein generally will be referred to as a pen.

The present invention provides a pen-based computer input system comprising a pen and tablet. The writing tablet includes a writing surface with a pixel array connected to a host processor, which may be incorporated in the writing tablet. The pen includes an optical input connected through an optical sensor to a pen controller. The pen controller, in turn, is connected to an output device which can be directly or capacitively coupled to a remote receiver. The tablet may include or be connected to a receiver designed to receive information from the pen or brush. One form of receiver is a transparent, resistive sheet essentially covering the pixel array. The host processor can control the pixel display to illuminate pixels near the pen, thereby communicating with the pen, and the pen can communicate with the host processor through the output device and remote receiver. When the user brings the pen to the writing surface, the host can display pixels under the pen, leaving a trail of electronic ink and giving the user the feel of actually writing. By displaying a wider path of pixels when the user presses the pen more firmly, the effect can approximate writing with a fine fountain pen. The process of stroke processing is illustrated in the flowchart of Figure 1.

Optical means in the pen or brush provide information about drawing tool angle and pressure and brush bristle flex and spread. Logic and or microprocessors in the pen or brush body act alone or in conjunction with software in a host computer to provide stroke processing and estimate or refine the position of the pen nib or brush bristles on the tablet surface.

Software in the host also acts to increase input speed by providing a method for abbreviating common or frequently used words, phrases or text sections from either keyboard or pen sources and to suggest new abbreviations by evolving strokes from often used words as distinct from the overall stroke history and letter abbreviations from the overall word and phrase history. The expansion process is illustrated by the flowchart of Figure 2.

One object of the invention is to provide a set of stroke parameters peculiar to the user and an associated personal identification which can be stored in the pen to allow optimal stroke recognition in a host system, including a new host when the user first uses it.

Another object of the invention is to provide a system in which pen position hardware cost is minimized in the tablet and host and an electronic pen cartridge is created suitable for inclusion in a wide variety of pen body styles, diameters and weights by many different suppliers.

Another object of the invention is to provide a dual function device which may contain a drawing tool of this invention plus a traditional writing device such as a fountain pen or ball point pen, a graphite pencil or eraser or a second drawing tool of this invention.

Brief Description Of The Drawings

Figure 1 is a overview flow chart of the pen and input accelerator system.

Figure 2 is a detailed flow chart of the accelerator input software subsystem.

Figure 3 is a side view of the pen and tablet.

Figure 4 is a top view of the tablet and display.

Figure 5 is a detailed view of the tablet and writing surface.

Figure 6 is a detailed sectional view of a pen nib and of a brush and its active elements.

Figure 7 is a view of the pen body and an enlarged view of the nib and its electronic elements.

Figure 8 is a block diagram showing the interaction of a host computer, tablet and writing tip.

Detailed Description Of The Invention

The present invention consists of two major components - stroke entry and character handling. Stroke entry involves detecting the presence of a writing instrument near a writing tablet and selectively turning on pixels in the tablet to reflect the user's actions, effectively applying "electronic ink." Once a character is complete, that character is analyzed to detect special encoding and to speed information entry.

The present invention preferably utilizes the tablet illustrated Figures 4 and 5. Tablet **50** includes a raster display of individually addressable pixels **100** arranged in rows and columns. The pixels can be of most any type of conventional manufacture, including LCD, ferroelectric liquid crystal, phosphor or other type known in the industry. The tablet writing surface **51** is covered by a relatively transparent, resistive sheet **53**, which can be a metal such as gold or a metal oxide such as indium tin oxide. The resistive sheet may be the topmost layer of the display or it may be covered with a protective layer **54**, illustrated in Figure 5. The conductive sheet is connected to conductive sensors **52A, 52B** at opposite ends of the writing surface. The conductive sensors **52A,**

52B are connected in turn to circuitry which calculates the relative strength of a signal generated by the pen as picked up by the two conductive sensors and then connected to the host computer (not shown). This allows the ratiometric determination of the approximate position of the pen along one dimension of the writing surface. This signal also carries information from the pen.

Referring to Figures 7A and 7B, pen 30 includes writing tip 31, which can be manufactured as an interchangeable insert adapted to pen barrels of many manufacturers. Writing tip 31 consists primarily of support 40 which carries conductive nib 36 and light guides 35A, 35B which are optically coupled to photodetectors 34A, 34B which are connected in turn via leads 42 to controller 33. Nib 36 is a conductive or semi-conductive material such as pen brass terminated by tip 39, constructed of a material which provides a pleasing feel on the "writing" surface of tablet 50. Either or both photodetectors 34A and 34B independently detect light, each carrying an external signal to controller 33 in a manner well known in the art. For example, some methods used in photo-optical isolation devices can be utilized in the present invention. Battery 32 provides power for all components of writing tip 31. Controller 33, preferably an ASIC chip, controls and receives information from photodetectors 34A, 34B and transmits information to nib 36 through lead 43 and coupling pin 37. Controller 33 can send information to a host computer through a direct cable connection (not shown) but preferably broadcasts a signal through nib 36. If nib 36 is sufficiently close to resistive sheet 53, a signal generated by controller 33 is transmitted through nib 36 using direct or capacitive coupling through resistive sheet 53 and conductive sensors 52A, 52B to appropriate circuitry and thence to the host computer.

Referring to Figure 6, an alternative input device consists of electronic brush 29. Brush 29 includes "bristles" in the form of a fiber optic bundle of fibers 26 constrained by collar 27 and coupled to photodetector array 28. Each fiber 26 is coated with a conductive film or alternately (not shown) interspersed with conductive flexible wires. Brush collar 27 constrains the fiber and blocks extraneous light. Photodetector array 28 may be a linear or area array, with elements connected to a controller 33 (not shown) which in turn can transmit electronic signals through conductive fibers 26 in a manner closely analogous to the actions of writing tip 31. The system and method of this invention will be generally described using writing tip 31 as an example, but one skilled in the art will understand how to apply brush 29 in a similar manner.

Referring to Figure 8, the host can communicate with controller 33 by cycling the on-off state of pixels 100 in tablet 50. The host can selectively activate pixels 100 and compare timing of pixel illumination versus return signals from controller 33, through nib 36 and resistive sheet 53, to determine where the pen is located, using techniques well known for interacting with prior art light pens. Since the pen contains photo-detectors 34A and 34B, each photodetector can pickup signals from separate illuminated pixels which can provide information about the angle of nib 36 on tablet 50. In addition, combinations of controlled pixel illumination and detection schemes in writing tip 31 can provide information about the proximity of writing tip 31 to tablet 50. This information can be used to vary the width of a stroke to be illuminated by pixels according to the pressure applied by the user. A brush, as illustrated in figure 6, can receive signals through multiple sensors, providing information about the location of each bristle. For any drawing tool, any pixel which is sufficiently close to the tool can be maintained in the "on" state. The net result

is the application of "electronic ink" which reflects the angle and pressure of the user's writing actions.

The input system can detect the location of the drawing tool in relation to the writing tablet including which pixels are closest to any component of the pen and which pixels near the pen may be weakly interacting with the pen. This allows calculation of the timing, pressure and angle of each stroke as it is applied. This information can be used to derive stroke kernal information which can be compared with recorded stroke kernal information or other information to identify the user. This may be used for identification of an individual, to select a certain stroke recognition sequence, or otherwise used to facilitate interaction of a user with a stroke recognition system. Recorded stroke information may be stored in a central host, for example a computer tied to a series of ATM or point of sale systems, in a host controller in the tablet, or in the local controller in the pen. There may be multiple stored patterns for a single user, or there may be multiple stored patterns for a series of users, who might for example share a single pen. Alternately the controller can transfer user signature identification information to the host. The controller may act to confirm a transaction from the host and contain stored information about the user's credit limit or debit account which is updated by the host with each transaction.

In using brush **29**, fibers **26** are in optical contact with the pixels as they are changed which is detected by photodetector array **28**. Thus brush **29** has the ability to sense individual "bristle" contact and contact angle to provide the same qualities of use as a normal paintbrush.

A writing implement of this invention can be personalized to contain user identification information as well as information about user preferences and writing habits. This information can be stored in memory, e.g. EPROM, connected to controller **33** so that pen **30** or brush **29** can be used with multiple tablets **50** effectively interchangeably.

Referring to Figure 6, pixels **100** are cycled at times known to the host computer. Pen nib **36** with light guides **35A** and **35B** under the tip (not shown in Figure 6) "see" a series of pixel elements. As the pixels are cycled the position of the nib and its angle with the paper is determined by coordinating the pixel change with the sensed light through the pen. In addition as the nib pressure increases the light guides make greater contact with more pixels which provides pressure information to the pen system.

When writing tip **31** is activated, for example by being held by the user, it repeatedly transmits stored user identification and key parameters. When writing tip **31** is held sufficiently close to tablet **50**, the host "wakes up" and begins processing input. The host can read and store the user identification and key parameters. The host can activate pixels **100** on tablet **50**. The signal from writing tip **31** is picked up through resistive sheet **53** by conductive sensors **52A** and **52B** producing two signal amplitudes according to distances **20** and **21** between pen **30** and conductive sensors **52A** and **52B**. The ratio of these signal amplitudes provides a precise location of pen **30** along the tablet surface. The host uses this information to flick on pixels sequentially across the tablet surface while collecting the detected changes as described above. Once position, nib angle and pressure are determined, the information is passed to subroutines which update the display to show the proper flow of electronic "ink." Stroke recognition is used to identify and convert the character input by the user. Conventional character recognition programs can perform

basic character conversion but the stroke information regarding angle, pressure and other information provided by the new pen input system can facilitate character recognition.

One means for ratiometric positioning in one dimension has been described in detail above. Other ratiometric means include determining a resistance, a capacitance, or an appropriate electromagnetic value in the X dimension, the Y dimension, or both dimensions, preferably simultaneously or nearly so. A sonic position evaluation system may be used, as may an active optical position evaluation method.

In an alternate embodiment in conjunction with an adequately fast display (such as ferroelectric liquid crystals, gas plasma, etc.) pen or brush position may be entirely determined by sequentially flicking (turning on and off or XOR'ing) pixels only while monitoring the response of the pen (brush). In this embodiment, transparent resistive film **18** (Figure 4) is not required, but it may still serve as the primary communications channel of the system.

The system allows rapid determination of a drawing tool's location by first performing a coarse search, then refining the search until a specific pixel or group of pixels in the immediate vicinity of the drawing tool can be identified.

Traditional means of identifying the position of a light pen or similar pointing device is to activate all pixels in a display sequentially, usually by a regular raster scan, then signalling a host computer when the pointing device has detected an activated pixel. This timing information can be coordinated with information about the raster driver to derive the position of the pointing device. Traditional pointing devices are crude equivalents of the drawing tool of the present invention but the principles for locating the precise location of an older pointing device or the new drawing tool are similar. The coarse search of this invention improves this old method by locating the approximate position of the drawing tool and only pixels near the drawing tool need be tested in order to locate the precise location of the drawing tool.

A new means of approximating the location of the drawing tool is a pattern search. Selected regions of the writing tablet can be activated selectively and the drawing tool can be monitored for a response. Where a response is detected, the corresponding region can be subdivided into smaller patterns which can be tested again, then subdivided as much as necessary until the precise location of the drawing tool is determined. One potential set of patterns is a traditional binary tree search where half the display is activated, then the other half. The half in which the drawing tool is located is then subdivided into two portions, and the process is repeated until the position is sufficiently determinate. This pattern matching does not rely on raster scanning and is best performed with a device that can randomly address pixels, either singly or in groups.

This method of coarse positioning is also useful with existing precision positioning devices such as digitizing tablets. Current devices can be modified to utilize and benefit from coarse positioning through minimal modification of the hardware or software drivers.

Once the approximate location of the drawing tool has been determined, specific pixels in the approximate area can be activated while responses from the drawing tool are monitored. The system can continue to test pixels in that approximate area while the drawing tool is in contact or approximate contact with the writing tablet, enabling the system to follow which pixels are near the drawing tool at times in the future. If the drawing tool is moved away from the writing surface,

as between words or between some strokes, the system can continue to monitor the general area of the last stroke. It is convenient to think of the approximate location of the drawing tool as a "bubble" which expands or contacts depending on the state of the system. The system should test pixels within the bubble with sufficient frequency to track changing states of the drawing tool in relation to the writing surface.

One skilled in the art can readily set a frequency of pixel testing and a range of pixel locations to test during various operating states, including: no drawing tool near the writing tablet, a drawing tool in contact with the writing tablet, a slowly or rapidly moving drawing tool, a drawing tool recently in contact with the writing tablet, and other states of interest. For example, when the drawing tool is moving slowly, then the bubble can be rather small and the sampling frequency can be relatively low. If the drawing tool is moved away from the writing tablet, the bubble can be checked periodically within an expanding bubble, the expectation being that there is a high initial probability that the drawing tool will be returned in the vicinity of the previous stroke, but the probability of returning to that specific or general location decreases with time. At some point, the system resumes the general pattern testing to identify when the drawing tool does return and then to locate the position of the drawing tool.

Once a character has been input, it is processed according to the following steps. Each character is placed in a character buffer. The Fasthand software then takes this character (whether from pen, brush or keyboard source) to perform the input acceleration and shorthand functions illustrated in Figure 2. An input character is first tested to see if it is a special character of some sort including an ideograph or evolved stroke set. If not, the character is placed in a word buffer and the next character is accepted. If the input character is a normal word terminator, such as space, comma, period (followed by space) or other punctuation, the text contents of the word buffer are examined. The word terminator is combined with the text and the text is tested to see if it matches any of a list of abbreviations, stored, for example, in a look up table. If any matches are found, the corresponding expanded text is substituted for the abbreviated text and output and processing continues with the next character in the input stream.

If there are no exact matches for the text and the known abbreviations, then the text is examined for prefix or suffix abbreviations. The first two or three letters are tested against a table of known prefix abbreviations and if a prefix abbreviation is found, the corresponding expanded prefix is substituted in the text. In a similar fashion, the last letters of the text are tested against either a suffix algorithm or a user updatable table of suffix abbreviations or both and any corresponding suffixes are substituted in the text regardless of whether the first portion of the word has been abbreviated. In a preferred embodiment, the last letter of a word or entry may be capitalized using the conventional shift key or caps lock, signalling to the controller that the user desires a suffix expansion using an available algorithm or table entry. The text is then output and processing continues with the next character in the input stream.

One or more special characters can be interpreted as commands. In the preferred implementation, the equals sign is used to enter new abbreviations in the abbreviation table. Any text, generally a series of characters, followed by an equals sign, with no intervening space, is treated as a new abbreviation and any text following the equals sign, including spaces or other punctuation, up to the next equals sign following a non-space character, is treated as the

corresponding expanded text. The new abbreviation and corresponding expansion are stored with other known abbreviations for subsequent interpretation.

Another useful command causes the processor to cease interpreting the subsequent word or series of words. One could develop an elaborate language of special symbols to implement various control and interpretation functions. Each of these command schemes are encompassed within the claims of the present invention.

Another useful command uses an up arrow (^) as an alternative way to set "caps lock" on, causing any text following it to be capitalized. This may also be accomplished by setting the caps lock directly.

Words which are not expanded, in other words, which have no abbreviation, either in whole or in part, are examined for potential abbreviation. Each non-abbreviated word is entered in a history table, setting a frequency flag for that word or incrementing a counter associated with that word. If the count is high enough then a proposed abbreviation is shown via pop-up to the user. Alternately for stroke based characters the stroke collection making up the letters and/or ideographs in the word are sorted and reduced to a unique related subset which is presented to the user as a suggested new efficient form. A fully spelled out word elicits a pop-up window with the abbreviation from the table as a teaching tool and as a reminder.

A batch program periodically analyzes the history table creates a frequency table and suggests abbreviations for text from other sources, or from abbreviations previously created by the user.

Source code for one implementation of this process is included as Example 1. This source code carries out the novel features of the software but does not include all of the code for basic functions or unrelated prior art software which is used in the actual device and method.

A general description of a device and method of using the present invention as well as a preferred embodiment of the present invention has been set forth above. One skilled in the art will recognize and be able to practice additional variations in the methods described and variations on the device described which fall within the teachings of this invention.

Example 1:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "pcstuff.h"
#include "ab.h"
#include "list.h"

/***Copyright 1992 Electronic Ink, Inc.*********************************/
int PrefixCount = 0;
int DefCount = 0;
int SuffixCount = 0;
int ShortCount = 0;

PTR Abbreviations[MAX_DEFS];
PTR Words[MAX_DEFS];

char *ShortWords[MAX_SHORTS];

PTR Prefixes[MAX_PREFIXES];
PTR PreAbbrevs[MAX_PREFIXES];

PTR Suffixes[MAX_SUFFIXES];
PTR SufAbbrevs[MAX_SUFFIXES];

/*************************************************************************/
/*****
  This program uses two independently sorted arrays of pointers to nodes.
  One array is sorted by Abbreviations, and the other list is sorted by
  Words.  This allows us to use a binary search on both Words and
  Abbreviations.
*****/
/**********************LINKED LIST OPERATIONS *************************/
/*
  MakeNode() mallocs space for and returns a pointer to a list node,
  initiallizing the fields to the specified character string parameters
*/

PTR MakeNode( char * Word, char * Abbrev, int objtype )
{
  NODE * nodeptr ;  /* Points to allocated node - returned to caller */
  int TextSize;
  char *TextPtr;

  /*     Make the node    */
  if ( ( nodeptr = (PTR) HeapAllocate ( sizeof (NODE) ) ) == NULL )
    {
    return (NULL);
    }

  TextSize = strlen (Word) + strlen (Abbrev) + 2;
  TextPtr = (char *)HeapAllocate (TextSize);
  if (NULL == TextPtr)
    {
    HeapFree (nodeptr);
    return (NULL);
    }

  nodeptr->word = TextPtr;
  strcpy (nodeptr->word, Word);
```

```
    nodeptr->abbrev = nodeptr->word + strlen (Word) + 1;
    strcpy (nodeptr->abbrev, Abbrev);

    /* Initialize type and lengths */
    nodeptr->type = objtype ;
    nodeptr->WordLength = strlen (Word);
    nodeptr->AbbrevLength = strlen (Abbrev);

    nodeptr->UseCount = nodeptr->WarnCount = 0;
    nodeptr->TempFlag = 0;
    return( nodeptr ) ;
}
/*****************************************************************************/
/*
   MakeShortNode() mallocs space for and returns a pointer to a list node,
   initiallizing the fields to the specified character string parameters
*/

char * MakeShortNode( char * Word )
{
   char * nodeptr ;  /* Points to allocated node - returned to caller */

   /*     Make the node     */
   if ( ( nodeptr = (char *) HeapAllocate ( strlen (Word) + 1) ) == NULL )
      {
       return (NULL);
      }

   strcpy (nodeptr, Word);
   return( nodeptr ) ;
}
/*****************************************************************************/
/*
   InitList() initializes the arrays of pointers
*/

void InitList (void)
{
 int i;

 for (i = 0; i < MAX_DEFS; i++)
    Words[i] = Abbreviations[i] = NULL;
 for (i = 0; i < MAX_PREFIXES; i++)
    Prefixes[i] = PreAbbrevs[i] = NULL;
 for (i = 0; i < MAX_SUFFIXES; i++)
    Suffixes[i] = SufAbbrevs[i] = NULL;
 InitHeap ();
}
/*****************************************************************************/
int WordSortFunction ( const void *a, const void *b)
{
 PTR APtr, BPtr;

 APtr = *(PTR *)a;
 BPtr = *(PTR *)b;
 return (stricmp (APtr->word, BPtr->word) );
}
/*****************************************************************************/
int AbbrevSortFunction ( const void *a, const void *b)
{
 PTR APtr, BPtr;
```

```
    APtr = *(PTR *)a;
    BPtr = *(PTR *)b;
    return (stricmp (APtr->abbrev, BPtr->abbrev) );
}
/*************************************************************************/
void ResortWordLists (void)
{
 /*** Sort lists here ***/
 qsort ( (void *) Words, DefCount, sizeof (Words[0]), WordSortFunction);
 qsort ( (void *) Abbreviations, DefCount, sizeof (Abbreviations[0]), AbbrevSortFunction);
}
/*************************************************************************/
void ResortPreSuffixes (void)
{
 /*** Sort prefix lists ***/
 qsort ( (void *) Prefixes, PrefixCount, sizeof (Prefixes[0]), WordSortFunction);
 qsort ( (void *) PreAbbrevs, PrefixCount, sizeof (PreAbbrevs[0]), AbbrevSortFunction);

 /*** Sort suffix lists ***/
 qsort ( (void *) Suffixes, SuffixCount, sizeof (Suffixes[0]), WordSortFunction);
 qsort ( (void *) SufAbbrevs, SuffixCount, sizeof (SufAbbrevs[0]), AbbrevSortFunction);
}
/*************************************************************************/
PTR AddNode (char *Word, char *Abbrev,
             PTR *WordList, PTR *AbbreviationList,
             int *ItemCount, int Type)
{
 PTR Node;

 if ( (!Word[0]) || (!Abbrev[0]) )      /*** If word or abbrev are empty ***/
    return (NULL);
 if ( (NULL == Word) || (NULL == Abbrev) )
    return (NULL);
 Node = MakeNode (Word, Abbrev, Type);
 if (NULL == Node)
    return (NULL);
 if (SUFFIX == Type)
    {
     MaxSuffixLength = max (MaxSuffixLength, strlen (Word));
     MaxSuffixLength = max (MaxSuffixLength, strlen (Abbrev));
    }
 WordList[*ItemCount] = AbbreviationList[*ItemCount] = Node;
 (*ItemCount)++;
 return (Node);
}
/*************************************************************************/
char * AddShortNode (char *Word, char **WordList, int *ItemCount)
{
 char *Node;

 if (!Word[0])                  /*** If word is empty ***/
    return (NULL);
 if (NULL == Word)
    return (NULL);
 Node = MakeShortNode (Word);
 if (NULL == Node)
    return (NULL);
 WordList[*ItemCount] = Node;
 (*ItemCount)++;
 return (Node);
}
/*************************************************************************/
```

```
PTR AddWord (char *Word, char *Abbrev, int Type, int ResortFlag)
{
 PTR Result;

 Result = AddNode (Word, Abbrev, Words, Abbreviations, &DefCount, Type);
 if (ResortFlag)
   ResortWordLists ();
 return (Result);
}
/*****************************************************************************/
void BuildList ( FILE *wordfile )
{
 int type;
 char buffer[ MAXWORDSIZE ] ;
 char word[ MAXWORDSIZE ] ;
 char abbrev[ MAXWORDSIZE ] ;
 char *ChrPtr;

 InitList () ;

 MaxSuffixLength = 0;
 while( fgets( buffer, MAXWORDSIZE, wordfile ) )
   {
    ChrPtr = &buffer[strlen(buffer)-1];
    while (isspace (*ChrPtr))          /*** Skip past trailing spaces ***/
      ChrPtr--;

    /*** Skip to start of number ***/
    while ( (isdigit (*ChrPtr)) && (ChrPtr > buffer) )
      ChrPtr--;

    type = atoi (ChrPtr);
    *ChrPtr = '\0';

    ChrPtr--;                  /*** Points to last char of abbrev ***/
    while (isspace (*ChrPtr))
      ChrPtr--;                /*** Points to last non-space of abbrev ***/
    *(ChrPtr+1) = '\0';        /*** Terminate abbrev ***/
    while (!isspace (*ChrPtr))
      ChrPtr--;                /**** Points to first non-space of abbrev ***/
    strcpy (abbrev, ChrPtr+1);

    while (isspace (*ChrPtr))
      ChrPtr--;                /*** Points to last non-space of expansion ***/
    *(ChrPtr+1) = '\0';
    strcpy (word, buffer);

    if ( (KWORD == type) && (abbrev[0] != '9') )
      AddNode (word, abbrev, Words, Abbreviations, &DefCount, type);
    else if (PREFIX == type)
      AddNode (word, abbrev, Prefixes, PreAbbrevs, &PrefixCount, type);
    else if (SUFFIX == type)
      AddNode (word, abbrev, Suffixes, SufAbbrevs, &SuffixCount, type);
    else
      AddShortNode (word, ShortWords, &ShortCount);
   }
 ResortWordLists ();
 ResortPreSuffixes ();
}
/*****************************************************************************/
int ReadWordList (char *FileName)
{
```

```
    FILE *FPtr;

    if ( NULL == (FPtr = fopen (FileName, "r")) )
       return (0);
    BuildList (FPtr ) ;
    fclose ( FPtr ) ;
    return (1);
    }
    /*************************************************************************/
    int WordFindFunction ( char *a, PTR b)
    {
    if (PREFIX == b->type)
       return (strnicmp (a, b->word, b->WordLength) );
    else      /***** if (KWORD == b->type)  *****/
       return (stricmp (a, b->word));
    }
    /*************************************************************************/
    int AbbrevFindFunction ( char *a, PTR b)
    {
    if (PREFIX == b->type)
       return (strnicmp (a, b->abbrev, b->AbbrevLength) );
    else      /***** if (KWORD == b->type)  *****/
       return (stricmp (a, b->abbrev) );
    }
    /*************************************************************************/
    int SuffixWordFindFunction (char *Key, PTR Node)
    {
    return (stricmp (Key, Node->word));
    }
    /*************************************************************************/
    int SuffixAbbrevFindFunction (char *Key, PTR Node)
    {
    return (stricmp (Key, Node->abbrev));
    }
    /*************************************************************************/
    PTR BinarySearch (char *Key, PTR *List, int Count, int Compare())
    {
    int Low, High, Mid, Comp;

    Low = 0;
    High = Count - 1;
    while (Low <= High)
       {
       Mid = (Low + High) >> 1;
       Comp = Compare (Key, List[Mid]);
       if (Comp < 0)
          High = Mid - 1;
       else if (Comp > 0)
          Low = Mid + 1;
       else
          return (List[Mid]);
       }
    return (NULL);                  /*** Not found ***/
    }
    /*************************************************************************/
    PTR Find ( char *word, int *Type )
    /*** Search list for word as a word, prefix, or suffix, and set Type to
        the appropriate type as found. ***/
    {
    PTR NodePtr;
    int i, Start;
```

```
    NodePtr = NULL;

    /*** Search words ***/
    NodePtr = BinarySearch ( word, Words, DefCount, WordFindFunction );
    if (NodePtr)
      {
       *Type = KWORD;
       return (NodePtr);
      }

    /*** Search for abbreviations ***/
    NodePtr = BinarySearch (word, Abbreviations, DefCount, AbbrevFindFunction );
    if (NodePtr)
      {
       *Type = ABBREV;
       return (NodePtr);
      }

    /*** Search Prefixes ***/
    NodePtr = BinarySearch ( word, Prefixes, PrefixCount, WordFindFunction );
    if (NodePtr)
      {
       *Type = KWORD;
       return (NodePtr);
      }

    /*** Search for abbreviations ***/
    NodePtr = BinarySearch (word, PreAbbrevs, PrefixCount, AbbrevFindFunction );
    if (NodePtr)
      {
       *Type = ABBREV;
       return (NodePtr);
      }

    /*** Search for suffixes ***/
    for (i = min (MaxSuffixLength, strlen (word) ); i > 0; i--)
      {
       Start = strlen (word) - i;
       NodePtr = BinarySearch (&word[Start], Suffixes, SuffixCount, WordFindFunction);
       if (NodePtr)
         {
          *Type = KWORD;
          return (NodePtr);
         }
       NodePtr = BinarySearch (&word[Start], SufAbbrevs, SuffixCount, AbbrevFindFunction);
       if (NodePtr)
         {
          *Type = ABBREV;
          return (NodePtr);
         }
      }

    return (NULL);
    }
    /*****************************************************************************/
    char *FindShort (char *Short)
    {
     int i;
     char *Result;

     Result = NULL;
     for (i = 0; ( (i < ShortCount) && (NULL == Result) ); i++)
```

```
    if (0 == stricmp (Short, ShortWords[i]) )    /*** If a match ***/
      Result = ShortWords[i];

  return (Result);
}
/*************************************************************************/
int DeleteNode (PTR Node)
{
 int i, WordIndex, AbbrevIndex;

 if (NULL == Node)
    return (0);

 WordIndex = AbbrevIndex = -1;
 for (i = 0; (i < DefCount) && ((WordIndex < 0) || (AbbrevIndex < 0)); i++)
    {
     if (Words[i] == Node)
        WordIndex = i;
     if (Abbreviations[i] == Node)
        AbbrevIndex = i;
    }
 if ( (WordIndex < 0) || (AbbrevIndex < 0) )
    return (0);
 Words[WordIndex] = Words[DefCount-1];
 Abbreviations[AbbrevIndex] = Abbreviations[DefCount-1];
 DefCount--;
 HeapFree (Node);
 ResortWordLists ();

 /***** DEBUG - Update MaxSuffixLength - DEBUG *****/

 return (1);
}
/*************************************************************************/
void FormNodeString (PTR Node, char *Msg)
{
 char MaxWord[80], MaxAbbrev[80];

 strncpy (MaxWord, Node->word, 22);
 MaxWord[18] = '\0';

 strncpy (MaxAbbrev, Node->abbrev, 6);
 MaxAbbrev[6] = '\0';

 sprintf (Msg, "%-22s %-6s %4d", MaxWord, MaxAbbrev, Node->UseCount);

#if 0
 switch (Node->type)
    {
     case KWORD:
        strcat (Msg, "WORD");
           break ;

     case PREFIX:
        strcat (Msg, "PREFIX");
           break ;

     case SUFFIX:
        strcat (Msg, "SUFFIX");
           break ;
    }
#endif
```

```
}
/*****************************************************************************/
PTR FormSuggestion (char *Word, char *Suggestion)
{
 char NoVowels[80], *SrcPtr, *DstPtr;
 int i, Type, Unique;
 PTR Test;

 DstPtr = NoVowels;
 SrcPtr = Word;
 *(DstPtr++) = *(SrcPtr++);            /*** Always first character ***/
 while (*SrcPtr)
    {
     if ( (!strchr ("aeiouAEIOU", *SrcPtr) ) && (!ispunct (*SrcPtr)) )
        *(DstPtr++) = *SrcPtr;
     SrcPtr++;
    }
 *DstPtr = '\0';

 Unique = 0;
 i = 2;
 while ( (!Unique) && (i <= strlen (NoVowels) ) )
    {
     strncpy (Suggestion, NoVowels, i);
     Suggestion[i] = '\0';
     i++;

     Test = Find (Suggestion, &Type);
     if ( (NULL == Test) && (NULL == FindShort (Suggestion)) )
        Unique = TRUE;
    }
 if (Unique)
    return (NULL);
 else
    return (Test);
}
/*****************************************************************************/
void ExpandString (char *Str)
{
 char *CharPtr;

 CharPtr = strstr (Str, "[CR]");
 while (CharPtr)
    {
     *(CharPtr++) = '\r';
     memmove (CharPtr, CharPtr+3, strlen (CharPtr+3) + 1);
     CharPtr = strstr (CharPtr, "[CR]");
    }
}
/*****************************************************************************/
/*
  DisplayList() prints the list of word and abbrev fields of all nodes
  in lexicographic order
*/
void DisplayList (int Top, int Left, int Bottom)
{
  int i;
  int Count = 0 ;
  int Key;
  char* prompt2 = "Press [SPACE] to continue." ;
  char LineBuffer[50];
```

```
    PTR *WordList;
    int WordCount;

    print_at (Top+1, Left+1, "Display Words", NORMAL);

#if 0
    print_at (Top+3, Left+1, "Would you like to display: (W)ords,", NORMAL);
    print_at (Top+4, Left+1, "    (P)refixes, or (S)uffixes? :", NORMAL);
    do
      {
      Key = toupper (bioskey(0) & 0xff);
      }
    while ( (Key != 'W') && (Key != 'P') && (Key != 'S') );
#endif
Key = 'W';

    switch ( Key )
    {
      case 'W':
        WordList = Words;
        WordCount = DefCount;
        break ;

      case 'P':
        WordList = Prefixes;
        WordCount = PrefixCount;
        break ;

      case 'S':
        WordList = Suffixes;
        WordCount = SuffixCount;
        break ;

      default:
        WordCount = 0;
        break ;
    }

    ClearPopUp ();
    for (i = 0; i < WordCount; i++)
      {
      FormNodeString (WordList[i], LineBuffer);

      /* Pause every 22 lines of output */
      print_at (Top+1+Count, Left+1, LineBuffer, NORMAL);
      if( !(++Count%(Bottom-Top-3)) )
        {
          print_at (Bottom - 1, Left+1, prompt2, NORMAL ) ;
        bioskey (0);
        ClearPopUp ();
        Count = 0;
        }
    } /* end while */

  if (Count)
    {
    print_at (Bottom - 1, Left+1, prompt2, NORMAL ) ;
    bioskey (0);
    }
} /* end DisplayList() */
/*****************************************************************************/
int WriteWordList (FILE *fPtr, PTR *WordList, int WordCount)
```

```
{
 int i;
 char Str[80];

 for (i = 0; i < WordCount; i++)
    {
     if (EOF == fprintf (fPtr, "%s %s %d\n", WordList[i]->word, WordList[i]->abbrev,
WordList[i]->type))
        return (0);
    }
 for (i = 0; i < ShortCount; i++)
    {
     if (EOF == fprintf (fPtr, "%s %d\n", ShortWords[i], SHORT) )
        return (0);
    }
 return (1);
}
/*****************************************************************************/
int SaveWordList (void)
{
 FILE *fPtr;

 unlink (WORD_BACKUP);                          /*** Delete back up file ***/
 rename (WORD_FILE, WORD_BACKUP);               /*** Save old file ***/
 fPtr = fopen (WORD_FILE, "wt");
 if (NULL == fPtr)
    return (0);
#if 0
 if (!WriteWordList (fPtr, Prefixes, PrefixCount))
    {
     fclose (fPtr);
     return (0);
    }
 if (!WriteWordList (fPtr, Suffixes, SuffixCount))
    {
     fclose (fPtr);
     return (0);
    }
#endif
 if (!WriteWordList (fPtr, Words, DefCount))
    {
     fclose (fPtr);
     return (0);
    }
 fclose (fPtr);
 return (1);
}
/** ab.c  **/
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <process.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "cscreen.h"

#include "ab.h"
#include "list.h"
#include "history.h"
```

```
#define WORD_PROCESSOR "WP.EXE"

/* Function declarations */
void syserr() ;
void Help() ;
void WriteToFile() ;

/**Copyright 1992 Electronic Ink, Inc.***GLOBAL VARIABLES ************/
char *ProgramName = "Abbreviate";
char Word[ MAXWORDSIZE ];
char SearchWord [MAXWORDSIZE];
char *WordPtr = Word;
int FlashDuration = 500;        /** Duration of flash in milliseconds **/
int Active = TRUE;
int MaxSuffixLength = 0;
int DelimitFlag = 0;
int KillFlash = 0;
int KillThreshold = 2;
int HistoryThreshold = 6;
int RemindStart = 3;
int RemindEnd = 7;
int Expanding = 0;
int PreviousKey = 0;
int InPopupFlag = 0;
int CRExpandFlag = TRUE;

unsigned long int Keystrokes = 0;
unsigned long int NonSpaces = 0;
unsigned long int NetKeystrokes = 0;
unsigned long int WordsTyped = 0;
unsigned long int WordsWarned = 0;
unsigned long int WordsExpanded = 0;
unsigned long int PotentialNetKeystrokes = 0;
/**************************TSR Related Variables ***********************/
unsigned ProgramSize = 5500;    /*** Number of paragraphs ***/
unsigned ScanCode = 52;         /*** '.' ***/
unsigned KeyMask = 8;           /*** ALT ***/

unsigned BACK_SCAN = KEY_BACKSPACE;
unsigned ASSIGN_KEY = '=';
unsigned DELIMIT_KEY = '~';
/**************************Display Colors ****************************/
BYTE BORDER_COLOR = 0x9f;
BYTE BACKGD_COLOR = 0x1f;
BYTE HIGHLIGHT_COLOR = 0x71;
/*********************************************************************/
void UnbufferWord (char *Word)
{
 char *Ptr;

 for (Ptr = Word; *Ptr; Ptr++)
   BufferKey (BACK_SCAN);
 BufferKey (BACK_SCAN);
}
/*********************************************************************/
void BufferWord (char *Word)
{
 char *Ptr;

 for (Ptr = Word; *Ptr; Ptr++)
```

```
    BufferKey (*Ptr);
}
/*****************************************************************************/
void ShowAbbreviation (char *Word, PTR Node, char Key)
/*** Remind user that abbreviation exists for Word ***/
{
 char FirstChar, TempStr[MAXWORDSIZE];

 if ( isupper (Word[0]) )
    FirstChar = toupper( Node->abbrev[0] ) ;
 else
    FirstChar = Node->abbrev[0] ;

 if ( (KWORD == Node->type) || (PREFIX == Node->type) )
    {
     /*** Flash abbreviation on screen ***/
     TempStr[0] = FirstChar;
     TempStr[1] = '\0';
     strcat (TempStr, Node->abbrev+1);
     strcat (TempStr, &Word[strlen(Node->word)]);
     FlashWord (TempStr, FlashDuration);
    }
 else if ( SUFFIX == Node->type )
    {
     /*** Flash abbreviation on screen ***/
     strcpy (TempStr, Word);
     TempStr[strlen(Word)-strlen(Node->word)] = '\0';
     strcat (TempStr, Node->abbrev);
     FlashWord (TempStr, FlashDuration);
    }
 Node->WarnCount++;
 if ( (Node->WarnCount >= 3) && (Node->UseCount == 0) && (Node->TempFlag) )
    {
/****     FlashWord ("Deleting. . .", FlashDuration);   ****/
    DeleteNode (Node);
    }
 WordsWarned++;
 PotentialNetKeystrokes += (Node->WordLength - Node->AbbrevLength);
}
/*****************************************************************************/
void ExpandAbbreviation (char *Abbreviation, PTR Node, char Key)
{
 char FirstChar, *CharPtr;
 char NewWord[MAXWORDSIZE];

 strcpy (NewWord, Node->word);
 ExpandString (NewWord);
 if ( isupper(Word[0]) )
    FirstChar = toupper( NewWord[0] ) ;
 else
    FirstChar = NewWord[0] ;

 if (KWORD == Node->type)
    {
     /*** Backspace over abbreviation ***/
     UnbufferWord (Abbreviation);

     /*** Insert word into buffer ***/
     BufferKey (FirstChar);
     BufferWord (&NewWord[1]);

     if (strlen (Abbreviation) > Node->AbbrevLength)
```

```
            BufferWord (&Abbreviation[Node->AbbrevLength]);
        }
    else if ( SUFFIX == Node->type )
        {
        CharPtr = strstr (Abbreviation, Node->abbrev);
        if (CharPtr == Abbreviation)    /*** If word begins with abbreviation ***/
            return;
        if (CharPtr)
            UnbufferWord (CharPtr);      /** Unbuffer suffix **/
        BufferWord (NewWord);            /** Expanded suffix **/
        if (strlen (CharPtr) > Node->AbbrevLength)
            BufferWord (&CharPtr[Node->AbbrevLength]);
        }

    Node->UseCount++;                              /*** Increment use counter ***/
    Expanding = TRUE;
    WordsExpanded++;
    NetKeystrokes += (Node->WordLength - Node->AbbrevLength);
    /*** Display new character ***/
    BufferKey ( Key ) ;
}
/*****************************************************************************/
void ExtractSearchWord (char *Word, char *SearchWord)
{
    int i;

    strcpy (SearchWord, Word);
    i = strlen (SearchWord) - 1;
    while ( (i > 0) && (ispunct(SearchWord[i]) ) )
      SearchWord[i--] = '\0';
}
/*****************************************************************************/
void ShowReminder (char *Word)
{
    char Suggestion[80], Reminder[80];
    PTR Node;

    Node = FormSuggestion (Word, Suggestion);
    if ( (NULL != Node) && (0 == Node->UseCount) && (1 == Node->TempFlag) )
        DeleteNode (Node);
    if (NULL == Node)
        {
    /***     sprintf (Reminder, "Suggestion: Use '%s' as an abbreviation for '%s'.",
                Suggestion, Word);          ***/
        strcpy (Reminder, Suggestion);
        Node = AddWord (Word, Suggestion, KWORD, TRUE);
        if (Node)
            Node->TempFlag = 1;
        }
    else
        sprintf (Reminder, "Suggestion: Create an abbreviation for '%s'.", Word);
    FlashWord (Reminder, FlashDuration);
}
/*****************************************************************************/
int ProcessKey (WORD Key, WORD ShiftMask)
{
    PTR   nodeptr;
    char  *cptr ;
    int   type;
    int   MaskedKey;
    char  Msg[80];
```

```
    MaskedKey = Key & 0xff;

    if ( (!Expanding) && (isprint (MaskedKey)) && (!InPopupFlag) )
      {
      Keystrokes++;
      if (!isspace (MaskedKey))
        NonSpaces++;
      }
    if (BACKSPACE == MaskedKey)
      {
      if ( WordPtr != Word )              /* Modify word: */
        WordPtr-- ;
      }
    else if (DELIMIT_KEY == MaskedKey)
      {
      if (DELIMIT_KEY != (PreviousKey & 0xff))
        return (PreviousKey = Key);
      if (Expanding)
        return (PreviousKey = Key);
      if (!DelimitFlag)
        {           /*** Start of delimited text ***/
        DelimitFlag = 1;
        BufferKey (BACK_SCAN);  /*** Back over delimiter ***/
        BufferKey (BACK_SCAN);  /*** Back over delimiter ***/
        }
      else
        {
        DelimitFlag = 0;
        if (WordPtr != Word)      /*** If not first key after delimiter ***/
          {
          BufferKey (BACK_SCAN);  /*** Back over delimiter ***/
          BufferKey (BACK_SCAN);  /*** Back over delimiter ***/
          }
        }
      WordPtr = Word;
      *WordPtr = EOW;         /*** Reset WordPtr ***/
      }
    else if (ASSIGN_KEY == MaskedKey)
      {
      if (ASSIGN_KEY != (PreviousKey & 0xff))
        return (PreviousKey = Key);
      if (Expanding)
        return (PreviousKey = Key);
      if (DelimitFlag)
        return (PreviousKey = Key);

      *WordPtr = EOW;           /*** Terminate abbreviation ***/
      if (Word == WordPtr)
        return (PreviousKey = Key);
#ifdef _TEST_
      PopUp (ASSIGN_POPUP);
      BufferKey (BACK_SCAN);    /*** Back over ASSIGN_KEY ***/
      UnbufferWord (Word);      /*** Back over abbreviation ***/
      WordPtr = Word;           /*** Clear word buffer ***/
      *WordPtr = EOW;
#else
      BufferKey (BACK_SCAN);    /*** Back over ASSIGN_KEY ***/
      UnbufferWord (Word);      /*** Back over abbreviation ***/
      SetPopFlag (ASSIGN_POPUP);  /*** Enable popup for assignment ***/
#endif
      }
    else if ( (' ' == MaskedKey) ||
```

```
        ( ('\r' == MaskedKey) && (CRExpandFlag) ) )
                                    /* Delimits word */
    {
    if (!Expanding)
       WordsTyped++;
    if (DelimitFlag)
       {
        *WordPtr++ = MaskedKey;                    /* Add char to word: */
        return (PreviousKey = Key);
       }

    *WordPtr = EOW ;                              /* Terminate word */
    WordPtr = Word ;                             /* Reset wordptr */
    ExtractSearchWord (Word, SearchWord);
    nodeptr = Find ( SearchWord, &type ) ; /* Find word or Abbreviation */

    if ( (NULL == nodeptr) && (strlen (SearchWord) >= HistoryThreshold) )
       {       /**** Word not abbrev or word - add to history ****/
        int Count;
        char Msg[80];

        Count = IncHistory (SearchWord);
        if ( (Count == RemindStart) && (!Expanding) )
          ShowReminder (SearchWord);
        WordPtr = Word;
        *WordPtr = EOW;
        return (PreviousKey = Key);
       }
    else if (NULL == nodeptr)
       return (PreviousKey = Key);

    if (KWORD == type)
       {
        if (!Expanding)
          ShowAbbreviation (Word, nodeptr, MaskedKey);
       }
    else if ( ABBREV == type )
       {
        if (!Expanding)
          ExpandAbbreviation (Word, nodeptr, MaskedKey);
       }
    }   /*** else if isspace. . . ***/
  else if (isprint (MaskedKey))
    *WordPtr++ = MaskedKey;                       /* Add char to word: */
  else
    {
     *WordPtr = EOW ;                             /* Terminate word */
     WordPtr = Word ;                            /* Reset wordptr */
    }
  return (PreviousKey = Key);
}
/*****************************************************************************/
/* Function to print error messages */

void syserr( msg )
char* msg ;
{
  extern int errno, sys_nerr ;
  extern char *sys_errlist[] ;

  fprintf( stderr, "ERROR: %s (%d", msg, errno ) ;

  if (  errno > 0  &&  errno < sys_nerr  )
```

```
        fprintf( stderr, "; %s) \n", sys_errlist[errno] ) ;
      else
        fprintf( stderr, ")\n" ) ;

    exit( 1 ) ;

}
/*************************************************************************/
int Initialize (void)
{
 initdisplay ();                    /*** Initialize display routines ***/

 InitHistory ();
 if (!ReadWordList (WORD_FILE))
     syserr ("Error: Word list not found.");

 WordPtr = Word;
 *WordPtr = EOW;          /*** Reset WordPtr ***/
 return (1);
}
/*************************************************************************/
void interrupt iFunc (bp, di, si, ds, es, dx, cx, bx, ax)
{
}
/*************************************************************************/
int ExecuteWordProcessor (char *WordProcessor, char **ArgList)
/*** Executes Word processor, with arguments in char * array ArgList.
     If successful, function returns non-zero value.  If not successful,
     an error message is printed, and zero is returned. ***/
{
 int Result;

 Result = spawnvp (P_WAIT, WordProcessor, ArgList);
 if (Result)
    {                   /*** Error ***/
     char Msg[80];

     sprintf (Msg, "Unable to run %s.", WordProcessor);
     perror (Msg);
    }
 return (0 == Result);
}
/*************************************************************************/
void main (int argc, char **argv)
{
 int iVect;
 char c, *Ptr;

 printf ("\n%s, Copyright 1992, Electronic Ink\n", ProgramName);
 Initialize ();

 InitResident ();        /*** Doesn't return ***/

 ExecuteWordProcessor (WORD_PROCESSOR, &argv[1]);

 Terminate ();
} /* end of main() */
/** addhist.c  **/
/******Copyright 1992 Electronic Ink, Inc.*********************************/
HISTORY_PTR AddHistory (char *Word)
{
 HISTORY_PTR NewHist, HistPtr;
```

```
int CompResult, Done;

/*** Allocate space for node ***/
NewHist = HeapAllocate (sizeof (HISTORY_NODE) );
if (NULL == NewHist)
  return (NULL);

/*** Allocate space for word ***/
NewHist->Word = HeapAllocate (strlen (Word) + 1);

if (NULL == NewHist->Word)
  {
   HeapFree ((void *)NewHist);
   return (NULL);
  }
strcpy (NewHist->Word, Word);
NewHist->UseCount = 0;
NewHist->Left = NULL;
NewHist->Right = NULL;

/**** Add node to tree ****/
if (NULL == HistoryRoot)
  {
   HistoryRoot = NewHist;
   return (NewHist);
  }

HistPtr = HistoryRoot;
Done = 0;
while (!Done)
  {
   CompResult = stricmp (Word, HistPtr->Word);
   if (CompResult > 0)
     {
      if (HistPtr->Right)
        HistPtr = HistPtr->Right;
      else
        {
         HistPtr->Right = NewHist;
         Done = 1;                    /*** Exit loop ***/
        }
     }
   else if (CompResult < 0)
     {
      if (HistPtr->Left)
        HistPtr = HistPtr->Left;
      else
        {
         HistPtr->Left = NewHist;
         Done = 1;
        }
     }
   else                               /*** Matching Node was found ***/
     {
      HeapFree (NewHist->Word);
      HeapFree ((void *)NewHist);
      NewHist = NULL;
      Done = 1;
     }
  }
 return (NewHist);
}
```

What is claimed is:

1. An input system comprising
a drawing tool having an input connected to an optical sensor;
a controller connected to said optical sensor;
a host processor;
a writing surface having array of pixels connected to said host processor;
a receiving means connected to said host; and
an output device connected to said controller and coupled to said receiver, whereby said system can sense information at the input and communicate that information to the receiving means.

2. The input system of claim 1 wherein said drawing tool comprises a plurality of optical sensors.

3. The input system of claim 2 wherein said drawing tool further comprises a plurality of light guides, each of which is connected to one of said optical sensors.

4. The input system of claim 1 wherein said drawing tool comprises a pen having two light guides and two optical sensors.

5. The input system of claim 1 wherein said drawing tool comprises a brush having a plurality of optical sensor elements and a plurality of light guides, each of which is connected to at least one of said optical sensor elements.

6. The input system of claim 5 wherein one of said light guides comprises a fiber optical element coated with conductive material.

7. The input system of claim 5 further comprising a conductive strand intermixed with said plurality of light guides.

8. The input system of claim 1 wherein said receiving means comprises
a relatively transparent, planar, conducting surface;
sensor means connected to said conducting surface;
signals from said output device for communication to said receiver; and
means for ratiometric measurement of said signals from said output device.

9. The input system of claim 1 further comprising means to identify a pixel closest to said drawing tool.

10. The input system of claim 1 further comprising means to transfer information to said host processor regarding said drawing tool in relation to said writing surface, said information selected from the group consisting of position, angle and pressure.

11. A process for accelerating the input of information and graphic elements by use of abbreviated strokes of an input system comprising:

taking an input character from said input system, said input character in the form of a keyboard input, a menu selected character or a stroke pattern which has been preprocessed and identified as a selected character or an arbitrary stroke pattern;

testing said input character and

if said input character is not a special character then putting said character into a word buffer;

if said input character is a special character and is a word terminator such as a space, comma or other punctuation, then adding said special character to said word buffer then retrieving the contents of said word buffer as text and

testing said text against a list of known abbreviations and, if such an abbreviation is found, then expanding said abbreviated text and outputting said expanded text, and if said text is all capitalized, then outputting said expanded text as all capitals,

but if no abbreviation is found, then examining the first letters of said text and comparing said first letters against a table of known prefixes and, if a known prefix abbreviation is found, then expanding said prefix abbreviation and then examining the last letters of said text and, if the last letters are capitalized, comparing said letter against a list of available routines for expanding suffixes, and, if said last letter is not capitalized, comparing said last letters against a table of known suffixes and, if a known suffix abbreviation is found, then expanding said suffix abbreviation and outputting said expanded text or, if the last letter is capitalized, then expanding the abbreviation,

if said input character is a special character and is an instruction code, then proceeding according to the nature of said instruction.

12. The process of claim 11 wherein said special character is an instruction to read the preceding characters as an abbreviation, said process further comprising

storing the preceding characters as an entry in a list of known abbreviations;

acquiring subsequent characters as expanded text, regardless of word terminators and punctuation, until finding a second special character which indicates the end of said expanded text and entering said expanded text into said list of known abbreviations, correlated with said entry of said preceding characters.

13. The process of claim 11 wherein said special character is an instruction to not expand the following or preceding input text.

14. The process of claim 11 further comprising storing in a history table each word which is output without being expanded.

15. The process of claim 14 further comprising retrieving a word from said history table and deriving a proposed abbreviation for said retrieved word, said abbreviation consisting of either unique characters a unique subset of strokes not already being used in said list of known abbreviations.

16. A drawing tool input system in which a drawing tool comprises a multi-point light detecting means to establish drawing tool position.

17. The drawing tool input system of claim 16 in which said drawing tool further comprises a means to establish drawing tool angle.

18. The drawing tool input system of claim 16 in which said drawing tool further comprises a means to establish drawing tool pressure.

19. A drawing tool input system comprising individual identifying data are encoded in a drawing tool means.

20. The method of claim 19 wherein said individual identifying data comprise debit/credit information.

21. The method of claim 19 wherein said individual identifying data comprises personal information selected from the group consisting of an individual's name, home address, business address, phone number, taxpayer ientification number, passport number, blood type, bank account number, name of an individual to contact in case of an emergency and other personal information.

22. The method of claim 21 further comprising a method of accessing said individual identifying data by a host computer, wherein said individual identifying data cannot be modified or accessed by a typical user but can be accessed by the host computer and read out or, under selected conditions, modified.

23. A method of identifying the approximate position of a pointing device in two dimensions, said method comprising:

providing a generally flat surface subdivided into a plurality of small regions, substantially each of which can be activated;

providing a pointing device sensitive to whether or not a first one of said small region is activated, wherein said pointing device can be tested to determine whether it is pointing to an active small region;

activating a selected plurality of said small regions; and

testing said pointing device to determine whether said first small region is within said selected plurality of small regions.

24. A method of identifying the approximate position of a pointing device in two dimensions, said method comprising:

providing a generally flat surface;

providing a pointing device;

providing ratiometric detection means sensitive to the position of said pointing device; and

evaluating said ratiometric detection means to determine the approximate position of said pointing device.

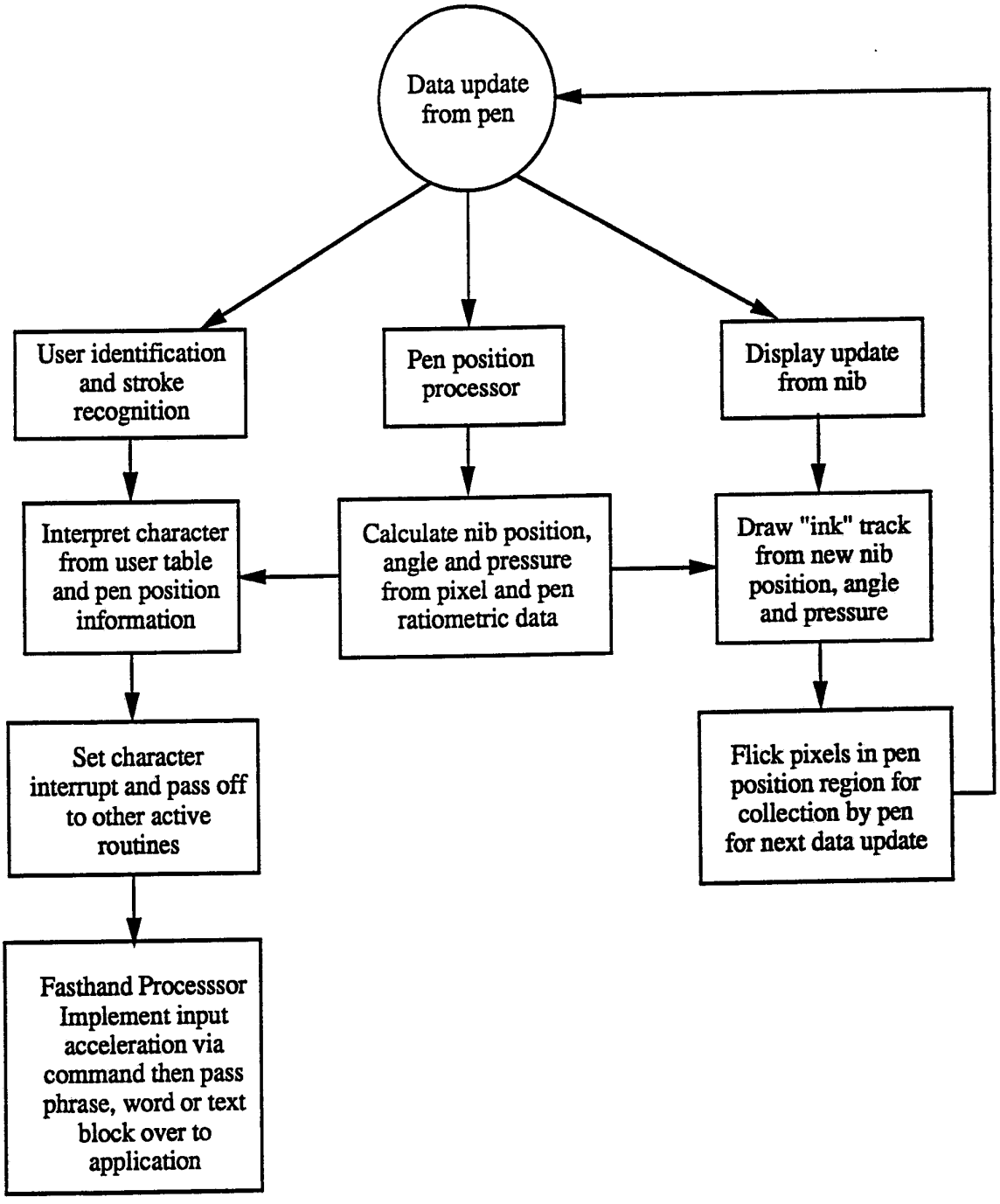

Data update from pen
User identification and stroke recognition
Pen position processor
Display update from nib
Interpret character from user table and pen position information
Calculate nib position, angle and pressure from pixel and pen ratiometric data
Draw "ink" track from new nib position, angle and pressure
Set character interrupt and pass off to other active routines
Flick pixels in pen position region for collection by pen for next data update
Fasthand Processsor Implement input acceleration via command then pass phrase, word or text block over to application

Figure 1

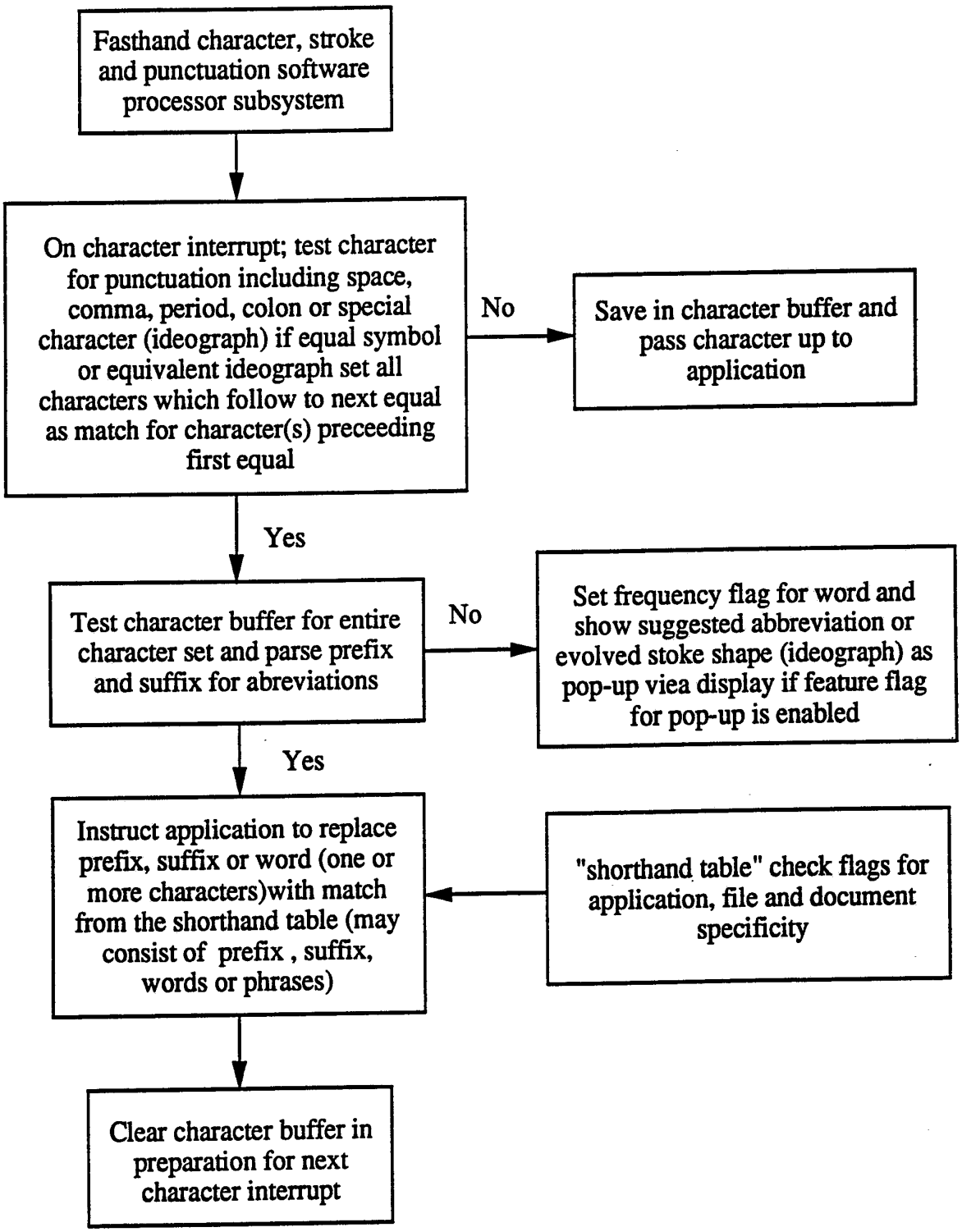

Fasthand character, stroke and punctuation software processor subsystem
On character interrupt; test character for punctuation including space, comma, period, colon or special character (ideograph) if equal symbol or equivalent ideograph set all characters which follow to next equal as match for character(s) preceeding first equal
No
Save in character buffer and pass character up to application
Yes
Test character buffer for entire character set and parse prefix and suffix for abreviations
No
Set frequency flag for word and show suggested abbreviation or evolved stoke shape (ideograph) as pop-up viea display if feature flag for pop-up is enabled
Yes
Instruct application to replace prefix, suffix or word (one or more characters)with match from the shorthand table (may consist of prefix , suffix, words or phrases)
"shorthand table" check flags for application, file and document specificity
Clear character buffer in preparation for next character interrupt

Figure 2

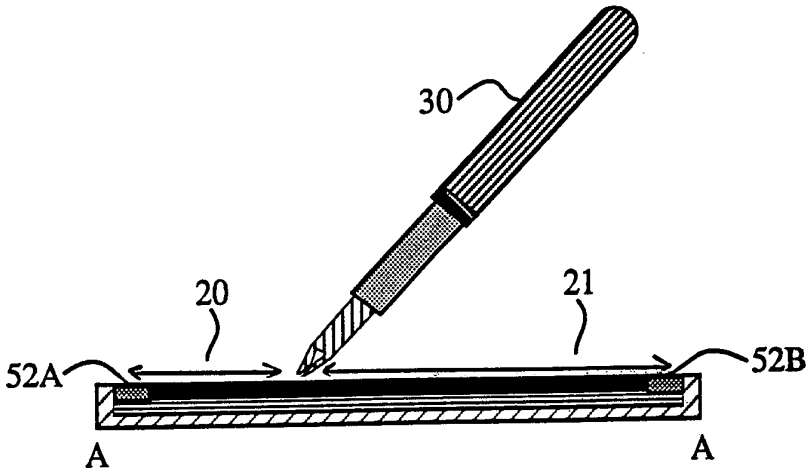


Figure 3 AA

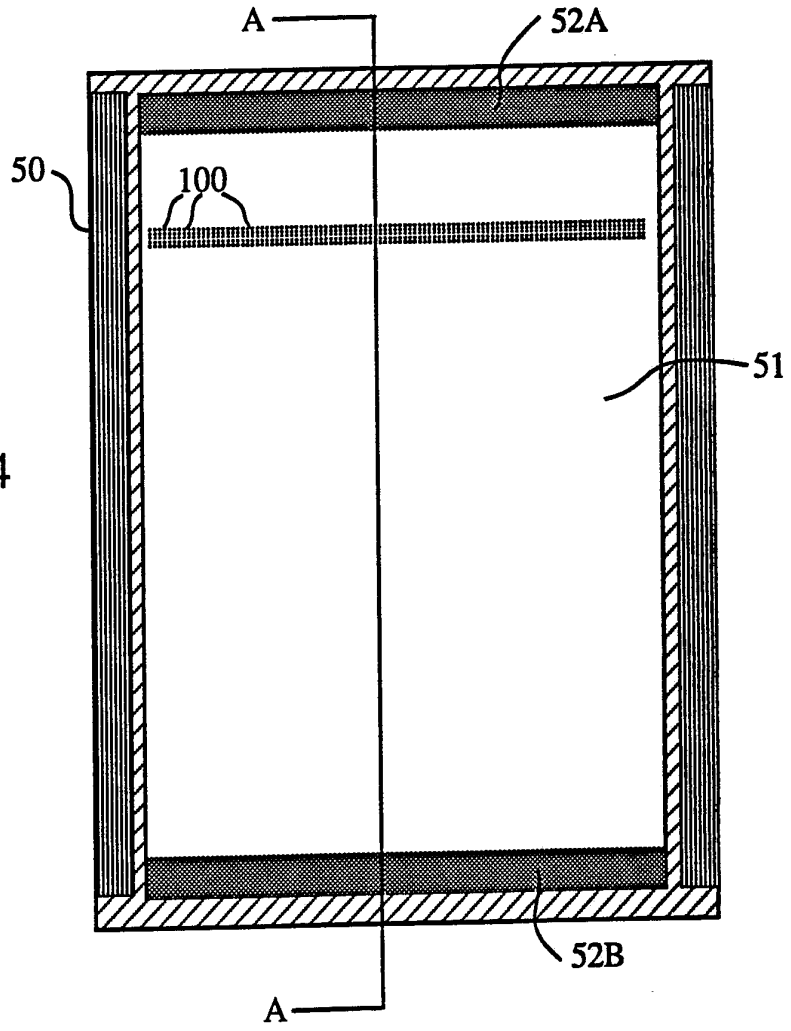


Figure 4

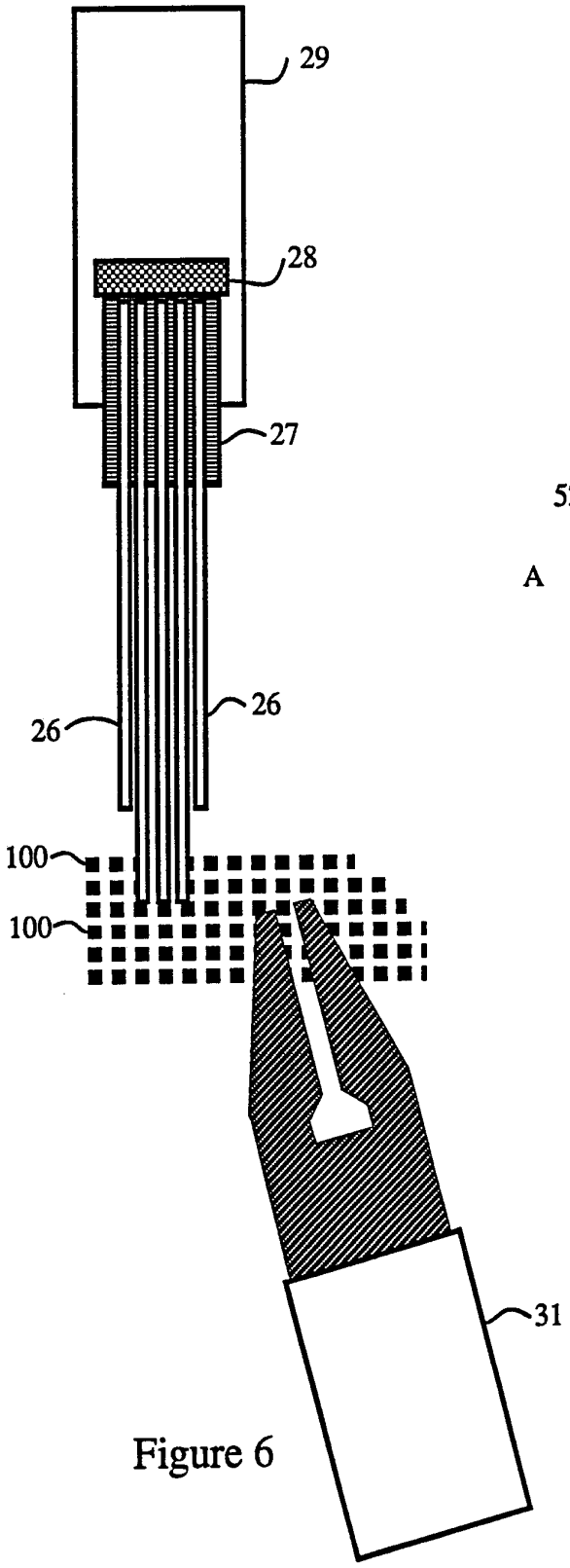


Figure 6

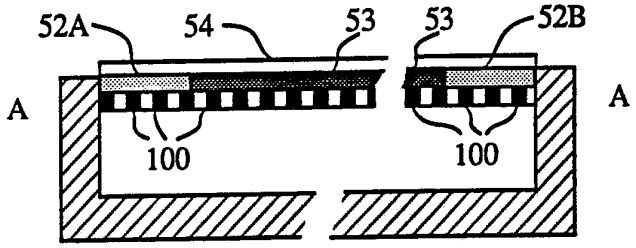


Figure 5

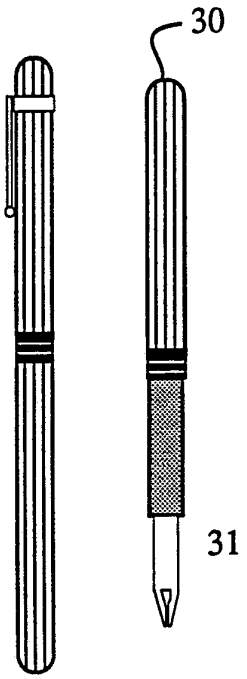


Figure 7A

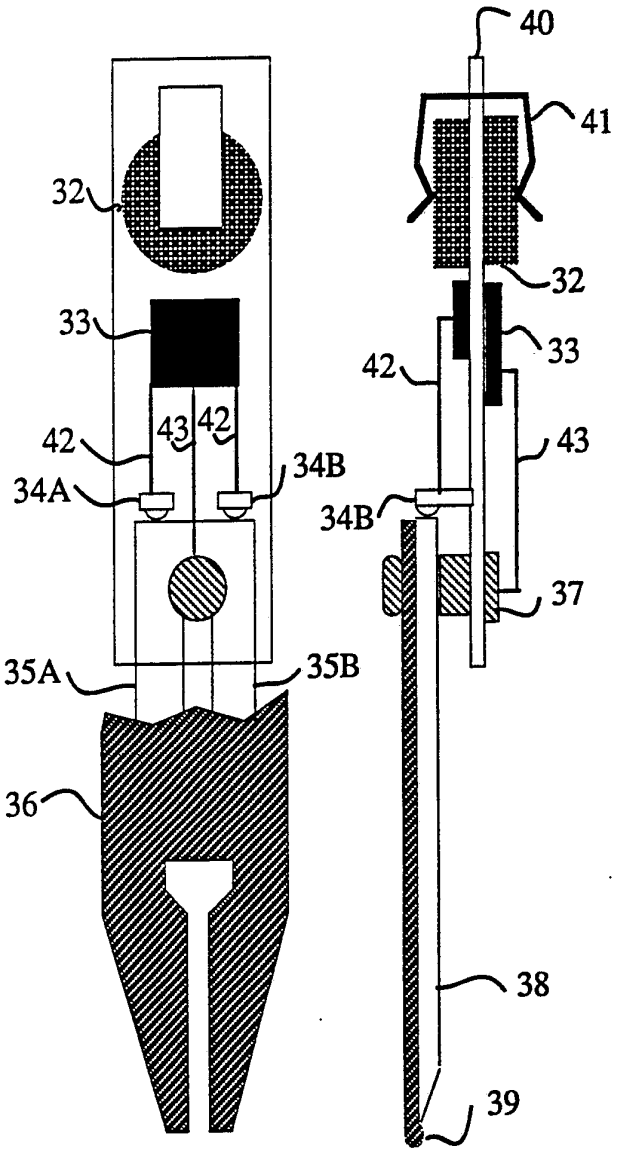


Figure 7B

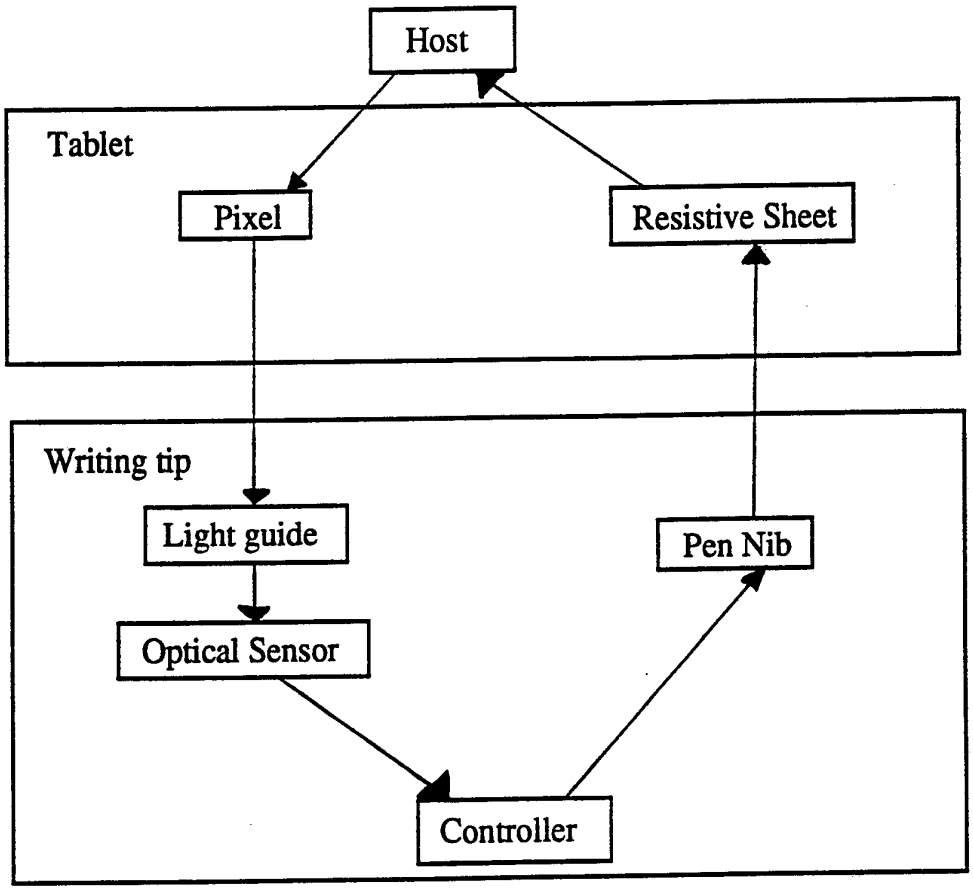

Host
Tablet
Pixel
Resistive Sheet
Writing tip
Light guide
Pen Nib
Optical Sensor
Controller

Figure 8

INTERNATIONAL SEARCH REPORT

PCT/US92/09123

A. CLASSIFICATION OF SUBJECT MATTER

IPC(5) :G09G 3/02
US CL :340/706,707,709 178/18, 341/29
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 340/706,707,709 178/18, 341/29

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	US,A, 5,051,736 (Bennett et al.) 24 September 1991 See Fig. 1 (1,3,4,10), Fig. 2 (14).	1,15 2-9,16-18
X Y	GB,A, 2,193,827 (Sklarew) 17 February 1988 See Col. 1, lines 14-24, Col. 2, lines 34-55.	19 20-24
Y	US,A, 4,891,786 (Goldwasser) 02 January 1990 See Fig. 1 (10), Fig. 7E (172,176,180).	10-14
Y	US,A, 4,922,236 (Heady) 01 May 1990 See Fig. 1, (23,3).	2-4,6-9

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 04 JANUARY 1993	Date of mailing of the international search report 08FEB 1993
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. NOT APPLICABLE	Authorized officer AMARE MENGISTU Telephone No. (703) 305-4880

Form PCT/ISA/210 (second sheet)(July 1992)★

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US92/09123

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US,A, 4,078,151 (McNeary) 07 March 1978 See Col. 1, lines 14-24.	5,16,17
Y	US,A, 4,947,156 (Sato et al.) 07 August 1990 See Col. 2, lines 34-55, Fig. 5.	5,16,17