ELECTROSTATIC PLOTTING
PLOTTING ON AN ELECTROSTATIC PRINTER/ PLOTTER

by

CHRISTOPHER A. BRYCE, B.Sc.

A Project
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree
Master of Science

McMaster University
April 1975
TITLE: Plotting on an Electrostatic Printer/Plotter.

AUTHOR: Christopher A. Pryce, B.Sc. (McMaster)

SUPERVISOR: Professor D. Kenworthy

NUMBER OF PAGES: viii, 199
ABSTRACT

A survey of printers and plotters is given, and in particular the operation and capabilities of electrostatic printer/plotters is discussed. An implementation of a plotting system which plots on an electrostatic printer/plotter is presented. This plotting system is designed to be compatible with the Benson-Lehner plotting system. The standard "PLOT" routine is replaced by a two pass system, which generates plots on the printer/plotter. In addition to the plotting system, an implementation of a graph utility is presented. This utility provides a single one pass system that plots one or more functions (where the function has one value for each value of x.)
ACKNOWLEDGEMENTS

I wish to thank my supervisor Dr. D. Kenworthy for his assistance during the work on this project. I would especially like to thank Dr. D. Kenworthy and Professor K. Redish for their invaluable assistance and time during the writing of this project. Acknowledgements are also due for assistance at various times to Dr. D. Wood, Dr. N. Solntseff, G. Hicks, and M. Relec.

Finally, I would like to thank Mrs. Margaret Harris for her excellent typing.
# Table of Contents

## Chapter 1 -- Introduction

1.1. General Discussion of Printers and Plotters (Survey) .................................................. 1
1.2. Specific Discussion of Electrostatic Printer/Plotters ..................................................... 4
1.3. Description of the Implementation Environment .............................................................. 8
1.4. Overview of the Project ........................................................................................................ 9
1.4.1. Plot Package .................................................................................................................. 10
1.4.2. Graph Utility ................................................................................................................ 11

## Chapter 2 -- Theory

2.1. General Description of the Method Used ............................................................................ 13
2.2. Definitions and Terminology ............................................................................................. 16
2.3. Detailed Description of Method Used in Plot Package ....................................................... 22
2.3.1. Description of PASS 1 (PLOT) (SAVING PROCESS) .................................................... 22
2.3.2. Description of PASS 2 (PRINT) (PLOTTING PROCESS) ............................................ 26
2.4. Description of Theory Used in Graph Utility ..................................................................... 30

## Chapter 3 -- Implementation Description

3.1. General Description and Breakdown of the Plot Package .................................................. 32
3.2. "PLOT" Routine .................................................................................................................. 35
3.3. "INVRT" Routine ............................................................................................................... 38
3.4. "PRINT" Routine .............................................................................................................. 38
3.5. "ORDER" Routine ............................................................................................................. 39
3.6. "INSRT" Routine .............................................................................................................. 40
3.7. "BFOUT" Routine .............................................................................................................. 40
3.8. Reason for Version II .......................................................................................................... 41
3.9. "PLOT" Routine (Version II) ............................................................................................. 46
3.10. "DSKIO" Package ............................................................................................................ 46
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Schematic of an EPP</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>Character Fonts</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>Table of Different EPP Specifications</td>
<td>7</td>
</tr>
<tr>
<td>1.4</td>
<td>Typical Plotting Program Organization</td>
<td>10</td>
</tr>
<tr>
<td>2.1</td>
<td>A Point in XxY Plane</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>Lines in XxY Plane</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>A Straight Line</td>
<td>18</td>
</tr>
<tr>
<td>2.4</td>
<td>A Straight Line Segment</td>
<td>18</td>
</tr>
<tr>
<td>2.5</td>
<td>Contiguous Line Segments</td>
<td>19</td>
</tr>
<tr>
<td>2.6</td>
<td>N-Tuple of Line Segments (String)</td>
<td>19</td>
</tr>
<tr>
<td>2.7</td>
<td>Monotonic Strings</td>
<td>20</td>
</tr>
<tr>
<td>2.8</td>
<td>List (of Strings)</td>
<td>21</td>
</tr>
<tr>
<td>2.9</td>
<td>Possible Line Segments</td>
<td>23</td>
</tr>
<tr>
<td>2.10</td>
<td>3 Pen Situations</td>
<td>24</td>
</tr>
<tr>
<td>2.11</td>
<td>Line Segments Over 7 Plot-Lines</td>
<td>26</td>
</tr>
<tr>
<td>2.12</td>
<td>Logical Organization of Strings</td>
<td>27</td>
</tr>
<tr>
<td>2.13</td>
<td>X Value and Line Segments</td>
<td>28</td>
</tr>
<tr>
<td>2.14</td>
<td>Graphing Using 3 Consecutive Y-Values</td>
<td>31</td>
</tr>
<tr>
<td>3.1</td>
<td>Pictorial View of Plot Package</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>Table of Pen Calls</td>
<td>36</td>
</tr>
<tr>
<td>3.3</td>
<td>TAGS( ) and LIST( ) STRUCTURE</td>
<td>37</td>
</tr>
<tr>
<td>3.4</td>
<td>Version II of PLOT Package</td>
<td>44, 45</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

1.1 General Discussion of Printers and Plotters (Survey)

Computers communicate with users (humans) in human readable form primarily by means of the following three methods: line printer output, plotter output and teletype or CRT output. Of these methods, hardcopy communication is provided by the line printer, the plotter, and the teletype. What computers communicate comprises the following: the errors that the computer detects in the user's instructions (in Job Control Language, in the program, or in the job organization) and the results that the computer produces as a result of those instructions (the program listings and the results of the user's program). As a rule, the computer is required to produce large quantities of output, relative to the size of the input. Either by explicit user instructions, or by installation defaults, the computer usually produces program listings, quite often with cross-references, loader maps, results of user's programs, and a listing of the job control language. Unfortunately, most of this output gets wasted, as only a small portion actually will be used. It is all potentially useful, but the use of any given portion will depend upon the success or lack of success of the program. As a result, the only communication device capable of keeping up with the volume of output normally produced is the line printer. Plotters and teletypes provide special purpose output abilities. With the plotter, pictorial representation, when properly used, can say more than thousands of pages filled with relevant numbers. Plotters can be used in drafting and
providing the reliability and precision necessary in today's technological society, and can also be used to display results from scientific investigation. Teletypes provide an interactive ability that allows the user to dynamically communicate with, and interact with, a computer. Generally speaking, plotters and teletypes have a limited speed, which would choke or stifle the potential output of most computers if used for the same purposes as the line printer.

The line printers available today vary greatly. There are small line printers designed to take output from a CRT and to print the screen buffer onto paper at about 100 or so lines per minute. Then there are large line printers, typically used on large computers, which print at a rate of 1000 to 1200 lines per minute. Within this range lie many printers of different sizes, with different print speeds, and to date these printers have mainly been characterized by being of mechanical type. A mechanical printer (or an impact printer) usually consists of a typewriter like ribbon together with a rotating drum, or a moving chain, and prints by striking the paper when the desired print character comes into position.

Most large computers support 2 or 3 large 1000 lines/minute printers and sometimes several additional smaller printers (usually remote from the computer). These mechanical (impact) printers usually have 64 or 96 characters in their character sets, and in some cases this could be a restriction.

Other methods of printing include spitting chemicals onto special paper, burning special paper (with heat), a photographic type of printing, and electrostatic printing. Generally speaking the paper in each case is more expensive, but compared to impact printers, breakdown is usually much
more infrequent. Print quality varies generally from worse than to much better than an impact printer. Printing speed varies from less than 1000 lines/minute to 5000 lines/minute, and the character sets on non-impact printers can include up to 256 characters. Special type-sets such as Roman or Helvetian are sometimes available, and often non-standard character sets, such as French, German, Greek or Japanese, can be software generated. Non-standard character sets will add to computer processing time.

Plotting too has usually been a mechanical process. Mechanical plotters divide basically into two types, flatbed plotters, and drum type plotters. Flatbed plotters may be small but most of them are large, of the order of 2 feet by 3 feet. These are mainly used in drafting systems to provide very accurate plots, with the usual step size or precision being about 0.001 inches. Flatbed plotters plot by moving a pen on a fixed piece of paper. Drum type plotters can also be either small or large, with the physical size of the plotter usually related to the precision. These plotters plot by moving the paper lengthwise, back and forth, and moving the pen from side to side. The paper for these plotters comes, usually, in rolls of widths 11, 30, or 36 inches, with a virtually infinite length (usually 125 - 150 feet, with some up to 367 feet). In general these plotters provide very accurate plots with a precision of 0.0025 to 0.005 inches, although some only have a precision of about 0.01 inches.

As in printing, plotting can be done by non-mechanical means also, by using the same techniques as used in the non-impact printing. In fact many devices are both printers and plotters. In such a printer/plotter, printing is either a special case of plotting, with software generated characters, or in addition to the plotting capability, a line buffer is
provided together with a character generator (which relieves the computer from generating characters). Plotting is accomplished by setting points into an output line of densely packed points (with densities varying from 72.5 points per inch to 200 points per inch). Then successive lines of points, just as tightly packed (as in the line), are plotted. Thus in a manner similar to a television scan, an image or plot can be made.

1.2. Specific Discussion of Electrostatic Printer/Plotters.

To obtain the data and specifications of both the printer/plotters in this section and the plotters in the previous section a partial survey was made using available sales brochures and information. The survey was not very exhaustive, and obviously claims and specifications given in a sales brochure must be weighed accordingly.

The Electrostatic Printer/Plotter (EPP) is a versatile device. Most often the EPP comes as a printer/plotter, but occasionally just a printer or just a plotter, is provided. It varies in width from 8 inches (80 characters) to about 22 inches (264 characters), but most often the width is either 8 inches (80 characters) or 11 inches (132 characters).

As seen in figure 1.1., the EPP works in an efficient, simple manner. Paper, either roll paper or fan-fold paper, is passed between the writing head and the rear electrode. The writing head extends across the page and consists of densely packed writing nibs. These nibs individually, upon command, create minute electrostatic dots on the paper. As the paper advances, a liquid toner is applied to produce permanently visible points. Then, by passing air over the paper as it advances, the paper is dried and is ready to handle upon being stacked. Print characters (hardware)
Figure 1.1. Schematic of an EPP

Figure 1.2. Character Fonts
are produced in a matrix, (also called a grid or font), varying from 5 points x 7 points to 16 points x 20 points as illustrated in figure 1.2. Sometimes special type fonts are provided (for example Roman, Helvetian, French, German . . . ). Printing speed varies from 190 lines/minute to 5000 lines/minute depending on the page width and nib density. In the table (figure 1.3.) are included several different machines, but it is evident that the more character positions per line, and the more dense the nibs, the slower the print speed. The restrictions here appear to be electronic. Since one printer is able physically to print at 5000 lines/minute, in theory the rest should be capable of it. The problem would seem to be one of electronic buffering or high speed communication lines. It would not be surprising to see a 14 inch printer with 200 dots per inch printing in the 2000 to 5000 lines/minute range in the near future. The biggest problem at this point would be paper folding and developing high speed operators to remove the output as it is available.

Plotting is theoretically as fast as printing. Sometimes the computer will not be able to supply the information fast enough for maximum plot speed, but if the information were available, the plotter could take it. Comparing an EPP with other plotters, as far as plotting speed is concerned, is not fair. These plotters can produce an 8-1/2 x 11 page of plot, in theory, from 1 second for the fastest to about 24 seconds for the slowest. On the average this theoretical plot would take about 7 seconds. This time would be completely independent of what is to be plotted. In practice on a small mini computer this time will run up to 20 to 30 seconds. These times would compare with conventional plotter speeds running from 3 to 20 minutes. Also where, conventional plotters require
<table>
<thead>
<tr>
<th>Paper Width</th>
<th>Character Width</th>
<th>Point Density (pts/inch)</th>
<th>Print Speed (lpm)</th>
<th>Plot Speed (ips)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 inch (7.75)</td>
<td>80</td>
<td>72.5</td>
<td>600</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>80</td>
<td>1000(a)/</td>
<td>2.1(a)/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5000(s)</td>
<td>10.4(s)</td>
</tr>
<tr>
<td>11 inch (10.5)</td>
<td>132</td>
<td>100</td>
<td>500</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>160</td>
<td>300</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>500</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>1200</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>200</td>
<td>1.65</td>
</tr>
<tr>
<td>20 inch (18)</td>
<td>180</td>
<td>160</td>
<td>190</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>232</td>
<td>100</td>
<td>300</td>
<td>0.75</td>
</tr>
<tr>
<td>21 inch</td>
<td>232</td>
<td>100</td>
<td>1200</td>
<td>3.0</td>
</tr>
</tbody>
</table>

(a) - asynchronous  (s) - synchronous

Figure 1.3. Table of Different EPP Specifications
an operator to reset the pen and paper for the next user, on the EPP the output gets stacked just like output from a line printer. The precision or accuracy of the EPP ranges from 0.014 inches to 0.005 inches, which compares unfavorably with the precision of most conventional plotters (0.005 inches to 0.0025 inches). As far as precision or step size is concerned, to date, the EPP is just starting to approach that of conventional plotters. There are other things to consider though. Most conventional plotters are incremental and if the pen gets nudged out of line (for example, someone walks too close to it) all successive plot movements are out by a constant amount, and this type of error is cumulative. Secondly if the user, by mistake, takes the pen off the edge of the paper the rest of the plot is lost, which could have been informative as to what went wrong. On an EPP there is no pen to knock off line and if the user goes off the page, the software is usually set up to ignore these points and simply to show the pen when it comes back onto the page, if it does.

All in all the conventional plotter, for the moment, has its place in plotting just for the better possible resolution, but where regular resolution (0.005 inches) is good enough the electrostatic plotter is by far the best.

1.3. Description of the Implementation Environment

This project is being implemented using a Versatec Matrix Printer/Plotter 200A. The 200A can use 8.5 x 11 inch fan fold paper or 8.5 inches x 500 feet rolls. Primarily fan fold paper is used. The resolution is 72.5 dots per inch both vertically and horizontally. There are 560 writing nibs with a possible plot width of 7.75 inches. In printing mode,
The printing speed is 600 lines per minute, with a corresponding plotting speed of 1.6 inches of plot per second.

As the plotting width is only 7.75 inches compared to 10.95 inches for the Benson-Lehner on the CDC6400, we could either leave everything in inches and let the user make his own allowances or scale everything down so that the user can ignore the difference. At present, everything is left in inches, but eventually a switch will be built into the plot package to allow either inches or scaled (10.95 to 7.75).

The printer/plotter is attached to a Hewlett-Packard 2100A which is operating under the DOSM operating system. The project software was written primarily in FORTRAN, and partially in HP ASSEMBLER, but an all FORTRAN version is possible. The standard HP FORTRAN IV compiler and the standard HP ASSEMBLER were used.

1.4. Overview of the Project

The objectives of the project are to:

1) develop a plot package starting at routine PLOT which is capable of plotting on an electrostatic printer/plotter, this PLOT routine is written to be compatible with the Benson-Lehner plotting system and capable of supporting the parts of the Benson-Lehner system which call PLOT.

2) develop a graph routine capable of taking several functions and creating a graph or plot, as simply as possible, over a desired range, and with a desired scale.
1.4.1. **Plot Package**

The objectives were chosen for several reasons. First of all PLOT is a key routine in any plotting system, and is the routine through which all plotter instructions must pass. Typically the user writes his program, and through either direct or indirect calls to PLOT, instructs the plotter on how to create the plot. Routines such as LINE, NUMBER, SYMBOL, etc. ..., allow the user easily to produce lines, write numbers, write characters, etc. ..., and these routines all perform their functions by making calls to PLOT. In the diagram below this typical program organization is illustrated.

![Diagram showing the flow of information from the user's program through PLOT to the plotter output.](image)

**figure 1.4. Typical Plotting Program Organization**

The PLOT routine was chosen as the place to break in because this allows the standard Plot utilities to be used with the EPP. Therefore the
user need not change any of his program in order to plot on the electrostatic plotter, as long as the electrostatic PLOT routine is substituted for the standard PLOT package. Another reason for starting at routine PLOT, is because the differences between plotting systems are minimal in routine PLOT and to switch the package to another plotting system is easiest at this routine. These are the basic reasons why the plotting system starts at routine PLOT.

The reasons for choosing the Benson-Lehner system for a model, is that this is the plotting system on the CDC6400 at McMaster. This means there are practical reasons to choose this system such as:

1) The availability of information, both user information and software information, on this plotting system;
2) plotter library routines could be easily converted to a HP2160 (after the PLOT package is implemented);
3) users could run on one or the other interchangeably;
4) during development of the plotter package it could be tested, on the CDC6400, for compatibility with special plot routines (LINE, GREEK, NUMBER, ...)

For these overwhelming reasons this project was designed to be compatible with the Benson-Lehner plotting system.

1.4.2. Graph Utility

This utility virtually created itself, during the experimental period, where simple plotter output was being generated. This output was an attempt to determine just how the printer/plotter actually plots, and just how best to do the plotting. Upon mastering the ability to generate
simple plots, and how best to buffer the plotting process, it became evident that a function with one and only one Y-value for each X-value could be easily plotted as soon as it became available. Thus functions with this restriction can be easily plotted in the XxY coordinate plane. Note that this is all that the utility is intended to do. With little difficulty these functions were plotted, and gradually extras such as borders, reporting limits, and plotting axes were added to the graph utility.

What evolved was a routine, GRAPH, with the ability to plot 1 or more (up to 10) functions over a specified range of x and y, and with scales determined by the user. The author believes that this could be useful to obtain, with as little work as possible, graphs of 1 or more functions with 0.014 inch accuracy. By varying ranges and scales very good approximations could be obtained for the determination of the roots of equations, etc.
CHAPTER 2

THEORY

This chapter elaborates the theoretical base, upon which the Plotter Package and the Graph Utility is based.

2.1. General Description of the Method Used

Basically the plotter instructions come as three arguments to the subroutine, "PLOT" (X, Y, PEN). Excluding the special frills that are usually present, the first two arguments specify the point to which the pen should be moved, with PEN indicating whether the pen should be up (not plotting), or down (plotting). By convention the X-direction is chosen to be down the page, with an almost unlimited range, or, at worst, a very large range relative to the Y-direction. The Y-direction is chosen to be across the page, with a limited range usually of the order of 8 inches, 11 inches, or 30 inches.

The Electrostatic Printer/Plotter (EPP) plots one row of line
dots across the page at a time. Thus, as the printer/plotter proceeds down the page (in the positive X-direction), the plots or tracings can be made. But --- the EPP can only move in the positive X-direction (down the page). As a result, only as long as the calls to "PLOT" come in ascending order of the X-coordinate, can the plotter calls be handled as they occur. This describes a special case; and in fact, is the case dealt with in the Graph Utility. The Graph Utility is designed to plot the function f(x), where f(x) is unique for all values of x. This restriction allows for a one pass system, in which
the function is plotted as soon as it is evaluated.

In a conventional plotting system the pen is allowed to move in any direction (right, left, up the page, down the page). Of importance to this project is that it can move both up the page and down the page, where an electrostatic printer/plotter cannot. Note that freedom of movement in the Y-direction presents no problem to an EPP. Therefore in order to do conventional plotting on an EPP, the problem of pen movements up the page (in the negative X-direction) must be solved.

There is no simple solution to this problem involving a one pass system, where plotting is done as the plotter instructions come in since the very last call might involve plotting a line at the very top of the plot (involving a small X-coordinate). Therefore a two pass system is needed where, in some way, the movements of the pen are recorded until the user indicates that he has finished plotting. Then a second pass does the plotting.

The most straightforward solution is that in which a core image, or, more realistically, a disk image of the final plot is constructed. This image initially would be empty (no points). Then as the plotter instructions come in, they are processed and plotted on to the image by inserting the necessary bits. Then when the user is finished the image is copied on to the printer/plotter. There are drawbacks to this solution, however. The storage required, whether it is core storage or disk storage, is considerable. Just considering the case at McMaster University, the printer/plotter has a resolution of 72.5 points per inch and the computer involved has 16-bit words. Now consider a frame 7.75" x 7.75" involving 560 points x 560 points. The storage required
for this frame would be 313,600 bits (39,200 bytes or 19,600 words). This would not fit into the core (12,000 word) and would have to go on to the disk, taking 5.5 tracks (or 2.75 percent of the 200 tracks available). The real catch here is that only a small plot is being considered. It is not unreasonable for a user to want a 50 or 100 inch plot (in the X-direction). On some machines this solution might be reasonable but on the HP2100A at McMaster, it is not a practical solution.

The solution chosen is to take the calls to "PLOT" as they come in and to reorganize these calls into acceptable sub-plots (that is plots ascending in the X-direction only). This is done during the first pass, and for all but extreme cases, the storage is less as the (X,Y) coordinates to be stored are quite compact. The storage requirements are completely independent of the coordinate values, depending only on the number of plotting instructions. The set of all the sub-plots created is the user's plot. In the second pass, these sub-plots are plotted in parallel. This means that for a particular X-value many sub-plots may be adding points to the plot. When a sub-plot is exhausted, it is deleted, and when all the sub-plots are done, the plot is complete. That in a nutshell is the method used in the plotting package. In the next section, definitions and terminology relating to this method are formalized. To help tie things together this is the correspondence:

The term sub-plot used above corresponds to the term string which is a collection of contiguous line segments.

The term plot used above corresponds to the term list which is a of all of
2.2 Definitions and Terminology

For the purposes of this project the concepts of a point, of a line, of a straight line, a straight line segment, of a string and of a list must be formalized.

These first few definitions are intentionally intuitive rather than being rigorous or complete. All definitions are in terms of the XXY coordinate plane.

![Diagram](image)

**Figure 2.1. A Point in XXY Plane**

A point $P$ is defined as an ordered pair $(x,y)$ where $(x,y)$ represents the position in the XXY plane, and where $x$ is the displacement from the Y-Axis, and $y$ is the displacement from the X-Axis.
figure 2.2  **Lines in XxY Plane**

- a **line** is a set of contiguous points.

A line can have finite length (for example L2, L3, L4) or can have infinite length (for example L1, L5). A line may have no end points (L1, L3), one end point (L5) or two end points (L2, L4).
A straight line is a line for which the ratio of the change in the $y$ displacement to the change in the $x$ displacement is constant.
i.e. in figure 2.3.

$$\frac{\Delta y_1}{\Delta x_1} = \frac{\Delta y_2}{\Delta x_2} = \frac{\Delta y_3}{\Delta x_3} = \frac{\Delta y_1}{\Delta x_1} = K$$ for all $i$

This constant $K$ is called the slope.

A straight line segment is the set of points lying between $P_1$ and $P_2$, and including $P_1$ and $P_2$.

Henceforth, the line segment of $P_1$ and $P_2$ will be used.
figure 2.5. Contiguous Line Segments
- Two line segments \((P_1, P_2)\) and \((P_3, P_4)\) are said to be contiguous if \(P_2 = P_3\).

figure 2.6. N-Tuple of Line Segments (String)
- For \(n > 0\) an \(n\)-tuple of line segments
  
  \[ (P_1, P_2), (P_3, P_4), \ldots, (P_{2n-1}, P_{2n}) \]
  
  is a string \(S\) if \((P_{2i-1}, P_{2i})\) and \((P_{2i+1}, P_{2i+2})\)
  
  are contiguous for all \(i\), where \(1 \leq i \leq n\).
  
  This is usually written as an \((n+1)\)-tuple of points
  
  \((P_1, P_2, P_3, \ldots, P_{n+1})\).
In this project only monotonic strings are used.

Either a string \((P_1, P_2, \ldots, P_{n+1})\) is an up string, in which case \(x_i < x_{i+1}, \ 1 \leq i \leq n\).

Or a string \((P_1, P_2, \ldots, P_{n+1})\) is a down string, in which case \(x_i < x_{i+1}, \ 1 \leq i \leq n\).

In figure 2.7, \(S'\) is an up string and \(S\) is a down string.
figure 2.8. List (of Strings)

- In figure 2.8, is a set of down strings \( \{S_1, S_2, \ldots, S_7\} \).

An ordering is defined on this set of strings descending on the first x coordinate in each string.

The set of down strings \( \{S_1, S_2, \ldots, S_n\} \) is said to be ordered if \( x_i < x_{i+1} \) for all \( i \), where these \( x_i \)'s are the first x value in the string \( S_i \).

A set of ordered down strings is called a list and is written \( (S_1, S_2, S_3, \ldots, S_7) \).

i.e. an ordering of \( \{S_1, S_2, \ldots, S_7\} \) in figure 2.8. would be \( (S_2, S_1, S_4, S_6, S_5, S_7, S_3) \).
2.3. **Detailed Description of Method Used in Plot Package**

As it is the convention in plotting that across the page is the Y-direction and down the page is the X-direction, subsequent diagrams and illustrations will be oriented in this way so as to better relate these illustrations to the actual plots.

The method used in the plotting process breaks conveniently into two parts. The first part incorporates the accumulation of the plotter instructions and the reorganization of these instructions into a set of strings (or sub-plots) which can be handled by an electrostatic printer/plotter. This first part acts as a buffering process between the user's plotter instructions and the printer/plotter. This process is embodied in the "PLOT" subroutine which does the accumulation and reorganization.

The second part performs the actual plotting of the set of strings on to the printer/plotter. In this second part the set of strings is ordered to create a list. Then an interpretive process plots the information in the list. This second part is incorporated by the subroutine "PRINT".

2.3.1. **Description of PASS I (PLOT) (SAVING PROCESS)**

Before describing the high level method used, the low level method will first be discussed. First, the construction of the strings will be considered (i.e. the pen movements to be noted). Plotting, or moving the pen, with the pen up can be practically ignored. Only the pen position at the end of this pen movement, and the fact that the previous string, if any, has ended, is of any importance.
Plotting with the pen down is a different story. This can take place in 4 different ways as illustrated in figure 2.9.

$\text{(a) } (x_1, y_1) = (x_2, y_2) \quad \text{(b) } x_1 = x_2 \quad \text{(c) } x_1 < x_2 \quad \text{(d) } x_1 > x_2$

**figure 2.9. Possible Line Segments**

There are 4 cases:

- **case (a)** - the two points $P_1$ and $P_2$ are one and the same point.
- **case (b)** - the two points $P_1$ and $P_2$ have the same $X$-coordinate.
- **case (c)** - the $X$-coordinate for $P_1$ is less than that of $P_2$.
- **case (d)** - the $X$-coordinate for $P_1$ is greater than that of $P_2$.

Notice that differences in the $Y$-coordinate have no bearing on classifying cases. In case (a) there is a zero length line segment which in plotting does have a physical meaning as that is how a single point is plotted. To be more precise case (a) is simply a special case of case (b) and they will be considered together since they are logically the same.

What is to be done in the various cases in figure 2.9, depends entirely on whether a string is currently under construction or not and, if a string is under construction, whether the string is an up string or a down string. Thus there are three possible situations to be considered.
for the occurrences of the different types of line segments. (See figure 2.10.)

(I) Pen (up) II Pen (down) III Pen (down)

Figure 2.10: 3 Pen Situations

There are 3 cases:

Case I - pen is up and no string in progress.
Case II - pen is down creating a down string
(+)X-direction)
Case III - pen is down creating an up string
(-)X-direction)

It is temporarily feasible to construct an up
string which is converted into a down string at
its completion.

In Case I, a line segment as described above, is encountered
with no string in progress. It is a simple case of starting a new string.
When a string is started it is declared to be going up (-)X-direction) or
going down (+)X-direction). In the line segment cases (a,b,c) where x₁
is less than or equal to x₂ the string is declared to be going down, and
when the next line segment is encountered the situation is as described
in Case II. The equals case (x₁ = x₂) could be left undeclared until a
declaration was necessary, but the saving in core would be small as the
occurrence of this, where it leads into an up string, is statistically small.
In line segment case (d) where \( x_1 \) is greater than \( x_2 \) the string is declared to be an up string, and on subsequent line segments a Case III situation exists. In both cases above, the string is started by setting the current string \( S = ( (x_1, y_1), (x_2, y_2) ) \).

In case II, where a down string was already in progress, in cases (a,b,c) the new line segment is simply added to the string. Therefore for the \( i \)th line segment \( S = ( (x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), (x_{i+1}, y_{i+1}) ) \). But in case (d) where the line segment is starting to go up, the current string is terminated and an up string is started.

In case III, where an up string is already in progress, the situation is opposite to case II. In cases (a,b,d), a string is continued and to add the \( i \)th line segment to the string \( S = ( (x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), (x_{i+1}, y_{i+1}) ) \). In case (c) the current up string must be terminated and a down string started. The final strings must all be down strings, and so the up string must be converted. To convert the up string \( S = ( (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \) it is inverted, becoming the corresponding down string \( S = ( (x_n, y_n), (x_{n-1}, y_{n-1}), \ldots, (x_1, y_1) ) \).

(Remark - this reversing process must also be done when in case III the pen is picked up and moved, or the terminal plot call made).

With these techniques and procedures it is possible to take any sequence of pen movements (or plotter instructions), and to organize these movements into strings (or sub-plots) and in fact into down strings. The plotting information at the end of this first pass becomes a set of strings (or sub-plots). Thus the basic philosophy fragments the unmanageable plot into sub-plots all of which are manageable (Divide and conquer).
2.3.2. **Description of PASS 2 (PRINT) (PLOTTING PROCESS)**

As in 2.3.1, the low level method is described first. First the method that the printer/plotter uses to plot must be considered. The printer/plotter plots by putting points onto the output line as required. Successive lines form a plot or picture, in the same manner as a television picture is formed. Consider the plotting of a line segment over 5 or 6 of these lines: as the slope of the line changes a problem becomes evident. (See figure 2.11.)

![Diagram of line segments](image)

**Figure 2.11. Line Segments Over 7 Plot-Lines**

Starting in case (i) and case (ii) where the slope is small there is no problem as a continuous line results. This holds true until the slope becomes greater than 1. Then in case (iv) where the slope is very large there are large gaps in the line: it is smooth but the lack of continuity is undesirable. Looking at case (ii) there is another slight problem but this problem is actually a solution to the first problem. In case (ii) the line is staggered with a few rough edges, but at least it is continuous. There is no solution to this problem and it is preferable to the disjointed line as in case (iv). Therefore changing case (iv) to be like case (ii) as indicated by the empty circles in figure 2.11. is the answer. A few rough (or staggered) lines will occur.
but continuity takes precedence. The solution is to interpolate backwards half way and forward half way. Then a range is plotted for each line depending on the slope, and, in fact, is equal in length to the value of the slope. (This is because lines are 1 unit apart (ΔX=1)). In this way each string creates a range on the line.

PASS 2 (or PRINT) is entered when the plot has been completed and is in the form of a set of down strings (or sub-plots). This pass faces the challenge of decoding (or interpreting) this set of strings and producing the desired plot.

It is quite possible to handle this set of strings as they are. However, this would be very inefficient since each string would be checked on each line although few strings extend over the whole range of x values. So in the interest of efficiency an ordering of the set of down strings is made to produce the corresponding list. (Note that until this ordering is done the set is not necessarily a list). At this point the starting x values for each string are in ascending order. This means that when a string Sj occurs where the starting x-value in Sj, xj is greater than the current x-value, x, we know that this string and all strings from Sj to Sn can be ignored for this x. When this Sj is reached or after handling Sn, the output line image is plotted. Excessive work is also avoided by removing the string when it is finally exhausted.

Both these techniques are illustrated in figure 2.12.

Need not search past Sj

figure 2.12. Logical organization of strings
In the diagram in figure 2.12, the strings are shown not as they physically occur in the plot, but where they logically occur in the list. These two techniques help to minimize the work to be done. This is a destructive process as strings are destroyed when they are used up. Also note that when the last string is exhausted the plot is finished and cut on the paper.

How does one handle the individual strings? Because of the careful construction in PASS I, it is known that these are all down strings. A down string is simply a contiguous n-tuple of monotonic non-decreasing (x value) line segments, and the only problem is how line segments are handled. The correspondence between the x value and the line segment involved must be considered. This can occur in five logically different ways as illustrated in figure 2.13.

![Diagram](image)

Case (a) Case (b) Case (c) Case (d) Case (e)

**figure 2.13. X Value and Line Segments**

There are 5 cases:

Case (a) the line segment \((P_1, P_2)\) lies below current x value, i.e. \(x_1 \ll x_2 < x\)

Case (b) the line segment \((P_3, P_4)\) starts at the current x value, i.e. \(x = x < x_4\)
Case (c) the line segment \((P_5, P_6)\) lies on the line \(y = x\)

i.e. \(x_5 = x = x_6\)

Case (d) \(x\) is inside the line segment \((P_7, P_8)\)

i.e. \(x_7 < x < x_8\)

Case (e) \(x\) is at the end of the line segment \((P_9, P_{10})\)

i.e. \(x_9 < x = x_{10}\).

Case (a): the line segment lies completely above the \(x\) value.

This line segment is skipped, and it is deleted from the string. The next line segment is checked, again as one of the above cases. If no line segments are left, the string is deleted from the list. Case (a) can occur only at the initial \(x\) value:

i.e. if \(S = (P_1, P_2, \ldots, P_7)\) then \(S\) becomes \((P_2, P_3, \ldots, P_7)\)

and the line segment \((P_2, P_3)\) is immediately considered;

- in the special case \(S = (P_1, P_2)\) \(S\) would no longer be a string so the entire string is deleted from List.

Case (b): the line segment starts at the \(x\) value. From figure 2.13, it is clear that \(y_3\) must be included in the plot, but also we must interpolate towards \(y_4\). As the \(x\)'s change by 1 unit for each line, it turns out that the next \(y\) value is \(y_3 + \text{Slope}\) and to interpolate halfway we go to \(y_3 + 1/2 \text{Slope}\). So, in order to get a continuous line for this line segment the points between and including \(y_3\) and \(y_3 + 1/2 \text{slope}\) are put into the line buffer.

Case (c): \((P_5, P_6)\) lies on the line \(y = x\). The range of \(y\) values from \(y_5\) to \(y_6\) are put into the line buffer.

Case (d) \(x\) lies inside the line segment and not at an end point.

This is the general case where \(y\) values in the line must be interpolated and
the range \((y - 1/2 \text{ slope}, y + 1/2 \text{ slope})\) put into the line buffer.

Case (e): \(x\) lies at the end of the line segment. This is similar to case (b) and is handled in a similar fashion, as the \(y\) values from \((v_{10} - 1/2 \text{ slope}, v_{10})\) are put into the line image.

In addition, in cases (c) and (e), an extra step is necessary. The line segment is skipped as in case (a), and the next line segment checked immediately, if one exists. If no line segments are left the string is deleted from List.

Note that both the old line segments and the old strings are destroyed as soon as they are finished. It is also noteworthy that this iterative process allows the processing of strings as well as line segments, since the end of one line segment triggers the consideration of the next line segment. When no more line segments exist in the string, the string is automatically destroyed.

So in brief, the set of down strings is converted into the ordered set of strings, LIST. Then by processing only the relevant strings for each \(x\), and by interpolating as necessary for each line segment, the line buffer is formed and plotted. This continues until all the strings have been processed.

2.4. Description of Theory Used in Graph Utility

As stated earlier the Graph utility is that special case of a plot in which only increasing \(x\) values occur. This particular utility is designed to plot functions and, to be more precise, to plot functions with unique values for all \(x\) values. Although not theoretically important the package can handle several functions at once.
The theory upon which the graph utility is based is similar to that theory in the plot package. In order to plot continuous lines interpolation is necessary. In this case instead of using line segments and slope, the actual x-values are used. Y-values are calculated for each value of x, and in order to interpolate we must look back to the last y-value and ahead to the next y-value. Thus three y-values are used to determine the range, the last y, the present y, and the next y.

The interpolation simply involves halving the two intervals. Then these two interpolated values, together with the present y-value, are used to determine the range of y-values to be put into the line image. As seen in figure 2.14 below, the present y must be included to guard against the case on the right. In this case the present y does not lie in between the interpolated values but must be in range to be put into the line image.

![Diagram of graphing using 3 consecutive y-values](image)

**FIGURE 2.14. Graphing Using 3 Consecutive Y-Values**

The X-Axis and the Y-Axis are included automatically in the graph. In addition there are borders on each side and most important the limits of the graph and the scales used are reported. The Axes give the user reference points and the reporting of the limits and scales guards against keypunching and programming mistakes which could cause a communications breakdown (i.e. the user thinks he asked for one thing, but the computer is actually told something else).
CHAPTER 3

IMPLEMENTATION DESCRIPTION

This chapter explains how the theory developed in Chapter 2 is converted into a plotting system.

3.1. General Description and Breakdown of the Plot Package

As explained in Chapter 1, plotting is accomplished by describing sequences of pen movements. The pen is instructed to move from its current location to a new location, either with the pen up (and not plotting) or with the pen down (and plotting). As few plots are made up of one continuous line, the ability to move to a location without plotting is necessary. The plotter instruction is issued as a call to the subroutine PLOT. The user may call PLOT directly or may call special purpose plot utilities which in turn call PLOT. However, it is necessary that PLOT be called in order to perform the actual plotting. It is for this reason this Plotting Package begins with routine PLOT.

Figure 3.1. is a pictorial representation of the PLOT Package. In brief, the user's program either directly or indirectly calls PLOT with the plotter instructions. PLOT takes these calls, and reorganizes them into strings. The strings are stored away and upon a completion call, PLOT calls PRINT which does the actual plotting. PLOT has a utility routine called INVRT which converts an up string into a down string. PLOT and INVRT make up the first pass of the plotting package.

PRINT takes the strings created in PLOT, uses the utility ORDER
to create the ordered set of strings (or list), and then by using two
more utilities INSRT (to put points into the line buffer) and EFOUT (to
plot the contents of the line buffer), plots the line segments contained in
the list. Then control is passed back to PLOT and thence back to the
user. PRINT, ORDER, INSRT, and EFOUT make up the second pass of the
plotting package.
figure 3.1.

Pictorial View of Plot Package
3.2. "PLOT" Routine

This routine is the basic routine in the plotter package, and was written to be compatible with the Benson-Lehner Plotting System. Both the user and the high level plotter routines (e.g. SYMBOL) use PLOT to do their plotting. The calls to PLOT contain three arguments (X, Y, and PEN). X and Y are reals which usually contain the target location to which the pen will be moved, and PEN is encoded with either the pen position (up, off the paper, or down, on the paper) or special plot calls. These special calls allow for the setting or resetting of the origin, for the establishment of translation, and for returning the current pen position in the XY-plane. On the very first call to PLOT the scale factor can be adjusted to expect calls in centimetres rather than in inches (which is the default scale factor). Figure 3.2. is a table of plot calls with the corresponding action to be taken.

On the first call the needed initialisation is done, including resetting the scale for centimetres if required. Then from this point, on the first call, and on all subsequent calls, we proceed to convert X, and Y to integer plotter units, and to use current origin and translations to obtain an (X,Y) position in absolute plotter units (1 plotter unit = 1 step or 0.0138 inches).

Now we concern ourselves with PEN, according to the actions indicated in the table in figure 3.2. We shall ignore the special calls to PLOT as these are mainly bookkeeping problems. The main concern is when the pen is down (PEN=2) and plotting. Both up strings and down strings are constructed and stored sequentially in array LIST. There is a tag array, TAGS, which is set to the index of the first member of the string, when
<table>
<thead>
<tr>
<th>PEN value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>only on 1st call, set scale for centimetres</td>
</tr>
<tr>
<td>-3, -2, -1</td>
<td>set origin to arguments (X,Y)</td>
</tr>
<tr>
<td></td>
<td>change sign of pen and decode pen again.</td>
</tr>
<tr>
<td>Q</td>
<td>return current pen position to user</td>
</tr>
<tr>
<td>1</td>
<td>plot with pen unchanged</td>
</tr>
<tr>
<td>2</td>
<td>put pen down and plot to (X,Y)</td>
</tr>
<tr>
<td>3</td>
<td>pick pen up and move to (X,Y)</td>
</tr>
<tr>
<td>40</td>
<td>reset translation</td>
</tr>
<tr>
<td>999</td>
<td>end of plot, put plot out.</td>
</tr>
</tbody>
</table>

**figure 3.2. TABLE OF PEN CALLS**
that member is put into LIST. Two words are left in front of each string, the first is set to zero and used later by PRINT, and the second is a pointer to the start of the next sequential string, in array LIST. To better see this structure, see figure 3.3., where the organization is illustrated.

**figure 3.3.** TAGS () and LIST () STRUCTURE
An up string is stored with coordinate pairs backwards in LIST. Therefore instead of the sequence \( x_1, y_1, x_2, y_2, \ldots \) as in a down string, the coordinates are stored \( y_1, x_1, y_2, x_2, \ldots \) instead. Then when an up string has been completed the INVRT utility is called to reverse the up string to form the corresponding down string. The coordinates are stored in this strange manner because it simplifies the INVRT routine.

When a terminal call to plot is made (PEN=999), plot calls PRINT to put the plot onto paper.

3.3. "INVRT" Routine

This is a service routine for subroutine PLOT. It takes an up string and converts it into a down string. This conversion has been facilitated by the storing of the up strings with the coordinates backwards \( (y, x) \) instead of \( (x, y) \). To convert the string \((y_1, x_1, y_2, x_2, \ldots, y_n, x_n)\) into a down string simply interchange the \( n \) pairs \((y_1, x_n), (y_2, x_{n-1}), \ldots\) to get \((x_n, y_n, x_{n-1}, y_{n-1}, \ldots, x_1, y_1)\).

3.4. "PRINT" Routine

PRINT is called from PLOT upon a terminal plot call (PEN = 999).

PRINT is called with three arguments:
- \( \text{LIST} \) - which contains the set of down strings,
- \( \text{TAGS} \) - which contains the pointers to the start of the strings in LIST
- \( \text{ITAG} \) - which is the number of tags in TAGS (and also the number of strings in LIST). Note that at this point all strings are down strings.
PRINT takes this information and creates a plot in the following manner:

First ORDER is called to perform a tag sort on the strings in LIST, by reorganizing the tags in TAGS so that the initial x values of the strings indicated in consecutive TAGS are in ascending order. This process converts the set of strings in LIST into a list of strings as defined in Chapter 2.

Then we start at \( x = 0 \) and for increasing \( x \) values the following procedure is followed. Starting at the first string, the range of \( y \) values is determined for each string for this \( x \) value, and with necessary interpolations performed. These values are inserted into the line buffer using utility INSRT. We continue until we find either the last string or a string where the initial \( x \) value is greater than the current \( x \) value. Then the line buffer is plotted onto paper by routine BFCUT. Each string is deleted when it is completely plotted, and this whole process continues until no more strings exist. At this point we have finished and PRINT returns control to PLOT.

3.5. "ORDER" Routine

This routine is called from PRINT in order to convert the set of strings into a list. As Fortran does not allow string or pointer types, strings are stored sequentially in an array, LIST, with a corresponding tag array, TAGS. In this way the string data structure is achieved. In order to sort the strings, ORDER performs a tag sort on the tag array TAGS, and the array LIST.

The logic in this routine was obtained from a previous fourth year project at McMaster on sorting, and this was a normal sort routine. It
was considered in their report to be in the top 2 or 3 of the routines that they considered but, more important to this project, only 3 or 4 lines had to be changed in order to make it into a tag sort. The sort logic seems to be very efficient and, both small (in sense taken) and simple.

As this was written by someone else no attempt will be made to describe the method. The source is well annotated to indicate which changes were made.

3.6. "INSRT" Routine

PRINT calls INSRT to put a point into the line buffer. The formal parameter POINT is an integer number from 1 to the number of points in the line buffer, indicating the bit to be placed in the line buffer (LINE). As this machine is not bit addressable, the word address is determined, and a bit inserted into this word at the proper position. This routine could have been written in Fortran, and in fact a Fortran version was written and tested. The Assembler version is about one half as large (34 words versus 67 words) and much more efficient (i.e. in number of instructions to be executed). Since this routine is used so often, the Assembler version is used in the implementation, and the Fortran version inserted only into the documentation.

3.7. "BFOUT" Routine

PRINT calls BFOUT to plot the line buffer, LINE, on the printer/plotter. The plot is accomplished by an executive call. This call does a binary write — without — wait to the printer/plotter. This means that the output operation is initiated and allowed to go on automatically while program
control is returned to PRINT. This allows PRINT to be concurrently processing the strings for the next value of x.

Through careful testing it was found that three speeds could be obtained on the plotter through different methods of plotting, 10000 lines were plotted and the plotting time determined by subtracting the looping time from the elapsed time. The cases and times are as follows:

1) writing-with-wait (normal) took 108.1 seconds.

2) writing-without-wait and checking for status before starting successive write took 117.9 seconds.

and 3) writing-without-wait with no status checking took 89.7 seconds.

In each of the above cases, including case (3), there was no problem with overwriting of the buffer. It was also found that plot speed could be increased by writing large buffers (3500 words instead of 35 words).

It was only here that overwriting of the buffer could happen, if no status check were made. This is true on our machine but to help to protect the users of other machines buffer copying is done (though it is not necessary). The line buffer LINE is copied into BUFF and simultaneously LINE set to zero for the next x value. Then BUFF is written instead of LINE.

3.8. Reason for Version II

The plot package as described so far is quite satisfactory. In fact, this is probably the best version from a theoretical point of view. But unfortunately on the HP2100A there is a severe restriction. Only about one thousand calls to plot can be made, and although in some cases this is sufficient, this restriction could seriously hamper some plot users. The reason that only 1000 calls to plot are allowed is that the calls are stored in an (x,y) coordinate pair in the array LIST. Since the HP2100A only has
12K decimal words of core, and only 8K is available to the user, after his program is added only 2000 words are available for LIST without overflowing the available space. The word length is 16 bits and so packing is not feasible (as it would be on the CDC6400 with 60 bit words). Therefore there is no easy solution that will allow larger plots.

Obviously any expansion of the plotting ability is going to have to involve the disk. So what we have to do is, to use the disk and to take advantage of the fact that we basically have 2 passes, PASS 1 (PLOT, INVRT) which accumulates the plotter information and PASS 2 (PRINT, ORDER, INSRT, and BFOUT) which plots the information on the paper.

The solution arrived at incorporates both these features.

1) only one string at a time is kept in core. As soon as a string is finished it is put into a sector buffer and written to disk (Routine CLEAR). Instead of having an array LIST in core we gradually construct an image of LIST on disk. The assignment (LIST (I) = ...) and the accesses ( . = LIST (I) ) of array LIST are simulated with a subroutine PUT and a function LIST respectively. To tidy up the last sector buffer a call to CLSEP writes the last sector to disk. Also the calls to INVRT are handled through routine IVRTA. All these routines share common storage and are written in Assembler. This set of routines is called DSKIO. In the diagram in figure 3.4. we are at step (1).

2) Next we start to swap out core with a call to SWAPI (Step (2) in figure 3.4.). SWAPI saves core from 24000B to 30000B onto disk and replaces it from a binary data file called "SWAP2"
(Step 3 in figure 3.4.). Then control is passed to "SWAP2". Upon return the core swapped out in Step 2 is swapped back in (Step 9).

3) SWAP2 swaps out the core from 10000B to 24000B (which will be called LO-CORE in this section). This is Step 4 in figure 3.4. Next, the tag array TAGS is moved down to the start of user core (10000B) (Step 5 in figure 3.4.). Then the LIST array is read in from disk and placed right after the TAGS array. (Step 6 in figure 3.4.) Next the normal call to PRINT is made (Step 7) and then LO-CORE is reset (Step 8). Finally we return to SWAP1.

The above is the basic process needed to swap core. Unfortunately all of this is complicated because the HP2100A is a page-addressable machine and about 100B words of base page links must also be carefully swapped when entering and leaving SWAP2. The binary date file "SWAP2", which contains the object code for SWAP2 (in a loaded form), is configured by the program SETHI which only has to be run once.

All this work gains us the core from 10000B to 24000B for use by TAGS and LIST. This represents about 6000 decimal words which leaves 5500 words for LIST, representing an additional 3500 words for LIST over Version I (or about 1750 additional plot calls). Note that this is the true measure of the expansion of Version II over Version I as more core will have no effect on the difference between the two versions (it will still be 1750 additional plot calls). In fact, at about 20K, Version I would frequently be the better version for simple plots as there would be little if any restriction left.
- During PASS 1

- During SWAP1

![Diagram showing memory and disk storage during PASS 1 and SWAP1.]

Figure 3.4 (page 1 of 2)
1. put LIST out to work area
2. copy HI-CORE PASS 1 to work area
3. copy "SWAP2" into HI-CORE, call SWAP2
4. copy LO-CORE PASS 1 to work area
5. move TAGS to 10000B
6. read LIST into core
7. call PRINT & plot
8. reset LO-CORE PASS 1 into LO-CORE, return to SWAP 1
9. reset HI-CORE PASS 1 into HI-CORE, return to PLOT

figure 3.4. Version II of PLOT Package
3.9. "PLOT" Routine (Version II)

There are no logic changes whatsoever between Version I PLOT and Version II PLOT. The only changes necessary are in accesses of the array, LIST. An assignment to the array is changed into a CALL PUT (value, index, ...). Unfortunately several flags from PLOT have to be passed in case we overflow the string buffer in the Disk I/O package. A reference to the array LIST doesn't change in the code, but uses a function with the same name rather than an array access. Because the string record is in the Disk I/O package, the call to INVRT becomes a call to IVRTA which calls INVRT. Finally there are two subroutines CLEAR, called to put the string out when it is terminated, and CLSEF, which puts the last sector onto the desk if necessary.

Then instead of calling PRINT we call SWAPI which begins the core swapping process that ultimately calls PRINT.

3.10. "DSKIO" Package

This is the assembler package that includes:

1) PUT - to simulate an assignment (LIST (I) = )
2) LIST - to simulate a reference ( = LIST (I) )
3) CLEAR - take strings when they are done and route them to disk.
4) CLSEF - if last sector not written to the disk write it and
5) SWAPI - to start core swapping process that ends with PASS 2 in core with LIST array (from disk).

This package was written in Assembler because this allows for common buffers, pointers and flags. In addition, it was much easier to write in Assembler than in Fortran. Since this is a very machine dependent feature writing it in HP Assembler is quite reasonable.
3.10.1. "PUT" Routine

This routine simulates an assignment statement involving the array LIST. A string record is local to this routine, and after checking that the index is either in the string or after it, the value is put into the string record (i.e. no assignments are allowed before the starting index of the string, and in fact it should be impossible for PL/I to make such a call but the error checking is left in anyway). The only limit on these insertions is the size of the string record, 128 words. If this limit is reached, overflow occurs. The string is ended and a call to INVRT made if necessary. Then a new string is started (i.e. we break a large string into parts).

3.10.2. "LIST" Function

This routine simulates a reference to the array LIST. Again, as in PUT, the index is checked to see that it is referring to an element in the current string. Then the value is returned. (i.e. \( \ldots = \) LIST \( \ldots \) returns the Ith element of LIST).

3.10.3. "CLEAR" Routine

This routine takes the current string and starts putting it into the current sector buffer. When the buffer becomes full, it is written onto the disk in the work area. We continue until we have all of the string. This is essentially "flushing" the string record.

3.10.4. "CLSEP" Routine

This routine takes care of the last sector buffer mentioned in 3.10.3. If the buffer is empty we don't do anything, otherwise we write
it out. This is essentially closing the file.

3.10.5. "SWAP1" Routine

This is called from the PLOT routine and initiates the swapping sequence. First HI-CORE is written to the work area on disk. Then HI-CORE is loaded from the binary data file names "SWAP2". Finally, we transfer control to SWAP2, which continues the swapping sequence. Upon return SWAP1 resets HI-CORE as it was initially.

3.11. "SETHI" Package

This package includes the "SETHI" program and the "SWAP2" routine. SETHI is a routine which configures SWAP2 and writes the binary version onto data file called SWAP2. SWAP2 is the second part of the swapping sequence which saves LO-CORE, moves TAGS down to the first word available in user memory, reads in the LIST array from disk and then calls PRINT to plot the information. When PRINT is done SWAP2 resets LO-CORE and returns to SWAP1.

3.11.1. "SETHI" Program

This is a main program which configures the SWAP2 binary code in HI-CORE and then writes out the code to the binary data file called SWAP2. SWAP2 is configured by forcing it to load at address 24000B and copying the link area near the end of core, so that it can be restored when SWAP2 is executed. Then we have enough information that it can be saved on "SWAP2".
3.11.2 "SWAP2" Routine

This routine completes the swapping procedure. First the link area, which had been saved near the end of core during the execution of SETHI, is swapped with the link area (from the user's program). Then the user core below the partition point (10000B to 24000B) is written to the work area on disk. Next the tag array TAGS is moved down to the start of the user area of core (10000B). Then immediately after the last tag, we read in the LIST array from disk (which was saved in PASS 1).

Now we call PRINT to do the plotting, and when the plotting is completed, all these steps must be undone so that we can return to the user normally. So the core where TAGS and LIST is now (10000B to 24000B) is reset. Then the link area is reset to that of the user. Now we return.

3.12. "GRAPH" Utility

This utility is provided to allow the plotting of functions in a simple way. It is designed to handle functions which have a unique value for each x. Instead of a two pass system, GRAPH is a one pass system where the functions are evaluated, and the function values plotted as soon as they are available. GRAPH also allows the user to plot several functions simultaneously on the same graph. There exists no real limit on this number, except an artificial one of 10 which is imposed because three arrays had to be declared with dimensions equal to the number of functions allowed.

The GRAPH utility was designed to be as easy to use as possible, while still allowing enough features to make it useful. The author hopes that a happy medium was achieved.

The output is bordered on all sides, and should the X or Y axis occur in the frame requested, it is automatically plotted. Then complete
reporting is done for the minima, the maxima, and the scales used for both X and Y.

3.12.1. "GRAPH" Routine

This routine is called directly from the users program. The user specifies a frame by:

1) X minimum and X maximum
2) Y minimum
3) X scale and Y scale

Then the user indicates how many functions are to be plotted on the graph, and the name of the function. The user supplies these functions by supplying a function of the form F(I,X) where X is the x value to be used in the evaluation and I chooses which function to be used. Note I could also limit the number of terms in a series to be used in evaluation. See figure 3.5.

FUNCTION F (I,X)

F = 0.493
IF (I.EQ.1) RETURN
F = F + 0.32 * X
IF (I.EQ.2) RETURN
F = F + 1.23 * X * X
IF (I.EQ.3) RETURN
END

Figure 3.5. Example of Possible Function Routine

Graph works fundamentally in the same way as plot except that the y
values are evaluated each time instead of using line segments and their slopes. In fact the utilities INSRT and BFOUT from the PRINT program are also used in the GRAPH routine. Just as PRINT interpolates values, GRAPH interpolates, only using actual values in each case. The plots produced from GRAPH are a little more continuous, as no round-off error can occur (since the function value is calculated for each plot line). Examples of plots from GRAPH are included in Chapter 4.
CHAPTER 4
RESULTS

This chapter illustrates and comments on the output from the plot package, and from the graph utility. Plots were made with both the Benson-Lehner on the CDC6400 and the Versatec on the HP2100A. These plots are compared, discussed, and criticized. The case study approach is used, and there are basically two cases undertaken for the PLOT Package and two cases undertaken for the Graph utility. No attempt was made to try any special purpose plotter routines (e.g. LETTER) as these are not yet available on the HP2100A. Admittedly, routines such as LETTER, generate numerous calls to PLOT, and these might tend to push the PLOT Package to the limits of its capacity. (It is easy to overload the PLOT Package in any event.)

4.1. Results of the Plot Package

Two case studies were chosen to provide some non-trivial examples of plotting, and also to show the strengths and weaknesses of the Electrostatic printer/plotter. The Benson-Lehner on the CDC6400 (at McMaster) was used to plot the counterparts of the EPP plots, so that comparison could be made. When comparing these plots, keep in mind that the Benson-Lehner has a stepsize or precision of 0.005 inches compared to the Versatec at 0.0138 inches. It is not possible for this printer/plotter to compete evenly with the precision of the Benson-Lehner.
4.1.1. Plotter: Case Study I (SNOOPY)

This plot program draws a picture (of Snoopy) by reading in plotter instructions from a card deck. Each card contains one set of plot parameters. The program for this is very simple (see figure 4.1.) for very good reasons. This same program with a different data deck could plot a different picture (Charlie Brown). Therefore the plot from this program is dependent solely on the data supplied and not on the program.

The plotter output from the Versatec is shown in figure 4.2.1. and the plot from the Benson-Lehner is figure 4.2.2.

Comments on Versatec"SNOOPY" and Benson-Lehner "SNOOPY".  
1. The two plots are virtually identical in size, in shape, and in general appearance.
2. The most striking difference is the darkness of the Benson-Lehner "SNOOPY". This is due to two differences in the machines. The Benson-Lehner uses an ink that is much darker than the Versatec plotting, and the Benson-Lehner pen draws a line that is thicker than the dot size on the Versatec. (See under "SNOOPY'S" mouth).
3. Next the Versatec plot looks rougher and disjointed in spots, where the Benson-Lehner is smooth and continuous. There are several reasons for these differences. The precision difference accounts for some of the roughness: also, the thicker pen will tend to cover up some small bumps in the output (they will not be quite as noticeable). Because a pen is physically being used on Benson-Lehner, continuity is no problem compared to the Versatec where with this precision even contiguous points look slightly disjointed. Finally, the word size on the HP2100A (16 bits) results in some round off error, which shows up on the plot as a line that doesn't
TEST PROGRAM #.1 [ SNOOPY ]

PROGRAM SNUPY
C
C
CALL PLOT( 2.0, 25.0, -3 )
50 READ(5,1) X, Y, N
   IF(X GE 99.0) GO TO 100
   CALL PLOT( X, Y, N )
   GO TO 50
100 CONTINUE
   CALL PLOT( 0.0, 0.0, -3 )
   CALL PLOT( 0.0, 0.0, 99.9 )
1 FORMAT(2F9.4, 1X)
STOP
END
END!

figure 4.1. Test Program "SNUPY"
figure 4.2.1. "SNOOPY" on EPP.
figure 4.2.2. "SNOOPY" on Benson-Lehner
quite meet a second line. (See "SNOOPY'S" Left heel).

In summary, the Versatec "SNOOPY" had a few rough edges, mainly due
to the precision, but is quite a reasonable facsimile of the same plot on
the Benson-Lehner. This is especially true when you consider that "SNOOPY"
only takes about 15 seconds to be plotted.

4.1.2. Plotter: Case Study II (HARMONOGRAPHS)

This example is the plotting of the path of a double conical
pendulum, and was taken from an article in Software-Practice and Experience
(Volume 2, Pages 293-301 (1972)). In figures 4.3. is the program listing.
Twelve parameters, A1, ..., F1, and A2, ..., F2, control the resulting plot.
Then for each value of T from 0.0 to 200.0 on X and Y are determined and
plotted. In the article a stepsize of 0.05 was suggested. But, this
creates too many calls to plot, for even the disk version to handle. This
version could handle a stepsize of 0.10, so in all these examples this
stepsize is used. In some of the plots the plotter output became a little
ragged as a result of this, but these plots proved to be a useful test for
the plot package.

Three of these plots are chosen for test cases here. They are chosen
to illustrate different problems and also to demonstrate different abilities
of the printer/plotter. In figures 4.4.1. through 4.4.6. are the three
test examples plotted on both the Versatec and the Benson-Lehner.

- In figures 4.4.1. and 4.4.2.
  A1, ..., F1 = 1.00, 2.00, 2.00, -2.00, -2.00, 1.00
  and A2, ..., F2 = 0.30, 0.65, 0.65, 0.65, 0.65, 3.01

- In figures 4.4.3. and 4.4.4.
  A1, ..., F1 = 1.00, 3.00, 0.00, 0.00, 3.00, 0.20
  and A2, ..., F2 = 1.00, 0.00, 0.30, -3.00, 0.00, 5.00

- In figures 4.4.5. and 4.4.6.
  A1, ..., F1 = 0.25, 4.00, 0.00, 0.00, 0.00, 1.00
  and A2, ..., F2 = 2.00, 0.00, 0.00, 4.00, 0.00, 2.00
PROGRAM HRMNC
REAL A1, B1, C1, D1, E1
REAL A2, B2, C2, D2, E2
REAL DELTA, T, Tmax
INTEGER PEN

DATA TMAX / 200 0 /

READ STEP FACTOR

READ(5,1) DELTA
1 FORMAT(F10.0)

READ IN 12 PARAMETERS

READ(5,2) A1, B1, C1, D1, E1, F1
READ(5,2) A2, B2, C2, D2, E2, F2
2 FORMAT(6F10.0)

REPORT STEP SIZE
AND PARAMETERS

WRITE(6,3) DELTA, A1, B1, C1, D1, E1, F1,
1 A2, B2, C2, D2, E2, F2

3 FORMAT(///, 1X, 19HPLTONG HARMONICS ,
1 / / , 1X, 16HWITH STEPSIZE = , F9.3,
2 / / , 1X, 15HAND PARAMETERS ,
3 / / , 1X, 20HA1 B1 C1 D1 E1 F1 = ,
4 6F9.3,
5 / / , 1X, 20HA2 B2 C2 D2 E2 F2 = ,
6 6F9.3,///)

MOVE PEN TO (4, 3)
CALL PLOT (4, 0, 3.0, 3)

Figure 4.3: (Page 1 of 2)
TEST PROGRAM #2 [HARMONOGRAPHS]  PAGE #2

C INITIALIZE T, PEN
C
T = 0.0
PEN = 3
C
C BACK UP ONE STEP
C
T = T - DELTA
C
20 T = T + DELTA
C
IF( T.GT Tmax ) GO TO 40
C
C CALCULATE ALL OUR FACTORS
C
R1 = 100.0 * EXP( -A1*0.01*T )
R2 = 100.0 * EXP( -A2*0.01*T )
S1 = R1 * SIN( F1*T )
S2 = R2 * SIN( F2*T )
T1 = R1 * COS( F1*T )
T2 = R2 * COS( F2*T )
C
X = 511.0 + B1*S1 + B2*S2 + C1*T1 + C2*T2
Y = 511.0 + D1*S1 + D2*S2 + E1*T1 + E2*T2
C
X = X / 140.0
Y = Y / 140.0
CALL PLOT( X, Y, PEN )
C
pen = 2
GO TO 20
C
SEND IN TERMINATOR PLOT CALL
C
40 CALL PLOT( 0.0, 0.0, 0.0, 999 )
STOP
END
END$
figure 4.4.2. HARMONOGRAPHE (II) ON Benson-Lehner
figure 4.4.3. HARMONOGRAPH (2) on EPP.
figure 4.4.4. HARMONOGRAPHE (2) on Benson-Löhmer
figure 4.4.6. HARMONOGRAM (3) on Benson-Lehner
Comments on the three pairs of plots.

1) All three of these plots are art forms more than anything else, especially in the first case (4.4.1. and 4.4.2.) where a three-dimensional effect is obtained.

2) As in the "SNOOPY" case, the precision and continuity of the Benson-Lehner plots create a more pleasing result.

3) Of the three plots the second, with the shrinking spiral (4.4.3. and 4.4.4.), is the Versatec plot that compares best with its Benson-Lehner counterpart.

The first plot (4.4.1. and 4.4.2.) was chosen to illustrate the problem of over-plotting. On the Benson-Lehner, when the pen crosses another line or falls on top of a previous line, the resultant plot is darker at this point. This can be easily seen in the centre of the Benson-Lehner plot (4.4.2.). Contrast this with the Versatec plot (4.4.1.) where no overplotting is possible. Since a point can only be plotted once, it can't get any darker. The result is an almost grey plot in the same region. This is primarily because overwriting cannot take place and also, the allowable density (72.5 points per inch) is not high enough to create a black area. (i.e. if the entire page were plotted it would look grey. If a more precise EPP were used the result would be black.)

The second plot was chosen (4.4.3. and 4.4.4.) to consider small movements of the pen. As the pen swings inward the movements become smaller and smaller creating smaller and smaller loops. The Versatec plot is a little rough in spots (again because of precision), but on the whole quite reasonable. The Benson-Lehner does very well. This type of plotter would run into a problem near the centre if a ball-point were used instead of
The third plot was chosen because, in addition to showing some
overwriting problems as in the first case, the round off error problem
is easily seen as lines of breaks appear to radiate out at several places
in the loops. This round off error is due to the word length problem
on the HP2100A (only 16 bits).

4.2. Results of the Graph Utility

Two cases were chosen to illustrate what the graph utility can do.
- In the first case a family of ten sine functions is plotted, illustrating
  how the index can choose one member of a family. (See figure 4.5. and
  figure 4.6.) This can be done in several ways, depending on the
  ingenuity of the user. This case also tested the speed of this package,
as 10 sine calculations per dot line amount to a considerable amount of
work. The speed was greatly affected (taking about 20 seconds for the
page instead of about 10 seconds in a single case). These plots are
continuous, with no breaks, as no round off error can accumulate, since
real arithmetic is used. The lines are still a little rough owing to
the precision.

- In the second case, two straight lines, and a circle are graphed. Note
  that a circle cannot normally be plotted but by treating it as two
  functions, one positive and one negative function, these problems can
  sometimes be overcome. (See figure 4.7. and figure 4.8.)

- The outputs in figures 4.6. and 4.8. are reduced from the normal size
  for purposes of this project.
GRAPH TEST PROGRAM # 1 [ CBFN2 ] PAGE # 1

FTN4, L

PROGRAM CBFN2
EXTERNAL F
CALL GRAPH - 2 0.2 75.1 0. - 2 0.1 14. F;
STOP
END

FUNCTION F : I : :
F = 3.0 + SIN : : + 3.1415 ± 1.0I + 1 : :
RETURN
END
END

Figure 4.5. GRAPH Test Program "CBFN2"
figure 4.6. GRAPH Output from "CBFN2"
PROGRAM FNSP
  EXTERNAL FSP
  CALL GRPH,-3,3,5,1,0,-3,0,1,0,4,FSP,
  STOP
  END
  FUNCTION FSP (X)
  FSP = 0
  GO TO (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1)
  CONTINUE
  Y = 4 * X
  IF(Y GT 0) FSP = SQRT(Y)
  RETURN
  CONTINUE
  Y = 4 * X
  IF(Y GT 0) FSP = -SQRT(Y)
  RETURN
  CONTINUE
  FSP = X
  RETURN
  CONTINUE
  FSP = 2.0 - X
  RETURN
  CONTINUE
  RETURN
  CONTINUE
  RETURN
  CONTINUE
  RETURN
  CONTINUE
  RETURN
  CONTINUE
  RETURN
  END
  END

Figure 4.7. GRAPH Test Program “FNSP”
YMIN = -3.00  XMAX = 3.50  YMAX = .274

SCALE USED IN X-DIRECTION (DOWN) 1 UNIT = 1.00 INCHES.
IN Y-DIRECTION (ACROSS) 1 UNIT = 1.00 INCHES

figure 4.8. GRAPH Output from "FNSP"
CHAPTER 5

CONCLUSIONS

5.1. Evaluation of the Plot Package

In Chapter I the following objectives were stated:

to develop a plot package, starting at routine PLOT,
which is capable of plotting on an electrostatic
printer/plotter, this PLOT routine to be compatible
with the Benson-Lehner plotting system and capable
of supporting the parts of the Benson-Lehner plotting
system which call PLOT.

I believe that I have achieved those objectives. First of all,
in Version I of the plotting system, we have a working, and tested
plotting system. This is shown by the examples in Chapter 4 and in
Appendix E. These plots were all produced by Version II, but the two
versions are logically identical as far as the plotter logic is concerned.
(To ensure this is so, when the final version of PLOT (in Version II) was
available, it was converted back to Version I by substituting for the
disk storage references, by core storage references. So, as far as
humanly possible these routines are logically the same). The basic
system involved in the plotter logic, (storing strings of line segments
in a first pass and plotting these strings in a second pass), makes
effective use of the available core, and attempts to maximize plotting
speed in the second pass. Two versions came about as a result of the
severe restriction that core size placed on the number of calls to plot
(about 1000). Thus the disk storage version was created, adding about
1700 potential calls to plot (i.e. Version II can accommodate 2750 calls
to plot). In speed, Version I with core storage is superior, as no disk
accesses are needed during Pass 1. Pass 2 will be handled at the same
speed in both versions. With only 12K available on the HP2100A, Version I
is not too useful whereas Version II, though it is still restricted, is
a useful plotting aid. With more core available the restriction on
Version I would become tolerable and at about 20K (providing about 5000
calls to PLOT) Version I would become superior because of its faster
processing speed. At present, a few of the newer features in the Benson-
Lehner PLOT routine are not available, but as the plot-library from the
Benson-Lehner, is converted to the HP2100A the necessary additions will
be made.

In summary, I have implemented and tested a plotting system which
can plot on an electrostatic printer/plotter. It is almost completely
compatible with the Benson-Lehner plotting system. Also, additional tests
on the CDC6400, producing plots on the printer, have shown that routines,
such as LETTER are compatible with my plotting system. I believe that
the basic design of the package is sound, and well structured, so that
improvements and modification should involve no major rewriting.

5.2. Evaluation of the Graph Utility

The objectives of this utility were as follows:

- develop a graph utility capable of taking several
  functions and creating a graph or plot, as simply
as possible, over a desired range, and with a desired scale.

As seen in the examples in Chapter 4, this has been done very successfully. I believe that the most appealing feature is its ease of use. The programs can be very simple, and quickly written. (i.e. within a half hour, and without writing complicated Fortran programs, a user could get a desired function plotted, and then re-plotted to graph the function using the most useful scales and limits)

5.3. Future Improvements and Additions

1. The plot package will first need a few simple modifications in order to make it as compatible as possible with the current Benson-Lehner plotting system on the CDC6400.

2. The Benson-Lehner plotter library, will have to be converted to the HP2100A. A few HP library routines are available, and may be feasible alternatives to some of the Benson-Lehner routines.

3. Documentation on the use of the plot package should be prepared and published (locally) for users of the package. Only a page or two will be needed, highlighting the minor differences between plotting on the CDC6400, and plotting on the HP2100A. For the most part the CDC documentation can be used on the HP (saving duplication of a 50 page document).

4. A major improvement, involving major changes, would be to try to utilize the Simultaneous-Print-Plot facility of the Versatec. At present the Plot Package does not allow this.

5. The graph utility could be improved by putting out scale
indicators on the margins and on the axes. Some experimentation
would be needed to arrive at the best possible combination.

6. Also a facility to put out headings with a special call to
either GRAPH or, maybe, an independent utility, could be
added to the graph utility.
APPENDIX A

Instructions for Implementing and Using the Plot Package

1) **Implementation** (Version II)

- first the implementation at McMaster will be given
- then the likely procedures necessary elsewhere.

a) Plot routines are compiled or assembled and the relocatable binary generated is added to the system library. (to actually add these routines a system generation is needed)

b) During system generation the routine SETHI can either be included in the system as a system utility or left as a user program. This does not make any significant difference. Before running any plot jobs a binary file of suitable length must be created and called SWAP2. Then the routine SETHI is run. This initializes SWAP2 for all subsequent uses of the plotter package. Only if someone purges SWAP2, or writes onto SWAP2, will the file need to be reinitialized.

- to implement this package on another HP with different core size would mean a few small changes in DSKIO and SETHI to adjust for the extra core. If much more core is available Version I of the package with expanded arrays would be much more attractive.

- if a different electrostatic printer/plotter were used with different resolution or with a different type of interface changes in constants and buffer sizes in PRINT
and BFCUT would be needed.

On different machines an all Fortran form of Version I would be easiest to use. Changes would probably be needed in BFCUT, for I/O on the particular machine.

2) Use of the Plot Package
   a) The user writes a program calling any special plotter routines, and possibly calling PLOT directly. The only restriction is that all calls to PLOT be compatible with the Benson-Lehner plotting system, and the reduced width of the printer plotter.
   b) Then the user compiles his program.
   c) Next load the program. This loading will automatically load plot routines from library.
   d) Execute the program.
   e) CDC documentation on the Benson-Lehner plotting system would be almost completely compatible with the HP plotting system as far as user is concerned.
APPENDIX B

Instructions for Using GRAPH.

At McMaster the GRAPH routine will be added to the library, along with the plot package. On a different machine this might be handled differently.

a) the organization of the package is shown in the diagram below. The user writes a main program and makes calls to GRAPH. In addition the user writes a Function subprogram which is called from GRAPH, and is an actual parameter of GRAPH. The format of the function is given below.

```
FUNCTION F'(X,I)
  1  F = SIN(X)
  Return
  2  F = COS(X)
  Return
```

The function the user supplies is exemplified by:
3 \[ P = X^X \]
Return

This allows multiple function definitions, so that several functions can be plotted at once.

The fortran call is exemplified by:

\[
\text{CALL GRAPH (XMIN, XMAX, XSCAL, YMIN, YSCAL, NOFN, FNM)}
\]

where

- \( \text{XMIN} \) - minimum X value for function (in absolute units)
- \( \text{XMAX} \) - maximum X value (in absolute units)
- \( \text{YSCAL} \) - desired scale in X-direction (absolute units / inch)
- \( \text{YMIN} \) - minimum Y value (in absolute units)
- \( \text{YSCAL} \) - scale in Y direction (absolute units/inch)
- \( \text{NOFN} \) - number of functions to be plotted
- \( \text{FNM} \) - is an external name of the function to be called.

b) The program and function are compiled and loaded normally (assuming GRAPH is in the system library). Then the program is executed, producing a graph.
APPENDIX C

Program Algorithms of Plot Package and Graph Utility

All algorithms are written in pseudo-PASCAL. This is done to take advantage of the structured programs, and data structures available in PASCAL. As much as possible legal PASCAL is used. However, some of the logic in the HP ASSEMBLER programs are beyond the ability of legal PASCAL to express. So in these cases the author took liberties with the language.

The algorithms appear in the following order:

PLOT, (page 81)
INVRT, (page 85)
PRINT, (page 86)
INSERT, (page 89)
BFOOT, (page 90)
DSKIO, (page 91)
SETHI, (page 94)
and GRAPH, (page 97)
PROCEDURE PLOT: VAR :R, YP : REAL,
        PEN : INT16;

" IN THIS ROUTINE THERE ARE 2 MAIN
DATA STRUCTURES

LIST -- WHICH IS A LINKED LIST OF
        STRINGTYPE (: STRING )
        THIS VALUE IS PASSED TO
"PRINT" & POINTS TO THE
1ST STRING IN THE LIST

STRING -- WHICH IS ALSO A LINKED LIST
        OF COORDINATES (: "COORD"
        OR LINE-SEGMENTS. IN THIS
ROUTINE "STRING" POINTS TO THE
CURRENT STRING IN "LIST"

LABEL 92.120.141;

TYPE
        INT16 = 0.1777778;

COORD = RECORD
        NXTPOINT : *COORD;
        X, Y : INT16;
        END;

STRINGTYPE = RECORD
        NXTSTRING : *STRINGTYPE;
        FIRSTPOINT : *COORD;
        END;

STRINGSTATE = (: NEW, UP, DOWN :);

VAR
        STATEOFSTRING : STRINGSTATE;
        X, Y, XOLD, YOLD : INT16;
        FIRSTTIME : BOOLEAN;
        STRING, LIST : *STRINGTYPE;
BEGIN " OF PLOT "

IF FIRSTTIME
THEN

BEGIN
INITIALISATION,
IF PEN = -20
THEN
BEGIN
CALLSINCENTIMETRES;
GOTO 92;
END;
END;

CONVERTTOPLOTTERUNITS(XR,'R.X.Y');

IF LEGAL(PEN)
THEN

120 : CASE PEN OF

-3,-2,-1 : BEGIN
RESETORIGIN;
PEN := -PEN;
GOTO 120;
END;

0 : SETCURRENTPOSITION(XR,YR);

1 : PLOTWITHPENUNCHANGED: "UP/DOWN"

2 : " PLOT WITH PEN DOWN "

CASE STATEOFSTRING OF

NEW "BEGIN"
ENDOLDSTRING.
CASE NOLD OF
TRUE STMPNEWSTRING DOWN,
FALSE STMPNEWSTRING UP,
END.
END.

DOWN BEGIN
IF NOLD THEN GOTO 141
ADDLINESegment(x,y),
END.

UP BEGIN
IF NOLD THEN,
BEGIN
INVRPT(STRING),
GOTO 141,
END,
ADDLINESegment(x,y),
END,
END, "CASE STATEOFSTRING"

3 "PLOT WITH PEN UP"
BEGIN
IF STATEOFSTRING = UP
THEN INVRPT(STRING);
STATEOFSTRING := NEW;
END;

40 SETNEWTRANSLATIONS;

999 "END OF THIS PLOT
PUT PLOT OUT TO
PRINTER/PLTTER"
BEGIN
IF STATEOFSTRING = UP
THEN INVRPT(STRING),
PRINT(LIST),
FIRSTTIME := TRUE;
END,
END "CASE PEN"
ELSE "PEN NOT LEGAL"

PRINT "ERROR"

END "OF POINT"
PROCEDURE INVPT:VPN FIRSTPOINT *COORD

VPN
NEXTPT. LASTPT *COORD

BEGIN " SUBROUTINE INVPT "

LASTPT = NIL " END-OF-STRING "

" GET POINTER TO NEXT POINT IN STRING
THEN POINT THIS CURRENT POINT TO PREVIOUS POINT

NOW SET LAST POINT TO CURRENT POINT
AND CURRENT POINT TO NEXT POINT

KEEP GOING TILL END-OF-STRING "

REPEAT

NEXTPT := FIRSTPOINT+.NEXTPOINT;
FIRSTPOINT+ NEXTPOINT := LASTPT;
LASTPT := FIRSTPOINT;
FIRSTPOINT := NEXTPT;

UNTIL
FIRSTPOINT = NIL

END. " OF INVPT "

PROCEDURE PRINT (VAR LIST : STRINGTYPE);

" IN THIS ROUTINE THERE ARE 2 MAIN
DATA STRUCTURES

LIST -- WHICH IS A LINKED LIST OF
STRINGTYPE : STRING
THE FORMAL ARGUMENT PASSED
TO THIS ROUTINE POINTS TO
THE 1ST STRING IN THE LIST

STRING -- WHICH IS ALSO A LINKED LIST
OF COORDINATES ("COORD"
OR LINE-SEGMENTS IN THIS
ROUTINE "STRING" ACTUALLY
POINTS TO THE CURRENT STRING
IN THE LIST

LABEL 35;

TYPE INT16 = 0.177777;

COORD = RECORD
  NXTPOINT : COORD;
  X, Y : INT16;
END;

STRINGTYPE = RECORD
  NXTSTRING : STRINGTYPE;
  FIRSTPOINT : COORD;
END;

VAR
  X, X1, X2, Y1, Y2, YMIN, YMAX, YLAST, YNEXT : INT16;
  LINE : ARRAY [1..35] OF INT16;
  STRING : STRINGTYPE;
BEGIN " OF PRINT."

" IN THIS ROUTINE (X1, Y1) AND (X2, Y2)
ARE USED TO SIMPLIFY THE ALGORITHM
THEM ARE USED AS ABBREVIATIONS FOR
X1 = FIRSTPOINT + X
Y1 = FIRSTPOINT + Y
X2 = FIRSTPOINT + NXTPOINT + X
Y2 = FIRSTPOINT + NXTPOINT + Y

ORDER(LIST),
PAGESKIP;
INITIALISATION;

WHILE SOME(LIST) DO " IE = LIST NOT NIL "
BEGIN

NEXT(X); " X CURRENT LINE POSITION "
FIRST(LIST); " GET POINTER TO 1ST STRING "

WHILE XIN(STRING) DO " X >,= X1 "
BEGIN

INITIALIZE(YMIN, YMAX), " RANGE INDICATORS IN LINE[] "

IF X < X2 " X2 = END-OF-1ST LINE-SEGMENT " THEN
BEGIN

INTERPOLATE(YLAST, YNEXT); IF X = X1 THEN YLAST = Y1;
MINMAX(YMIN, YMAX, YLAST);
MINMAX(YMIN, YMAX, YNEXT);
END
ELSE
BEGIN
IF X = X2 THEN
BEGIN

INTERPOLATE(YLAST);
IF X = X1 THEN YLAST = Y1;
MINMAX(YMIN, YMAX, YLAST);

END

END

END
ALGORITHM PRINT

"X = X2"
END: " IF X < X2 "

SKIPLINESegment.

IF DONE(STRING) " FIRSTPOINT+ NXTPOINT=NIL " THEN
DELETE(STRING).
ELSE
GOTO 35:
END: " IF X < X2 "

INSERTBITS(YMIN, YMAX); " INTO LINE "
NEXT(LIST).
END; "WHILE XIN(STRING)"

Bfout(LINE);
END. "WHILE SOME LIST"

Pageskip.
END: " OF PRINT "
PROCEDURE INSRT( VAR LINE : ARRAY [1..35] OF INT16;
        POINT : 1560);
TYPE INT16 = 0..1777777B; "16-BIT INTEGERS"
VAR
  I, J : INTEGER;
BEGIN "SUBROUTINE INSRT "
  I = (POINT+15) 16;
  J = POINT - 15 + 1 - 1;
  LINE(I) = INCLUSIVE(R(LINE(I)),BIT(J));
END. "OF SUBROUTINE INSRT"
PROCEDURE Bfout{(VAR LINE : ARRAY [1..35] OF INT16; )
TYPE INT16 = 0..177777B, "16 BIT INTEGERS"
VAR
  BUFF : ARRAY [1..35] OF INT16,
  J : INTEGER;
BEGIN "SUBROUTINE Bfout"
  BUFF = LINE, "COPY ARRAY LINE[]"
  CLEAR(LINE), "LINE[] := 0"
  WRITEBINARYWITHOUTWAIT(BUFF)
END: "OF SUBROUTINE Bfout"
"GLOBAL TYPES & VARIABLES"

TYPE
INT16 = 0..1777778,

VAR
BUFF : APRAY [ 1..128 ] OF INT16,

PROCEDURE PUT(VAR VALUE, INDEX, INT16);
BEGIN "PUT"

IF TOO_SMALL(INDEX)
THEN

REPORT ERROR "RN GE"
ELSE
BEGIN

IF TOO_LARGE(INDEX)
THEN
BEGIN,
ENDOLOSTRING : " INVRT & CLEAR",
STARTNEWSTRING : " AT OLD ( X, Y )"
END ; " IF TOO_LARGE "
ADD(VALUE, BUFF, INDEX);

END ; " IF TOO_SMALL "

END; " OF PUT "
FUNCTION LIST(VAR INDEX : INTEGER) : INTEGER
BEGIN " LIST"
  IF TOO_SMALL(INDEX) OR TOO_LARGE(INDEX)
  THEN
    REPORT ERROR " RANGE "
  ELSE
    SET(LIST, BUFF, INDEX),
  END; " OF LIST"

PROCEDURE CLEAR;
BEGIN " CLEAR"
  TRANSFER_STRING(BUFF, SCTR);
  " WRITE 'SCTR' TO DISC AS IT IS FILLED"
  END; " OF CLEAR"

PROCEDURE CLSEF;
BEGIN " CLSEF"
  CLEAR; " CURRENT STRING"
  EMPTY LAST SECTOR;
  REINITIALIZE POINTERS;
  END; " OF CLSEF"
PROCEDURE SWAP1(VAR TAGS,
                ITAG,
                IMAX, INT16,);)

VAR
    WORKFILE,
    SWAP2, FILE OF INT16;
    TRK, 0, 128;

BEGIN "SWAP1"

GETFREETRACKINWORKAREA(TRK).
WRITE(WORKFILE, HICORE, TRK, 0);
READ(SWAP2, HICORE);
SWAP2;
READ(WORKFILE, HICORE, TRK, 0);
INITIALIZE;

END; "OF SWAP2"
PROGRAM SETHI(SWAP2).

CONST CORESIZE = 375000.
    PARTITIONPOINT = 240000.

VAR
    BUFFERSIZE : 0.
    CORESIZE.
    SWAP2 : FILE OF INT16.

BEGIN "SETHI"

    BUFFERSIZE = CORESIZE - PARTITIONPOINT;
    SAVE(BASEPAGELINKS, CORESIZE-2000, 1000),
    WRITE(SWAP2, PARTITIONPOINT, BUFFERSIZE);

END. "SETHI"
PROCEDURE SWAP2(YAP TRK, TRACK : 0, 199,
TAGS ARRAY[1, 500] OF INT16,
ITAG, IMAX, INT16);

CONST FIRSTWORDAVAILABLEMEMORY = 10000B;

TYPE
  INT16 = 0 .. 17777777;

VAR
  WORKFILE, FILE OF INT16;

BEGIN "SWAP2"

GETSPECIALARGUMENTS;
SAVE(BASEPAGELINKS, CORESIZE-100B, 100B);
RESET(BASEPAGELINKS, CORESIZE-200B, 100B);

"SAVE CURRENT LO-CORE, (1000B - 23777B)"
WRITE(WORKFILE, LOCORE);

"MOVE TAGS ARRAY DOWN TO 10000B"
SHIFT(TAGS, FIRSTWORDAVAILABLEMEMORY);

"READ IN 'LIST()' FROM DISK TO FOLLOW DIRECTLY
AFTER 'TAGS()'"
READ(WORKFILE, FIRSTWORDAVAILABLEMEMORY+ITAG);
" NOW CALL 'PRINT' AS PER NORMAL"
PRINT(LIST, TAGS, ITAG);
"PUT LO-CORE BACK INTO CORE JUST AS IT WAS."

END "SMALL"
PROCEDURE GRAPH (MIN, MAX, SCAL, YMIN, YSCAL, REAL, HMIN, HSCALE, INTEGER X)

TYPE INT16 = 0:1:32:

VAR LINE [1..35] OF INT16;

YLAST, YPPREV, YNEXT, YMIN, YMAX, YAXIS INT16;

DELTX, DELTY, YMAX, N REAL:

BEGIN " OF GRAPH "

CALCULATESTEPsize(DELTX, DELTY);
CALCULATE(YAXIS);
PAGESKIP;

REPORT(YMIN);
TOPORBOTTOMORDER;

FIRST(X);
WHILE X < XMAX DO

BEGIN
IF X = 0 THEN " SPECIAL - YAXIS "
PLOT(YAXIS)
ELSE
BEGIN "NORMAL CASE"

FOR EACH FUNCTION DO
BEGIN
INTERPOLATE(X, YLAST, YPRES, YNEXT),
GETRANGE(YMIN, YMAX, YLAST, YPRES, YNEXT),
INSPT(YMIN, YMAX),
END. "FOR EACH FUNCTION"

ADDBORDERSANDAXIS;
Bfout(LINE);
END. "NORMAL CASE"

END; "WHILE X < XMAX"

TOPORBOTTOMBORDER;
CALCULATE(YMAX);
REPORT(YMIN, YMAX, XMAX, XSCALE, YSCALE);

PAGESHIFT;

END. "OF GRAPH"
APPENDIX D

Program Listings of Plot Package
and Graph Utility

The program listings appear in the following order:

PLOT, (page 100)
INVRT, (page 113)
PRINT, (page 115)
ORDER, (page 125)
INSRT, (page 128)
Bfout, (page 135)
Dskio, (page 138)
SETHI, (page 164)
and GRAPH. (page 183)
SUBROUTINE PLOT

FTH4, LT

SUBROUTINE PLOT (NR, YR, PEN)

C

SUBROUTINE "PLOT"

C

WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975

C

BY C. A. BRYCE, FOR A MSC PROJECT

C

"PLOT" IS THE BASE ROUTINE WRITTEN FOR THIS "PLOT" PACKAGE. THIS
ROUTINE IS DESIGNED TO BE A REPLACEMENT FOR THE NORMAL
PLOT ROUTINE IN A PLOTTING SYSTEM IN PARTICULAR THIS
PLOT ROUTINE WAS WRITTEN TO BE COMPLETELY COMPATIBLE WITH THE
"BENSON-LEHNER PLOTTING SYSTEM" AS IMPLEMENTED ON THE CDC6400
AT MCMaster UNIVERSITY. ALL THE ROUTINES THAT NORMALLY CALL
"PLOT" CAN SUCCESSFULLY CALL THIS "PLOT" ROUTINE.
THIS ROUTINE TAKES THE CALLS TO PLOT AS THEY COME IN, AND
REORGANIZES AND RECODES THESE CALLS SO THAT EVENTUALLY THE
ROUTINE "PRINT" CAN DECODE THEM AND PUT OUT THE DESIRED PLOT.

C

THERE ARE 2 VERSIONS OF "PLOT". ONE VERSION USES ARRAY STORAGE, WHICH LIMITS
THE POTENTIAL SIZE OF PLOT THAT "PLOT" CAN HANDLE, AND THE OTHER USES DISK
STORAGE, WHICH INCREASES THE POTENTIAL SIZE BY A FACTOR OF ABOUT 3
THIS INCREASE FACTOR IS ONLY DUE TO THE LIMITED CORE SIZE OF 12K (DECIMAL)
AND WITH A LARGER CORE OF 24K, IT WOULD BE MUCH PREFERABLE TO USE THE ARRAY VERSION.

C

VERSION 1 OF "PLOT"
SUBROUTINE PLOT  --  MARCH 1, 1975  PAGE # 2

; ARRAY STORAGE

; THIS WAS THE 1ST VERSION CREATED AND
; BASILICHER THE STRINGS CREATED IN PLOT
; ARE STORED IN AN INTEGER ARRAY "LIST:"
; THIS WAS BY FAR THE PREFERABLE AND
; SIMPLEST WAY TO HANDLE THINGS. HOWEVER,
; "LIST:" COULD ONLY BE 2000 WORDS. WHY
; THIS ALLOWED JUST UNDER 1000 CALLS TO PLOT.

; VERSION I OF "PLOT" USES 2 SUBROUTINES

; 1) INVRT - THIS ROUTINE ALLOWS THE SIMPLE
; INVERSION OF A STRING. THIS IS
; USED AS A SUBROUTINE AS IT IS
; NEEDED IN 3 PLACES

; 2) PRINT - THIS ROUTINE IS CALLED ON
; COMPLETION OF A PLOT (OR
; UPON OVERFLOW OF AVAILABLE
; STORAGE). "PRINT" DOES THE
; PHYSICAL PLOTTING ONTO THE
; PRINTER-PLOTTER

; VERSION II OF "PLOT"
; ( DISK STORAGE )

; BY USING DISK STORAGE FOR STRINGS
; CREATED IN PLOT, THE EFFECTIVE SIZE WAS
; EXTENDED BY ABOUT 3500 WORDS. (OR
; ABOUT 1700 CALLS TO PLOT )

; VERSION II OF "PLOT" USES 6 SUBROUTINES

; 1) IVRTA - THIS IS AN ASSEMBLER ROUTINE
; THAT FORMS A CALL TO "INVRT"
; USING ARGUMENTS LOCAL TO THE
; DISK I/O ROUTINE

; 2) PUT - THIS EFFECTIVELY ACCOMPLISHES AN
; ASSIGNMENT.
; [ LIST ('i') = .... ]

; 3) LIST - THIS FUNCTION ACCOMPLISHES AN
; ARRAY ACCESS WITHOUT ANY NEEDED
CHANGE IN THE CODING EXCEPT TO
UNDIMENSION "LIST".
EQUIVALENT TO
[ = LIST < I > ]

4) CLEAR - THIS ROUTINE EMPTIES A COMPLETED STRING FROM THE RECORD BUFFER IN THE DISK I/O PACKAGE

5) CLOSEF - THIS "CLEAR" S LAST STRING OUT, AND PUTS LAST BUFFER OUT TO THE DISK

6) SWAP1 - THIS ACCOMPLISHES THE 1ST PART OF THE CORE SWAPPING PROCESS, AND EVENTUALLY LINKS TO "PRINT"

IN BOTH VERSIONS THE EFFECTIVE LOGIC IS THE SAME, WITH THE EXCEPTION OF REFERENCES TO ITEMS IN "LIST("), AND OF THE INCLUDED CLEARING & CLOSING OPERATIONS IN VERSION II.

METHOD

IN THE 1ST CALL TO "PLOT" INITIALISATION IS DONE TO FLAGS, ORIGIN, TRANSPOSITION, ETC., AND CHANGING "FACTOR" TO CENTIMETRES IS ALLOWED,
THEN <XR,YR> IS CONVERTED FROM REAL TO INTEGER PLOTTER UNITS <X,Y>, THEN ORIGIN, AND TRANSLATION CHANGES ARE MADE TO GET
AN <X,Y> IN ABSOLUTE PLOTTER UNITS (FROM <0,0>) NEXT "PEN" IS DECODED AND NECESSARY CODE IS EXECUTED IN ORDER TO ---
- ADD TO CURRENT STRING
- START A NEW STRING
- RESET ORIGIN OR TRANSLATION
- RETURN CURRENT CO-ORDINATES
- OR TERMINAL CALL TO "PLOT"
- INITIATING A CALL TO "PRINT"

KEY: VARIABLES:
SUBROUTINE PLOT

C
C   XP   - REAL CALLING VALUE OF X CO-ORDINATE
C   YP   - REAL CORRESPONDING VALUE OF Y
C   PEN   - INTEGER CALLING VALUE OF "PEN"
C       "PEN" IS-CODED TO TELL "PLOT" WHAT TO DO WITH (XP,YP)
C
C REAL XP,YP
C INTEGER PEN

C LOCAL VARIABLES
C
C X,Y   - INTEGER WORKING VALUE OF CO-ORDINATES
C       CORRESPONDING TO (XP,YP)
C
C XOLD, YOLD
C   - INTEGER CO-ORDINATES OF CURRENT
C       POSITION ( OR CO-ORDINATES FROM
C       PREVIOUS CALL TO "PLOT")
C
C XORG, YORG
C   - INTEGER CO-ORDINATES OF CURRENT
C       ORIGIN ( RELATIVE TO (0,0) )
C
C XTR, YTR
C   - INTEGER CO-ORDINATES OF CURRENT
C       TRANSLATION ( TO BE TAKEN AWAY
C       FROM (X,Y) )
C
C FLAG   - INTEGER FLAG TO INDICATE A STRING
C GOING DOWN ( FLAG=0 )
C GOING UP  ( FLAG=1 )
C OR NEITHER ( FLAG=-1 )
C
C PENP   - INTEGER FLAG INDICATING CURRENT
C PEN POSITION.
C   ( =2, PEN DOWN ; =3, PEN UP )
C
C FACTOR - REAL CONVERSION FACTOR USED TO
C CONVERT (XP,YP) TO PLOTTER UNITS.
C   ( BY DEFAULT SET FOR CALLS IN
C INCHES, BUT CAN BE SET TO
C CENTIMETRES ON 1ST CALL (ONLY) )
SUBROUTINE PLOT  --  MARCH 1, 1975  PAGE # 5

LIST() - INTEGER ARRAY (VERSION I) TO HOLD
    STRINGS. IN VERSION II - A FUNCTION.

I, IX - INDICES USED WITH "LIST()"

THGS() - INTEGER ARRAY HOLDING POINTERS
    INDICES TO HEAD OF STRINGS IN
    "LIST()

ITAG - CURRENT INDEX IN "TAGS()"

MAXL, MXT - MAXIMUM SIZE FOR "LIST()" & "TAGS()"
    RESPECTIVELY
    ( IN VERSION I -- (2000, 500)
    & IN VERSION II -- (5500, 500) )

Kount - INTEGER FLAG TO INDICATE 1ST CALL
    TO "PLOT" (=0) OR NOT (=1).

IW - LOGICAL UNIT OF PRINTER (=8)

INTEGER X, Y, XOLD, YOLD, XORG, YORG
INTEGER XTR, YTR
INTEGER FLAG, PENP, KOUNT, IW
INTEGER LIST, I, IX, TAGS, ITAG
INTEGER MAXL, MXT
REAL  FACTOR
DIMENSION TAGS (500)

DIMENSION LIST (2000) -- VERSION I

INITIALIZE DATA

DATA  KOUNT  /  0  /
DATA  IW    /  8   /
DATA  MAXL /  5500 /
C# DATA  MAXL /  2000  -- VERSION I
 DATA  MXT  /  500  /

*******************************************************************************


SUBROUTINE PLOT --- MARCH 1, 1975 PAGE # 5

C 'START OF SUBROUTINE "PLOT"
C ON THE 1ST CALL TO "PLOT" (KOUNT=0)
C
C INITIALIZE
C LAST X, Y: -- (XOLD, YOLD)
C ORIGIN: -- (XORG, YORG)
C TRANSLATION: -- (XTR, YTR)
C PEN POSITION: PEHP
C UP/DOWN FLAG: FLAG
C INDEX IN "LIST0" & "LIST1"
C INDEX IN "TAGS": ITAG
C SCALE-FACTOR: FACTOR
C & SAY NOT 1ST CALL TO "PLOT" (KOUNT=1)
C
C IF(KOUNT) 92, 90, 100
.90  KOUNT = 1
   XOLD = 0
   YOLD = 0
   XORG = 0
   YORG = 0
   XTR = 0
   YTR = 0
   PEHP = 3
   I = 1
   CALL PUT( 1, 1, I, FLAG, TAGS, ITAG, MAXT )
   ITAG = 0
   FLAG = -1
   FACTOR = 72.5

C ON 1ST CALL IF PEN = -20 SET
C FACTOR TO CENTIMETRES INSTEAD
C OF INCHES AND RETURN
C
C IF(PEN+20) 2000, 91, 100
.91  FACTOR = 0.3937 * FACTOR
.92  RETURN
C
C**************************************************************************
C AFTER 1ST CALL THIS IS THE EFFECTIVE
C STARTING POINT IN "PLOT".
CONVERT (XR, YR) FROM REAL IN INCHES (CM) INTO INTEGER (X, Y) IN PLOTTER UNITS.

THEN APPLY ORIGIN (XORG, YORG) AND TRANSLATION (XTR, YTR) TO GET AN (X, Y) IN ABSOLUTE PLOTTER UNITS (RELIATIVE TO (0, 0)).

100 X = FACTOR + XR
    Y = FACTOR + YR
    X = X + XORG - XTR
    Y = Y + YORG - YTR

HOW INTERROGATE "PEN" IN ORDER TO DETERMINE WISHES OF USER.

IF "PEN" HAS THE VALUE
-3, -2, -1 - SET NEW ORIGIN TO (X, Y) CHANGE SIGN OF "PEN" & PROCEED NORMALLY (=> 120)?
0 - RETURN CURRENT POSITION IN (XR, YR) IN PLOTTER UNITS
1 - (=>121) & TAKE LAST PEN POSITION AS CURRENT ONE
2 - (=>140) & PLOT WITH PEN DOWN
3 - (=>127) & PLOT WITH PEN UP
40 - (=>126) & RESET TRANSLATION TO (X, Y)
999 - (=>999) & PUT-OUT PLOT
OTHER - (=>2000) & REPORT ERROR IN CALL TO "PLOT".

IF(PEN) 130, 128, 120
120 IF(PEN-1) 124, 121, 122
121 IF(PENP-2) 127, 140, 127.
SUBROUTINE PLOT

122 IF (PEN=3) 140, 127, 123
123 IF (PEN=40) 125, 129, 124 C*
124 IF (PEN=999) 125, 999, 125
125 GO TO 3000
C
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C SET TRANSLATION (XTR, YTR)
C
126 XTR = X - XOR + XTR
YTR = Y - YOR + YTR
RETURN
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C 'PLOTTING WITH PEN UP ("PEN"=3)
C
C IF THE LAST STRING JUST FINISHED
C AND WAS GOING "UP" (FLAG=1), THEN
C CALL "INVRT" AND INVERT STRING
C
C THEN RESET (XOLD, YOLD) & SET FLAG
C TO INDICATE NO ACTIVE STRING EXISTS
C
C
C$127 IF (FLAG.EQ.1) CALL INVRT(LIST(I), I)
127 IF (FLAG.EQ.1) CALL IVRTA
XOLD = X
YOLD = Y
FLAG = - 1
RETURN
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C RETURN CURRENT CO-ORDINATE POSITION
C (IN PLOTTER UNITS), RELATIVE TO PLOTTER
C ORIGIN (0,0)
C
128 XR = FLOAT(XOLD)
YR = FLOAT(YOLD)
RETURN
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C "PEN" = -3, -2, -1
SUBROUTINE PLOT  --- MARCH 1, 1975

C
C THEN RESET ORIGIN (XPG, YORG) TO
C (X, Y). SWITCH SIGN ON PEN. X => 120.
C DECODE PEN AGAIN.
C
C IF "PEN" = -3. => 2000 & REPORT ERROR
C
130 IF (PEN+3) 2000, 131, 131
131 XPG = X.
YPG = Y.
PEN = -PEN.
GO TO 120
C
C**************************************************************
C
C PLOTTING WITH PEN DOWN
C
C .SET PEN POSITION TO DOWN (PENP=2)
C
C WILL THERE BE OVERFLOW IN "LIST()"?
C IF SO, => 3000 & REPORT IT
C
C WORKING ON A STRING, IF SO WHAT KIND ??
C =-1, =>141. NEW STRING
C = 0, =>145. DESCENDING STRING
C & = 1, =>160. ASCENDING STRING.
C
140 PENP = 2
IX = LIST ("I")
IF ( (IX+1).GT.MAXL ) GO TO 3000
IF ( FLAG ) 141, 145, 160
C
C START NEW STRING
C
C .RESET "I"
C
C ( CLEAR OUT LAST RECORD - VERSION II )
C
C PUT A WORD OF ZEROS INTO "LIST()"
C
C IS THERE ROOM FOR NEXT LINE SEGMENT
C ( (X1,Y1), (X2,Y2) ) OR WILL THERE
C BE OVERFLOW IN "LIST()" ?
C IF SO, => 3000 & REPORT IT.
C - OVERFLOW IN "TAGS:"?
C IF SO. = 3000 & REPOP IT
C
C - PUT POINTER TO HEAD OF STRING INTO
C "TAGS:""
C
C - DESCENDING OR ASCENDING STRING
C : HOLD
C
C 141 I = LIST ( I )
CALL CLEAR
IF ( I GT MAXL ) GO TO 3000
CALL PUT ( 0, I, I, FLAG, TAGS, ITAG, MAXT )
C$ LIST ( I ) = 0
I = I + 1
IF ( (I+4) GT MAXL ) GO TO 3000
CALL PUT ( I+5, I, I, FLAG, TAGS, ITAG, MAXT )
C$ LIST ( I ) = I + 5
ITAG = ITAG + 1
IF ( ITAG GT MAXT ) GO TO 3000
TAGS ( ITAG ) = I + 1
IF ( X-XOLD ) 143, 142, 142
C
C PUT A DESCENDING LINE SEGMENT INTO
C "LIST()", IN NORMAL ORDER, SET FLAG=0
C
C 142 FLAG = 0
CALL PUT ( XOLD, I+1, I, FLAG, TAGS, ITAG, MAXT )
CALL PUT ( YOLD, I+2, I, FLAG, TAGS, ITAG, MAXT )
CALL PUT ( X, I+3, I, FLAG, TAGS, ITAG, MAXT )
CALL PUT ( Y, I+4, I, FLAG, TAGS, ITAG, MAXT )
C$ LIST ( I+1 ) = XOLD
C$ LIST ( I+2 ) = YOLD
C$ LIST ( I+3 ) = X
C$ LIST ( I+4 ) = Y
GO TO 180
C
C PUT AN ASCENDING LINE SEGMENT INTO
C "LIST()" IN REVERSE ORDER.
C
C IE = ( Y1, X1, Y2, X2 )
C
C SET FLAG=1
143  FLAG = 1
    CALL PUT( XOLD, I+2, I, FLAG, TAGS, ITAG, MAXT )
    CALL PUT( YOLD, I+1, I, FLAG, TAGS, ITAG, MAXT )
    CALL PUT( X, I+4, I, FLAG, TAGS, ITAG, MAXT )
    CALL PUT( Y, I+3, I, FLAG, TAGS, ITAG, MAXT )

C$  LIST ( I+2 ) = XOLD
C$  LIST ( I+1 ) = YOLD
C$  LIST ( I+4 ) = X
C$  LIST ( I+3 ) = Y
    GO TO 180

C******************************************************************************
C
C CURRENTLY IN A DESCENDING STRING
C
C - CHECK FOR REVERSAL ( X < XOLD )
C   IF SO, THIS MUST BE THE START OF
C   AN ASCENDING STRING.
C   SO => 141 & START A NEW STRING
C
C - IF NO PROBLEM PUT (X,Y) INTO
C   STRING, THEN => 179
C
C 145  IF ( X-XOLD ) 141, 150, 150
C    CALL PUT( X, IX, I, FLAG, TAGS, ITAG, MAXT )
C    CALL PUT( Y, IX+1, I, FLAG, TAGS, ITAG, MAXT )
C$ 150  LIST ( IX ) = X
C$  LIST ( IX+1 ) = Y
    GO TO 179

C******************************************************************************
C
C CURRENTLY IN AN ASCENDING STRING
C
C - CHECK FOR REVERSAL ( X > XOLD )
C   IF SO, THIS MUST BE THE START OF
C   A DESCENDING STRING.
C   So INVERT CURRENT STRING BEFORE
C   STARTING A NEW STRING ( => 141 )
C
C - IF NO PROBLEM PUT (X,Y) INTO
C   STRING IN REVERSE ORDER
C   ( IE = Y, THEN X )
SUBROUTINE PLOT  ---  MARCH 1, 1975

C
160 IF ( XOLD-X ) > 161  170  170
161 CALL IVRTA
C$161 CALL INVRT ( LIST(I), I )
GO TO 141
170 CALL PUT ( Y, X, I, FLAG, TAGS, ITAG, MAXT )
CALL PUT ( IX+1, I, FLAG, TAGS, ITAG, MAXT )
C$170 LIST ( IX+1 ) = Y
C LIST ( IX+1 ) = X
C UPDATE "LIST(I)" TO "LIST(I)+2"
C SO THAT THIS POINTS TO NEXT AVAILABLE
C LOCATION IN "LIST()"
C
C THEN SET (XOLD, YOLD) <= (X, Y)
C
179 CALL PUT ( LIST(I)+2, I, I, FLAG, TAGS, ITAG, MAXT )
C$179 LIST ( I ) = LIST ( I ) + 2
180 XOLD = X
YOLD = Y
RETURN
C
C******************************************************************************************
C PLOT IS FINISHED, OR AT LEAST THE CALLS
C TO PLOT ARE. INVERT THE LAST STRING IF
C IT IS NECESSARY, AND SET I TO "LIST(I)"
C
C IN VERSION I -- SIMPLY CALL "PRINT"
C
C WHILE IN VERSION II --
C CALL "CLEFT" TO PUT OUT LAST
C BIT OF STORAGE
C THEN CALL "SWAP" TO PERFORM
C NEEDED SWAPPING BEFORE GET-
C TING TO "PRINT"
C
C IN BOTH VERSIONS, UPON RETURNING SET
C "KOUNT" TO INDICATE 1ST TIME SO THAT
C ANOTHER PLOT IS POSSIBLE FROM
C SCRATCH.
C
C 999 IF ( FLAG.EQ.1 ) CALL IVRTA
C$999 IF ( FLAG.EQ.1 ) CALL INVRT ( LIST(I), I )
I = LIST(I)
CALL CLEFT
CALL SWAP ( TAGS, ITAG, I)
CALL PRINT ( LIST, TAGS, ITAG )
NUMBER = 0
RETURN

+++++++++++++++++++++++++++++++ ERROR MESSAGE TO INDICATE ILLEGAL PEN ENTRY
2000 WRITE ( IW, 1 ) X, Y, PEN
   1 FORMAT ( 10X, 20H*ERROR - THERE IS
   .27AHAN ERROR IN PEN IN ENTRY,
   2 /, 10X, 3HX = , I4, 5X, 3HY =,
   3 I4, 5X, 5HPEN = , I4, / )
   RETURN

+++++++++++++++++++++++++++++++ ERROR MESSAGE FOR OVERFLOW IN ONE
OF "LIST()", OR "TAGS()"
AFTER ERROR MESSAGE ( => 999 ) & ATTEMPT TO PLOT WHAT HAS BEEN ACCUMULATED TO DATE
3000 I LIST = LIST + I
   WRITE ( IW, 2 ) I, I LIST, ITAG
   2 FORMAT ( 10X, 20H+ERROR - OVERFLOW L
   , 7HIST ( , I4, 5H ) = , I5, /
   2 30X, 7HITAG = , I5 )
   GO TO 999
   END
   END$
SUBROUTINE INVRT --- MARCH 1, 1975

```
SUBROUTINE INVRT ( LIST, I )

SUBROUTINE "INVRT"

WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975

BY C. A. BRAYCE, FOR A MSC PROJECT

"INVRT" IS A SIMPLE UTILITY ROUTINE USED
ONLY BY "PLOT". IT IS USED TO
REVERSE OR INVERT THE STRING
IN ARRAY "LIST()" AS FAR
AS THIS ROUTINE IS CONCERNED.
THE STRING STARTS IN "LIST(1)"
THIS IS A POINTER TO START OF
NEXT STRING, AND FROM THIS AND
"I" THE LENGTH OF THE STRING
CAN BE DETERMINED. THEN BY
GETTING THE INDEX OF THE START
OF THE NEXT STRING ( "KLAST" )
SIMPLY INTERCHANGE --
LIST(2) <= LIST(KLAST-1)
LIST(3) <= LIST(KLAST-2)

TILL LIST(N) <= LIST(M)
WHERE M = N+1

KEY VARIABLES

LIST() - INTEGER ARRAY CONTAINING STRINGS
I - INTEGER VALUE OF CURRENT POSITION
    IN "LIST()"

INTEGER LIST, I
DIMENSION LIST ( 0 )
```
SUBROUTINE INVPT    --- MARCH 1, 1975    PAGE 2

C LOCAL VARIABLES
C KLAST, K, KK
   - INDICES AND COUNTERS USED IN LOOP
C I1, I2 - FINAL INDICES USED WITH "LIST()"
C ISTORE - USED FOR INTERCHANGING

INTEGER KLAST, K, KK, I1, I2, ISTORE
C*************************************************************************

KLAST = LIST(1) - I + 1
K = (KLAST - 1) / 2
DO 210 KK = 1, K
   I1 = KK + 1
   I2 = KLST - I1
   ISTORE = LIST(I1)
   LIST(I1) = LIST(I2)
210   LIST(I2) = ISTORE
RETURN
END
END.
SUBROUTINE PRINT

SUBROUTINE "PRINT"

WRITTEN AT McMASTER UNIVERSITY, FEBRUARY 1375
BY G. A. BPYCE, FOR A MSC PROJECT

"PRINT" IS CALLED FROM SUBROUTINE "PLOT"
EITHER DIRECTLY IN VERSION I,
OR INDIRECTLY IN VERSION II
IT'S PURPOSE IS TO DO THE ACTUAL
PHYSICAL PLOTTING TO THE
ELECTRO-STATIC PRINTER/PLOTTER.
IT PLOTS THE STRINGS IN "LIST()
USING THE TAG ARRAY "TAGS()
AND THE NUMBER OF TAGS "ITAG".

"PRINT" USES 4 SUBROUTINES.

1) EXEC - THE SYSTEM EXECUTIVE TO GENERATE
PAGE THROWS BEFORE & AFTER PLOTTING

2) ORDER - A ROUTINE THAT DOES A TAG-SORT
ON "LIST()", WITH TAGS "TAGS()"
AND NUMBER OF TAGS "ITAG"

3) INSRT - A ROUTINE THAT ADDS NECESSARY BITS
INTO OUTPUT BUFFER "LINE()"

4) BFOUT - THE ROUTINE THAT PUTS "LINE()" OUT
TO THE PRINTER/PLOTTER

THIS ROUTINE ACCOMPLISHES ITS PURPOSE BY
FIRST DOING A TAG-SORT ON "LIST()", "TAGS()
AND "ITAG". THIS PUTS STRINGS INTO
ASCENDING ORDER SO THAT THE REST OF THIS
ROUTINE CAN WORK PROPERLY.
THEN LINE BY LINE, (X=0,1,2,3,...), PASS
DOWN THE PAGE, FIRST PUTTING OUT BLANK
LINES, ("LINE()"=0) UNTIL X REACHES FIRST
SUBROUTINE PRINT  -- MARCH 1, 1975.

STRING = LIST:THGS(1) THEN START
PLOTTING BY PUTTING THE Y POINTS AS THEY OCCUP IN THE 'STRINGS AND ARE IN THE RIGHT RANGE THE 1ST, 2ND, 3RD STRING IS CHECKED, AND THE NEEDED POINTS PUT INTO "LINE()" UNTIL A STRING IS MET THAT STARTS BELOW CURRENT X VALUE. "LIST:TAGS:" ALSO TO SAVE EXCESSIVE WORK IF THE FIRST STRING IS EXHAUSTED, IT IS NOT CHECKED. ALSO, NOPE, AND THIS GOES FOR THE NEXT 'FIRST' STRING. THUS ONLY THE STRINGS STARTING AT THE FIRST STRING WITH SOMETHING TO PLOT FOR PPREVIOUS X (X-1) TO THE FIRST STRING PAST WHERE PLOTTING OCCURS FOR CURRENT X IS CHECKED WHEN NO MORE STRINGS EXIST TO PLOT WE ARE DONE TO IMPROVE THE QUALITY OF THE PLOT, THE PRINTER IS PAGED BOTH BEFORE AND AFTER PLOTTING.

KEY VARIABLES

LIST() - INTEGER ARRAY CONTAINING STRINGS OR RECORDS TO BE PLOTTED

TAGS() - INTEGER ARRAY CONTAINING TAGS OR POINTERS TO START OF STRINGS IN "LIST()"

ITAG - NUMBER OF STRINGS IN "LIST()"

INTEGER LIST, TAGS, ITAG
DIMENSION LIST (), TAGS ()

LOCAL VARIABLES

LINE() - INTEGER BUFFER USED TO STORE PLOTTED POINTS (INITIALLY=0)

MAX - INTEGER CONSTANT OF NUMBER OF POINTS IN "LINE()+1 (< 561)

BIAS - INTEGER CONSTANT OF BIAS USED IN STORING SLOPE.
SLPM: - REAL CONSTANT OF MAXIMUM ALLOWABLE SLOPE X BIAS (EFFECTIVELY 561 0)

X: - INTEGER VALUE IN PLOTTER UNITS OF CURRENT X CO-ORDINATE INITIALLY = -1

IMIN: MINIMUM INDEX IN "TAGS<" TO BE CHECKED (INITIALLY 0)

I: - INDEX USED FOR "TAGS<"

IX1: - WHEN CONSIDERING A LINE SEGMENT (X1, Y1) TO (X2, Y2) IN ORDER TO GET THE SLOPE IX1 IS THE INDEX IN "LIST<" OF X1

IY1: - INDEX OF Y1, (IX1+1)

IX2: - INDEX OF X2, (IX1+2)

IY2: - INDEX OF Y2, (IX1+3)

ISL: - INDEX WHERE BIASED SLOPE FOR CURRENT LINE SEGMENT IS IN "LIST<", (IX1-2)

LAST: - INDEX OF START-OF-NEXT STRING

SLOPE: - REAL VALUE OF SLOPE

YMIN: - MINIMUM Y VALUE FOR CURRENT STRING IN "LINE<"

YMAX: - MAXIMUM CORRESPONDING TO YMIN

YLAST: - LAST Y VALUE IN LINE SEGMENT

YPRES: - PRESENT Y. VALUE IN LINE SEGMENT

YNEXT: - NEXT Y VALUE IN LINE SEGMENT

Y: - INDEX USED IN FILLING "LINE<"
INTEGER LINE, MAX, BIAS, X, IMIN, I
DIMENSION LINE(35)
INTEGER I:1, IY1, IY2, ISL, LAST
PEAL SLPMX, SLOPE
INTEGER Y, YMIN, YMAX
INTEGER YLAST, YPRES, YNEXT

INITIALIZE DATA
DATA LINE / 35+0 /
IN ABOVE DATA STATEMENT 35+0 IS NON-ANSI
THIS SETS 35 WORDS IN LINE() TO 0
DATA MAX, BIAS / 561, 58 /

START OF SUBROUTINE "PRINT"

PUT "STRINGS" IN ARRAY LIST() INTO
ASCENDING ORDER BY DOING A TAG
SORT ON [ LIST(), TAGS(), & ITAG ].

CALL ORDER( LIST, TAGS, ITAG )

PERFORM A PAGE EJECT SO THAT PLOT
STARTS ON TOP OF PAGE.

CALL EXEC ( 3, 1110B, 63 )

INITIALIZE -- % & IMIN TO -1
-- CALCULATE "SLPMX", SLOPE-MAX.
THEN SKIP THE BUFFERING ( BY GOING TO 11 )

SLPMX = FLOAT(MAX*BIAS)
X = -1
IMIN = 0
GO TO 11

C

BUFFERS OUT CONTENTS OF LINE

CALL BOUT (LINE)

INCREMENT I & START I AT IMIN

X = X + 1
I = IMIN

IN THE NEXT BLOCK OF CODE, BY STARTING
WITH I AT IMIN+1 AND WHILE I IS LESS
THAN ITAG, THE STRINGS ARE SEARCHED
FOR EXISTING STRINGS (TAGS(I) > 0)

IF A STRING IS FOUND, PROCEED TO 30
TO DO NECESSARY CALCULATIONS,
AND IF NOT CONTINUE SEARCHING
( I <= I + 1 ), BUT REMEMBERING
TO ELIMINATE DEAD STRINGS AT
BEGINNING. ( TAGS(I)=0, & I=IMIN+1 )

ALSO WHEN IMIN=ITAG, THERE ARE NO MORE
STRINGS TO PLOT, SO RETURN

I = I + 1
IF ( I.GT.ITAG ) GO TO 10
IX1 = TAGS (I)
IF ( IX1.GT.0 ), GO TO 30
IF ( IMIN.LT.(I-1) ) GO TO 20
IMIN = IMIN + 1
IF ( IMIN.LT.ITAG ) GO TO 20:

CALL EXEC (3, 1110B, 63)
RETURN

C

******************************************************************************
SUBROUTINE PRINT --- MARCH 1, 1975.

C
C INITIALIZE YMIN & YMAX FOR THIS STRING
C
30  YMIN = MAX
    YMAX = 0.
C
C CALCULATE INDEXES IN LIST() FOR
C ISL, LAST, IY1, IX1, IX2, & IY2:
C
32  IX1 = IX1 - 2
    IX2 = IX1 - 1
    LAST = IX1 - 1
    IY1 = IX1 + 1
    IX2 = IX1 + I
    IY2 = IX2 + 1
C
C IF "X1" < 0  =>  35 AND CALCULATE SLOPE
C WHEN "X1" < "X2"
C
C OTHERWISE ONLY CALCULATE SLOPE WHEN
C X = "X1" < "X2".
C
C IN THE CASE WHEN X < "X1", THIS IS 1ST
C STRING PAST THE STRINGS TO BE PUT
C OUT SO BUFFER-OUT LINE() ( => 10 )
C
C IF ( LIST(IX1).LT.0 ) GO TO 35
C IF ("X-LIST(IX1)") 10, 35, 40
35 IF ( LIST(IX1).EQ.LIST(IX2) ) GO TO 60
    SLOPE = FLOAT( BIAS*( LIST(IY1)-LIST(IY2) ) )
    SLOPE = SIGN(AMIN1(ABS(SLOPE),SLPMX),SLOPE)
    LIST ( ISL ) = IFIX ( SLOPE )
C
C NOW CONSIDER  X = "X2",
C
C - NORMAL CASE  X < "X2"  =>  70
C ( "X1" => X < "X2"
C
C - END-OF-LINE-SEGMENT WHEN X = "X2"
C => 60 ( HANDLE SPECIAL-CASES )
SUBROUTINE PRINT

- AT START WHEN X=0, IT IS POSSIBLE
  FOR - "X1" =, "X2" < X = 0
  -- SKIP THIS LINE SEGMENT

40 IF ('X-LIST(I*X2') .GT. 60, GO TO 30
50 LIST (IY1) = LAST
     IX1 = IX2
     TAGS (I) = IX1
     IF ( IX2 LT LAST ) GO TO 32

STRING IS FINISHED ( IX2=LAST )

WILL TAG FOR THIS STRING,
     ( TAGS(I) = 0 ),
=> 80 AND PUT NECESSARY POINTS
     INTO LINE().

TAGS(I) = 0
GO TO 80

******************************************************************************

GET TO 60 WHEN X = "X2".

- CALCULATE A YLAST & A YNEXT FOR CASES

  IF X = "X1", YLAST = "Y1"
     & YNEXT = "Y2",

  IF X > "X1", YLAST = "Y2" - SLOPE/2
     & YNEXT = "Y2",

  IN ANY EVENT ADJUST YMIN & YMAX AS
  REQUIRED FOR THESE VALUES OF
  YLAST AND YNEXT.

60 IF ( X GT LIST(I*X1) ) GO TO 61
   YLAST = LIST ( IY1 )
   GO TO 62
SUBP ROUTINE PRINT    --- MARCH 1, 1975

C 61  YLAST = LIST(IY2) - ( LIST(ISL)/(2+BIAS) )
C 62  IF ( YLAST.LT.YMIN ) YMIN = YLAST
    IF ( YLAST.GT.YMAX ) YMAX = YLAST
C 64  YNEXT = LIST(IY2)
    IF ( YNEXT.LT.YMIN ) YMIN = YNEXT
    IF ( YNEXT.GT.YMAX ) YMAX = YNEXT

C C C SKIP LINE SEGMENT BY ADJUSTING POINTERS
C ISL, IX1, & IX2
C
C BUT BEFORE COMPLETELY SKIPPING SEGMENT
C CHECK TO SEE IF THERE IS AN (X2, Y2)
C (IX2 < LAST).

C 78  ISL = IX1
IX1 = IX2
IX2 = IX1 + 2
    IF ( IX2.LT.LAST ) GO TO 68
C C C AT END-OF-STRING
C
C -- KILL TAG FOR STRING (TAGS(I)=0)
C
C -- => 88 AND INSERT NECESSARY POINTS:
C
C    TAGS(I) = 0
GO TO 88

C C C COMPLETE LINE SEGMENT SKIP BY ADJUSTING
C TAG: LIST(IY1), IY1, IY2
C
C CHECK IF INFINITE SLOPE (X1=X2)?
C IF SO => 64
C IF NOT => => 35 AND CALCULATE NEW SLOPE.
C C 68  TAGS(I) = IX1
    LIST(IY1) = LAST
IY1 = IX1 + 1
IY2 = IX2 + 1
SUBROUTINE PRINT

IF (X.EQ.LIST(IX2)) GO TO 64
GO TO 35

GET HERE UNDER NORMAL CONDITIONS

THAT IS - "XI" =< X <= "X2".

INTERPOLATE TO GET VALUE OF Y IN RANGE [ "Y1", "Y2" ]
& THEN BY GOING BACK & FORWARD .5*SLOPE GETTING VALUES FOR YLAST & YNEXT.
IF X="XI" THEN YLAST="Y1".

70  YLAST = LIST ( ISL ) * ( X- LIST(IXI) )
     1 - LIST ( ISL ) / 2
     YLAST = ( YLAST / BIAS ) + LIST ( IY1 )
     YNEXT = YLAST + ( LIST ( ISL ) / BIAS )
     IF ( X EQ LIST(IXI) ) YLAST = LIST ( IY1 )

NEXT ADJUST YMIN & YMAX AS NECESSARY TO ACCOMODATE YLAST & YNEXT SO THAT
-- YMIN <= ALL PTS IN LINE() <= YMAX

75  IF ( YLAST.LT.YMIN ) YMIN = YLAST
    IF ( YLAST.GT.YMAX ) YMAX = YLAST
    IF ( YNEXT.LT.YMIN ) YMIN = YNEXT
    IF ( YNEXT.GT.YMAX ) YMAX = YNEXT

CHECK IF ANYTHING TO PUT INTO LINE()
THAT IS -- 0 <= YMIN =< YMAX <= MAX.
IF SO -- PUT THEM INTO LINE()

80  IF ( YMIN.GE.MAX OR YMAX.LE.0 ) GO TO 20
SUBROUTINE PRINT —— MARCH, 1: 1975

IF ( YMIN .LE. 0 )  YMIN = 1
IF ( YMAX .GE. MAX )  YMAX = MAX - 1
C
DO 20 Y = YMIN, YMAX
20 CALL INSERT ( LINE, Y )
C
C
C  NEXT STRING  (= '20')
C
C
C  GO TO 20
C
C
C
C  END OF "PRINT"
C
END
END$
SUBROUTINE ORDER

SUBROUTINE ORDER ( LIST, TAGS, ITAG)

SUBROUTINE "ORDER"

ADAPTED AT MCMASTERR UNIVERSITY, FEBRUARY 1975

BY C. A. BRYCE, FOR A MSC. PROJECT

THESE SUBROUTINES HAVE BEEN OBTAINED
TO PROVIDE AN EFFICIENT SORTING PROCESS.
THE SORT IN ADDITION HAD TO BE A
TAG-SORT. THIS WAS NOT ORIGINALLY A
TAG-SORT, BUT WAS ADAPTED INTO ONE.

IT WAS FOUND IN A 4TH YEAR PROJECT AT
THE APPLIED MATH DEPARTMENT AT
MCMASTERR UNIVERSITY. THE PROJECT WAS
TITLED --

"INVESTIGATIONS OF AVAILABLE SORTING
ALGORITHMS AND THEIR BEHAVIOUR FOR
DIFFERENT ORDERINGS", AND WAS

AUTHORED BY --

MILKHA SINGH, AND BETTY M. PYDE
(1969-70)

THIS SORT WSS CHOSEN BECAUSE IT RATED
HIGHLY (IN THE TOP 2) AND BECAUSE
ONLY 4 CHANGES WERE NEEDED TO MAKE
IT INTO A TAG-SORT.

IN THIS DOCUMENTATION NO ATTEMPT WILL
BE MADE TO EXPLAIN HOW THE SORT WORKS
OTHER THAN TO SHOW WHERE THE NEEDED
CHANGES WERE MADE.

SUBROUTINE ORDER IS CALLED FROM "PRINT"
IN ORDER TO SORT THE TAGS IN
TAGS() TO THE STRINGS IN
LIST( ), EACH TAG POINTS TO
TOP OF A STRING, AND TAGS( )
IS REARRANGED SO THAT INDEXING
THROUGH TAGS( ) WILL ALSO BE
GOING THROUGH ASCENDING VALUES
FOR THE TOP OF EACH STRING

KEY VARIABLES

LIST( ) — INTEGER ARRAY CONTAINING STRINGS
OF RECORDS TO BE PLOTTED

TAGS( ) — INTEGER ARRAY CONTAINING TAGS OR
POINTERS TO START OF STRINGS IN
"LIST( )"

ITAG — NUMBER OF STRINGS IN "LIST( )"

INTEGER LIST, TAGS, ITAG
DIMENSION TAGS ( 0 ), LIST ( 0 )

LOCAL VARIABLES

I, J, K, L, M — INTEGER VARIABLES USED
IN SORTING PROCESS

II, IL — INTEGER VARIABLES ADDED ( CAB )
REPLACING I, & L IN PART
OF SORT

INTEGER I, J, K, L, M
INTEGER II, IL

M <= # OF ELEMENTS TO BE SORTED
M = ITAG
DIVIDE INTO 2 SUBSETS
WHEN M = 0 THE PROCESS IS COMPLETE

M = M / 2
IF ( M .EQ. 0 ) RETURN
SET K TO # ELEMENTS - M

K = ITHG - M

THE FOLLOWING " DO 3 " DO-LOOP
ACCOMPLISHES MIPACLES AND DOES
MOST OF THE WORK.

DO 3 J = 1, K
   I = J
   L = I + M

THE FOLLOWING 2 LINES SETTING
II, & IL WERE ADDED TO GET
NEEDED INDICES IN "LIST()"

II = TAGS ( I )
IL = TAGS ( L )

COMPARE LIST(II) : LIST(IL)

IF ( LIST(II).LE.LIST(IL) ) GO TO 3

THE FOLLOWING 2 LINES WERE ADDED
NOTICE THAT THE TAGS ARE
INTERCHANGED INSTEAD OF THE
ARRAY ELEMENTS ('LIST()')

TAGS ( L ) = II
TAGS ( I ) = IL
I = 'I - M
IF ( I GT 0 ) GO TO 2
CONTINUE

3

REDIVIDE WORK TO BE DONE

GO TO 1
END
END$
PAGE 0001

0001

INSRT R 000002
ENTR X 000001
LINE R 000000
POINT P 000001
MHSP P 000021
TBL R 000022
H 000000
B 000001
++ NO EProps
SUBROUTINE "INSRT"

WITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975

E. C. A. BRYCE, FOR A MSC PROJECT

"INSRT" IS CALLED FROM SUBROUTINE "PRINT"

AND IS DESIGNED TO DO A

LOGICAL INCLUSIVE OR OF A BIT

AT THE "POINT" TH BIT POSITION

IN THE ARRAY "LINE" ALL CALLS

FROM "PRINT" ARE SET UP SO THAT

"POINT" IS IN THE PROPER RANGE.

AN ASSEMBLER ROUTINE WAS USED PATER THAN

A FORTRAN ROUTINE AS IT WAS SMALLER.

< 42B COMPARED TO 103B FOR FORTRAN >

AND SUBSTANTIALY FASTER AS THIS IS

USED FREQUENTLY, THIS SPEED FACTOR BECAME

THE DECIDING FACTOR.

KEY VARIABLES

LINE() - INTEGER ARRAY CONTAINING PLOTTING

INFORMATION.

POINT - THE BIT POSITION IN "LINE" WHERE

BIT IS TO BE ADDED.

TBL() - LOOKUP TABLE CONTAINING THE

DIFFERENT REQUIRED BITS.

MASK - 4-BIT LOWER END MASK.

A & B - EQUIVALENCES TO A & B REGISTERS.
FORTAN EQUIVALENT ( TESTED FEB/13/75 )

SUBROUTINE INSRT ( LINE, POINT )

INTEGER LINE, POINT, TBL
DIMENSION LINE ( 0 ), TBL ( 16 )

DATA TBL / 10000B, 40000B, 20000B, 10000B, 4000B, 200B, 10B, 4B, 3B, 1B /

I = ( POINT + 15 ) / 16
J = POINT - 16 * ( I - 1 )

ON HP2100A "IOR" IS THE INCLUSIVE-OR FUNCTION

LINE ( I ) = IOR ( LINE(I), TBL(J) )

RETURN
END

0075*
0077*  DECLARE "INSRT" TO BE THE ENTRY-POINT
0078*  AND ".ENTR" TO BE AN EXTERNAL.
0079*  
0080+  ENT INSRT
0081+  ENT .ENTR
0082+  
0083+  
0084*  
0085*  THE ADDRESSES FOR "LINE" AND FOR "POINT"
0086*  WILL BE STORED HERE BY ".ENTR".
0087+  
0088+  00000 00000 LINE BSS 1
0089+  00001 00000 POINT BSS 1*
0090*  
0091*  
0092*  
0093*  WE SET UP THE ENTRY/EXIT: LINE - INSRT
0094*  WE START HERE
0095*  00002 00000 INSRT BSS 1
0096*  0100*
0097+  CALL ".ENTR" TO SET UP FORTRAN LINKAGE.
0098+  
0099+  00003 016001X  JSB .ENTR
0100+  00004 00000R  DEF LINE
0101*  
0102*  GET THE VALUE OF "POINT" - 1
0103+  AND PUT IT IN BOTH A & B
0104*  
0105*  
0106+  00005 003400  CCA  A <= -1
0107+  00006 142001R  ADA POINT I  A <= POINT-1
0108+  00007 064000  LDB A  B <= POINT-1
0116*  TAKE THE COPY IN B AND RIGHT SHIFT IT 4 BITs
0117*  THUS EFFECTIVELY DIVIDING BY 16. AND
0118*  B' ADDING THE ADDRESS OF LINE(1) OBTAINING
0119*  THE ADDRESS OF THE PROPER ELEMENT OF LINE()
0120*
0121*
0122* 00010 005121  BRS, BRS
0123 00011 005121  BPS, BPS  RIGHT SHIFT M 4 PLACES
0124 00012 046000R  .ADD LINE - B <= ADDRESS IN ARRAY LINE
0125*
0126*  MASK OFF LAST 4 BITS OF M AND USE
0127*  THE RESULT AS THE INDEX FOR LOOKUP
0128*  TABLE "TBL"
0129*
0130*  THEN USING THE ADDRESS IN B, OR IN
0131*  VALUE FROM LINE() AND THEN REPLACE
0132*  IT BACK INTO LINE().
0133*
0134*
0135* 00013 012021R  AND MASK' JUST LAST 4 BITS OF A
0136 00014 042022R  ADD TBL  GET ADDRESS IN "TBL"
0137 00015 160000  LDA A, I  A <= ENTRY FROM "TBL"
0138*
0139* 00016 130001  IOR B, I  A <= IOR < A-REG, LINE()
0140 00017 170001  STA B, I  LINE() <= RESULT IN A
0141*
0142*
0143*  RETURN
0144*
0145* 00020 126002R  JMP INSR, I
**DATA NEEDED**

<table>
<thead>
<tr>
<th>Hex</th>
<th>Oct</th>
</tr>
</thead>
<tbody>
<tr>
<td>0021</td>
<td>000017</td>
</tr>
<tr>
<td>0022</td>
<td>00023P</td>
</tr>
</tbody>
</table>

**TBL IS THE ADDRESS OF THE NEXT WORD**

<table>
<thead>
<tr>
<th>Hex</th>
<th>Oct</th>
</tr>
</thead>
<tbody>
<tr>
<td>0023</td>
<td>10000</td>
</tr>
<tr>
<td>0024</td>
<td>04000</td>
</tr>
<tr>
<td>0025</td>
<td>02000</td>
</tr>
<tr>
<td>0026</td>
<td>01000</td>
</tr>
<tr>
<td>0027</td>
<td>00400</td>
</tr>
<tr>
<td>0030</td>
<td>00200</td>
</tr>
<tr>
<td>0031</td>
<td>00100</td>
</tr>
<tr>
<td>0032</td>
<td>00040</td>
</tr>
<tr>
<td>0033</td>
<td>00020</td>
</tr>
<tr>
<td>0034</td>
<td>00010</td>
</tr>
<tr>
<td>0035</td>
<td>00040</td>
</tr>
<tr>
<td>0036</td>
<td>00020</td>
</tr>
<tr>
<td>0037</td>
<td>00000</td>
</tr>
<tr>
<td>0040</td>
<td>00004</td>
</tr>
<tr>
<td>0041</td>
<td>00002</td>
</tr>
<tr>
<td>0042</td>
<td>00001</td>
</tr>
</tbody>
</table>

**NO ERRORS***
<table>
<thead>
<tr>
<th>INSRT</th>
<th>CROSS-REFERENCE SYMBOL TABLE</th>
<th>PAGE 0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTR</td>
<td>00082 00103</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>00177 00114 00138</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>00178 00140 00141</td>
<td></td>
</tr>
<tr>
<td>INSFT</td>
<td>00099 00091 00147</td>
<td></td>
</tr>
<tr>
<td>LINE</td>
<td>00089 00104 00125</td>
<td></td>
</tr>
<tr>
<td>MASK</td>
<td>00153 00136</td>
<td></td>
</tr>
<tr>
<td>POINT</td>
<td>00090 00113</td>
<td></td>
</tr>
<tr>
<td>TBL</td>
<td>00154 00137</td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE Bfout ( LINE )

C
C SUBROUTINE "BFOUT"
C
C WRITTEN AT McMaster UNIVERSITY, FEBRUARY 1975
C BY C. A. BRYCE, FOR A MSC. PROJECT.
C
C "BFOUT" IS CALLED FROM SUBROUTINE "PRINT", AND IT'S PURPOSE IS TO OUTPUT
C THE CONTENTS OF "LINE" ONTO THE PRINTER/PLOTTER.
C
METHOD
C IN PUTTING OUT THE CONTENTS OF "LINE()" TO THE PRINTER/PLOTTER, DOUBLE BUFFERING IS USED
C IN ORDER TO TAKE ADVANTAGE OF THE CAPABILITY OF WRITING WITHOUT WAITING. "LINE()" IS
C COPIED TO "BUFF()", AND SIMULTANEOUSLY "LINE()" IS BEING CLEARED ( <= 0 ). WITH THE
C INFORMATION NOW IN "BUFF()", A BINARY WRITE WITHOUT WAIT IS INITIATED ON "BUFF()".
C THEN CONTROL IS RETURNED BACK TO "PRINT" TO ALLOW "PRINT" TO PROCEED WITH CALCULATIONS
C FOR THE NEXT LINE.
C
ALTHOUGH THROUGH TESTING IT WAS FOUND THAT ON THE HP, THIS DOUBLE BUFFERING WASN'T
C ABSOLUTELY NECESSARY, IT WAS INCLUDED IN ORDER TO PROTECT AGAINST POSSIBLE PROBLEMS
C WITH DIFFERENT MACHINES. IN THIS TESTING, IT WAS ALSO FOUND THAT A STATUS CALL WASN'T
C NECESSARY IN THE TYPE OF WRITING DONE IN "BFOUT", WITH BUFFERS LARGER THAN 35 WORDS
C STATUS CHECKING WOULD HAVE BEEN NECESSARY.
C
KEY VARIABLES
C LINE () = INTEGER ARRAY CONTAINING PLOTTING INFORMATION.
SUBROUTINE Bfout --- MARCH 1, 1975.

C
C BUFF(*) - INTEGER ARRAY USED FOR COPYING
INFORMATION FROM "LINE".

J - INDEX USED WITH "LINE(*)" & "BUFF(*)"

INTEGER LINE, BUFF, J
DIMENSION LINE ( 3S ), BUFF ( 3S )

C FIRST - COPY LINE(*) = > BUFF()
C & SIMULTANEOUSLY 0 = > LINE()

DO 100 J = 1, 35
   BUFF ( J ) = LINE ( J )
100 LINE ( J ) = 0

C NEXT - DO AN "EXEC" CALL TO DO A
C WRITE ( AN RCODE OF 2 ) TO
C
   LOGICAL UNIT 8 ( CHWD = 108 )
   IN BINARY MODE ( CHWD = CHWD + 100B )
   AND DON'T WAIT ( CHWD = CHWD + 20000B )

   FROM BUFFER ( NAMED "BUFF" )
   AND THE BUFFER LENGTH ( 3S ).

   CALL EXEC ( 2, 201100, BUFF, 35 )

   THE PRINTER HAS BEEN STARTED, BUT
   RETURN TO "PRINT" AND SAVE THE NORMAL
   WAITING PERIOD.

RETURN
END
PAGE 0001

0001

ASMB.R.L.T.C

PUT R 000007
LIST R 000315
CLEAR R 000332
CLSEF P 000331
IVPTA P 000101
SWAPI P 000654
ENTR X 000001
ERR0 X 000002
EXEC X 000003
INVRT X 000004
PAUS X 000005
VAL R 000000
IND P 000001
I R 000002
FLAG R 000003
TAGS R 000004
ITAG R 000005
MANT R 000006
RANGE R 000021
ERRP R 000835
OVFLW R 000043
O 2 R 000050
BASI R 000110
BSAB R 000111
BUFF R 000114
IL R 000314
LOOP R 000311
CHT P 000363
ADD R 000364
SCTR R 000365
SCTRL R 000565
SCTR1 R 000566
EMPTY R 000567
SCTR0 R 000601
ELP R 000607
GTRK 'R 000617
PI28 'R 000627
SECT R 000630
FOUR R 000645
M201 R 000646
TRK R 000647
TRACK R 000650
TGS 'R 000651
ITG R 000652
"DSKIO" -- DISK I/O PACKAGE (VERSION II)

WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975.

B. C. A. BRYCE, FOR A. MSC. PROJECT

"DSKIO" IS A SET OF 5 SUBROUTINES & 1 FUNCTION THAT PROVIDES DISK STORAGE TO PLOT INSTEAD OF A PPA\^{}Y STORAGE. ALSO, THE BEGINNING OF THE SWAPPING "SWAP1" IS INITIATED IN THIS PACKAGE.
THE FOLLOWING Routines MAKE UP "DSKIO":

PUT - THIS routine ALLOWS PLOT TO SIMULATE AN ASSIGNMENT STATEMENT OF THE FORM -- LIST(i) =

LIST - THIS FUNCTION RETURNS THE VALUE IN THE RECORD CONSTRUCTED IN PUT & SIMulates AN ACCESS OF THE PSEUDO-ARRAY [ LIST ] = LIST(i)

CLEAR - THIS routine TRANsFERS THE CURRENT STRING FROM THE RECORD ( OR STRING ) INTO A SECTOR BUFFER THE SECTOR BUFFER IS AUTOMATICALLY FLUSHED WHEN IT BECOMES FULL

CLSEF - THIS routine ALLOWS THE FLUSHING OF THE LAST SECTOR BUFFER TO DISK, AND THE RE-INITIALIZATION OF SOME VARIABLES USED IN "DSKIO" SO THAT WHEN WE RETURN WE CAN START OVER

IVRTA - THIS routine SIMPLY CALLS "INVRT" WHICH CONVERTS AN UP-STRING INTO A DOWN-STRING THE KEY POINT IS THAT IT DOES IT WITH THE STRING RECORD (BUFF)

SWAP1 - THIS routine BEGINS THE SWAPPING PACKAGE HI-CORE IS SAVED ON DISK AND THEN "SWAP2" IS READ INTO THIS SAME CORE, AND A CALL IS MADE OF SWAP2 WHICH COMPLET ES THE SWAPPING PROCESS

0056*
0057*
ENTRY POINTS

ENT PUT
ENT LIST
ENT CLEAR
ENT CLOSE
ENT INPUT
ENT SWAP

EXTERNALS USED

EXT ENTR
EXT ERROR
EXT EXEC
EXT INVPT
EXT PHUS
SUBROUTINE PUT -- IN FORTRAN -- CALL PUT VAL, I.
PUT -- INSERTS VAL INTO INDEX I IN PSEUDO-
ARRAY LIST. THIS SIMULATES AN ASSIGNMENT
STATEMENT ON LIST. LIST. IND. = VAL.
PUT REQUIREs SEVERAL ARGUMENTS SIMPLY BECAUSE
THE DIFFERENT VALUES OF I, FHG, THG, ITAG,
AND HLT ARE NEEDED WHEN THE RECORD-
BUFFER OVERFLOWS AND A STRING IS BROKEN
INTO TWO SUB-STRINGS:

VAL BSS 1
IND BSS 1
I BSS 1
FHG BSS 1
THG BSS 1
ITAG BSS 1
HLT BSS 1
PUT HOP
CLEAR UP THE LINKAGE
JSB ENTR
DEF VAL

CHECK THE RANGE OF THE INDEX
ON RETURN FROM RANGE---
B -TIVE --- NO PROBLEM
B +TIVE --- OVERFLOW
JSB RANGE
SSB, RSS
0126+ IF OVERFLOW GO TO "OVFLW"
0129+ JMP OVFLW
0130 00014 026043P
0132+ IF IND IN RANGE
0135+ H = ADDRESS IN THE BUFFER
0137+ E = VALUE
0139+ VALUE PUT INTO BUFFER
0141+ 0143 00015 042111P
0144 00016 166080P
0145 00017 174600
0147+ PETUPN
0149+ 0150 00020 126007P
0151+
0154+  
0155+  
0156+  RANGE. - CHECKS TO SEE IF "IND" IS IN  
0157+  RANGE BY COMPARING IT TO THE  
0158+  BASE INDEX.  
0159+  
0160+  
0161 0021 000040  PHNGE NOP  
0162+  
0163+  
0164+  
0165+  M = IND + 2 - BWSI  
0166+  
0167+  
0168 0022 062110P  LDW BWSI  
0169 0023 003004  CMW, INH  
0170 0024 002004  INH  
0171 0025 002004  INH  
0172 0026 142001P  NINH IND. I  
0173+  
0174+  
0175+  IF M-PEG . 0  
0176+  THINGS HPE OR  
0177+  
0178+  IF NOT REPORT ERROR (ERPP)  
0179+  
0180+  
0181 0027 002002  SZA  
0182 0030 002020  SSA  
0183 0031 02035P  JMP ERRP  
0184+  
0185+  
0186+  
0187+  
0188+  FOR EXITING WE SET B TO  
0189+  
0190+  INDEX IN RECORD - 2018  
0191+  
0192 0032 066646R  LDB M201  
0193 0033 044000  ADD A  
0194+  
0195+  
0196+  RETURN  
0197+  
0198 0034 126021R  JMP RANGE, I
PAGE 0009 #01 - RANGE ERRP OYFLW

0200*
0201*  ERRP  -  AT THIS POINT WE REPORT ERROR THRU
0202*  "ERR0"  ---  "RH GE"
0203*
0204*  00035 0627HEP ERRP  LIN = RH
0205*  00036 066761P  LDB = HGE
0206*  00037 016002J  JSB ERR0
0207*  00040 016005%; JSB PHIS
0208*  00041 002400  CLE
0209*
0210*  RETUPN
0211*
0212*  00042 126007R.  JMP PUT.I
0213*
0214*
0215*
0216*
0217*
0218*
0219*
0220*  OYFLW  -  THIS HANDLES THE CASE WHEN SOMEONE
0221*  TRIES TO PUT SOMETHING INTO AN INDEX
0222*  THAT IS NOT IN THE RECORD. IN THIS CASE
0223*  WE PUT OUR CURRENT RECORD AND START A
0224*  NEW STPING, AND PESTMRT THE STRING
0225*  STARTING AT LAST CHAR
0226*
0227*
0228*
0229*  1ST CHECK FLAG TO SEE IF WE MUST INVRT
0230*
0231*  00043 162063R OYFLW 'LDA FLAG.I
0232*  00044 002003  SZA; PSS
0233*
0234*  NO NEED TO INVRT STRING
0235*
0236*  00045 026050R  JMP 0.3
0237*
0238*  0239*  CALL INVRT INDIRECTLY THRU "IVRTA"
0240*
0241*  00046 016101R  JSB IVRTA
0242*  00047 000050R  DEF **1
0243* CLEAR OUT RECORD BUFFER
0244* GET NEXT WORDPSS OF 'ITAG' & CHECK IF
0245* 'ITAG' = 'MINT'
0246* GET 'ITAG' IF OK
0247* JSB CLEWP
0248* DEF ++1
0249* LDH ITHG. I
0250* ISS ITHG. I
0251* LDB MINT. I
0252* CMN. INP
0253* MDD M
0254* SSB PSS
0255* ITAG IS TOO BIG
0256* MAINT
0257* REPORT OVERFLOW
0258* JMP EPPP
0259* SET TAGS(ITAG) = I
0260* Compliance
0261* ADA TAGS
0262* LDB BASH
0263* STB I. I
0264* INB
0265* STB A. I
0266* How restart string by moving last two in buffer
0267* To the start of the buffer
0268* BUT 1ST SET -- LIST(I)
0269* INB
0288  0067  006004     INB
0289  00670  075113R    STB BUFF-1
0291  00071  062310P    LDA BUFF+124
0292  00072  072114P    STH BUFF
0293  00073  062311P    LDH BUFF+125
0294  00074  072115P    STH BUFF+1

0296*
0297*
0298*  'IND' = 'IND' + 4
0299*
0300  00075  122001P    LDH IND.1
0301  00076  042645P    HDH FOUR
0302  00077  172001P    STH IND.1

0304*
0305*    TRY AGAIN
0306*
0307*  00100  026012R    JMP PUT+3
IVRTA - MAKES A CALL TO "INVPT" BUT USES A LOCAL BUFFER 'BUFF' TO CALL WITH

0314 00101 000000  IVPTH  NOP
0315 00102 036101P  152 IVPTH

0317 00103 016004  JSB INVRT
0318 00104 000107P  DEF ++2
0319 00105 000113P  DEF BUFF-1
0320 00106 000110P  DEF BASI

PETUPH

0325 '00107 126101P  JMP IVRTA.I
0326 *

STORAGE USED FOR RECORD STRUCTURE USED

IN 'PUT' & 'LIST'

0335 00110 000002  BASI OCT 2
0336 00111 000111R BASAD DEF *
0337 00112 000000  OCT 0
0338 00113 000000  OCT 0
0339 00114 000000  BUFF BSS 128
FUNCTION LIST -- THIS ROUTINE SIMULATES AN
ACCESS OF THE FOPM

I := LIST(I)

LIST RETURNS VALUE OF "I"TH ELEMENT OF ARRAY

0342+
0343+
0344+ FUNCTION LIST -- THIS ROUTINE SIMULATES AN
0345+ HARR'S ACCESS OF THE FOPM
0346+
0347+ I := LIST(I)
0348+
0349+ LIST RETURNS VALUE OF "I"TH ELEMENT OF ARRAY
0350+

0352 00314 000000 IL ESS I
0353 00315 000000 LIST HOP

0354 00316 016001X JSB ENTP
0355 00317 000314R DEF IL

0356+ SET UP COMMON EXIT IN 'PUT'
0360+
0361 00320 062315P LDH LIST
0362 00321 072007F STM PUT

0363+ SET INDEX TO IL
0366+
0367 00322 162314R LDH IL,I
0368 00323 172001P STA IND,I

0370+ CHECK THE RANGE OF INDEX
0372+
0373 00324 016021R JSB RANGE
0375 00325 006021 SSB,RSS

0377+ OUT-OF-RANGE
0379+
0380 00326 026035R JMP ERRP
FUNCTION LIST\{I\}\n
A = VALUE OF LIST\{I\}

ADW BASAD
LDW H+1

RETUPH

JMP PUT: I
SUBROUTINE CLEAR -- TRANSFER STRING TO SECTOR

0334+ 0395+ SUBROUTINE CLEAR -- TRANSFERS STRING FROM BUFF
0396+ INTO SECTOR BUFFER "SCF"
0397+ 0398+

0400 00372 000000 CLEWP NOP
0401 00373 025372P ISP CLEWP
0402+
0403+
0404+
0405+ CALCULATE WORD COUNT IN STRING-BUFFER "BUFF"
0406+
0407 00334 062113P LDA BUFF-1
0408 00335 070001 STA E
0409 00336 006004 INB
0410 00337 003000 CMH
0411 00340 043110P HDH BSHI
0412+
0413+
0414+ IF A \neq 0 SOMETHING TO GO INTO SECTOR
0415+
0416 00341 002021 SSA, PSS
0417+
0418+ QUICK RETURN
0419+
0420 00342 126332P JMP CLEAR, I
0421+
0422+ UPDATE NEW BASE INDEX
0423+
0424+ SAVE -TIVE WORD COUNT
0425+
0426 00343 076110R STB BASI
0427 00344 066111R LDB BASAD
0428 00345 006004 INB
0429 00346 072363R STA CNT
0430 00347 076364R STB ADD

0432+
0433+ B \leq CURRENT SECTOR ADDRESS
0434+
0435 00350 066566R LDB SCTRI
0437:  TRANSFER RECORD OVER TO SECTOR
0439:  BUFFER CHECKING EACH TIME FOR
0441:  A FULL SECTOR / E = SCTRL
0442:  0444 00351 056545P LOOP CPE SCTRL
0445 00352 015667P ISB EMPTY
0446 00353 162364P LDH HIT "I"
0447 00354 1700001 STH B "I"
0449 00356 006004 INC
0450 00358 036764P ISE nid
0452 00357 036363P ISF CNT
0454 00360 026351P JMP LOOP
0456:
0457:
0458:  SAVE CURRENT SECTOR ADDRESS
0459:
0460 00361 075666P STB SCTRL
0462:
0463:  RETURN
0464:
0465 00362 126332R JMP CLEAR, I
0467:
0468:  DISC SECTOR STORAGE
0469:
0470 00363 000000 CNT NOP
0471 00364 000000 ADD NOP
0472 00365 000000 SCTR BSS 128
0473 00565 000565R SCTRL DEF *
0474 00566 000365P SCTRL DEF SCTR
0477+ WRITE 'SCTP' TO DISK -- CHECKING FOR 
0478+ END OF TRACI 
0479+ .00527 000000 EMPT NP 
0480+ .00570 006630P JSE SECT 
0481+ .00571 006670P LDA SECT 
0482+ .00572 040511 MDR MSECTION 
0483+ REED ANOTHER TRACK 
0484+ .00573 002021 SRP RPO 
0485+ REQUEST ANOTHER M又 APEX TRACK 
0486+ .00574 016617P JSB GTPI 
0487+ WRITE OUT 'SCTP' 
0488+ .00575 016003P JSB EXEC 
0489+ .00576 000605P DEF +7 
048a+ .00577 000055 DEF +2 
048b+ .00578 000055 DEF +2 
048c+ .00581 00365P.SCTRO DEF SCTR 
048d+ .00582 000687R DEF P128 
048e+ .00583 000647R DEF TPF 
048f+ .00584 000630P DEF SECT 
0490+ 
0491+ .00585 016003: JSB EXEC 
0492+ .00586 000605P DEF +7 
0493+ .00587 000055 DEF +2 
0494+ .00588 000055 DEF +2 
0495+ .00591 00365P.SCTRO DEF SCTR 
0496+ .00592 000687R DEF P128 
0497+ .00593 000647R DEF TPF 
0498+ .00594 000630P DEF SECT 
0499+ 
0500+ SET 'SECT()' TO 0 
0501+ .00595 002400 CLA 
0502+ .00606 066601R LDB SCTRO 
0503+ .00607 170001 E.LP STA B,I 
0504+ .00610 006004 INB 
0505+ .00611 056565R CPB SCTRL 
0506+ .00612 002001 RSS 
0507+ .00613 026607R JMP E.LP
PAGE 0018 #01  "EMPTY" -- FLUSH OUT 'SCTR' SCTR<0> = 0

0521* SET 'SCTR' TO START OF 'SCTR' (SCTR0)
0522* 0524 00614 066601P LDB SCTR
0525 00615 05552EP STB SCTR
0526* 0527* RETURN
0528* 0529 00616 126657P JMP EMPTY I.

0531* REQUEST A NEW TRK FROM
0532* 0534* THE MOPH/PREA
0535* 0536 00617 000000 GTPH NOP
0537* 0538* THIS CALLS GTRPH WHICH DOES
0539* 0540* THE MOPH
0541* 0542* GTRPH HAS TO TIDY UP SECT & TRK
0543* 0544 00620 016717R JSB GTRPH
0545 00621 002400 CLH
0546 00622 072630P STH SECT
0547 00623 06664TR LDB TPH
0548* 0549* IF 1ST CALL SET TRK TO TRK
0550* 0551 00624 052650P CPA TRK
0552 00625 076650P STB TRK
0553* 0554* RETURN
0555* 0556 00626 126617R JMP GTRPH I

0558* CONSTANTS FOR DISK ACCESS
0559* 0560* 0561 00627 000200 P128 OCT 200
0562 00630 000027 SECT DEC 23
0565+ SUBROUTINE CLESEF -- 'CLOSE' THE WORKFILE
0566+ TRANSFER LAST STRING TO COLTP
0567+ 'WRITE WHAT IS STILL IN COLTP'
0568+ 'UP THE LAST PORTION OF THE'
0569+ 'BUFFER COLTP TO THE DISK'

0574 00531 000000 CLSEF NOP
0575 00522 076677P ISZ CLSEF

0577 00533 016377P JSB CLESEF
0578 00534 000678P BEF ++1
0579 00535 016567P JSB EMPTY

0581+ INITIALIZE 'SECT' & 'BUFF-1' TO 0
0582+ BSI TO 2
0583+ Cla
0584 00636 002400 CLA
0585 00637 072630P STH SECT
0586 00640 072113P STH BUFF-1
0587 00641 002004 INH
0588 00642 002004 INH
0589 00643 072110P STH BASI

0593+ RETURN
0594+ JMP CLSEF, 1

0596 00644 126631R

0598+ CONSTANTS
0599+ 0600+
0601 00645 000004 FOUR OCT 4
0602 00646 177577 M201 OCT -201
PAGE 0020 #01 << SUBROUTINE SWAP1 -- SWAP HI-CORE FOR "SWAP2" >>

0605*  , SUBROUTINE SWAP1 -- SWAP HI-CORE FOR "SWAP2"
0606*        CALL "SWAP2"
0607*  ,
0608*  ,
0609*  0610  00647 000000  TRI  BSS 1
0611  00650 000000  TRCH  OCT 0
0612  00651 000000  TGS  BSS 1
0613  00652 000000  ITG  BSS 1
0614  00653 000000  IMAX  BSS 1

0615  ,
0616  00654 000000  SWAP1 HOP
0617  ,
0618  00655 016001X  JSB  ENTR
0619  00656 000651R  DEF TGS

0620*  ,
0621*  , CALCULATE CORE SIZE ABOVE 'PP'
0622*  ,
0623*  , FROM PARTITION POINT (PP) & LWAM
0624*  ,
0625*  , (LAST-WORD-AVAILABLE (USER)-MEMORY)
0626*  ,
0627*  ,
0628  00657 062757R  LDA PP
0629  00660 003004  CNA INH
0630  00661 040100  ADA LWAM
0631  00662 072753R  STA BFSZ

0632*  ,
0633*  , CONFIGURE 'RWHi' ROUTINE FOR A WRITE
0634*  , RWRCI = 2.
0635*  ,
0636*  ,
0637*  , GET A TRACK ON DISK
0638*  ,
0639  00663 036750R  ISZ RWRCI
0640  00664 016717R  JSB GTTRK
PAGE 0021 #01  -- SUBROUTINE SWAP1 -- SWAP HI-CORE FOR "SWAP2" --

0642*  MPITE HI-CORE TO WORK AREA
0643*  JSB FWHI

0644*  PEAD OVERLAY "SWAP2" INTO HI-CORE
0645*  FROM BI-FILE CALLED "SWAP2"

0646*  JSB EXEC
0647*  DEF ++7
0648*  DEF P14
0649*  DEF +3
0650*  DEF SWAP2, I
0651*  DEF BF32
0652*  DEF FHIME
0653*  DEF PSECT

0654*  CALL SWAP2

0655*  JSB SWAP2, I
0656*  DEF TRY

0657*  UPON RETURN FROM "SWAP2"

0658*  - RESET HI-CORE (FROM WORK AREA OF DISK)

0659*  - DEALLOCATE WORK AREA ON DISK

0660*  CLA: INA
0661*  STA RU.PCD

0662*  JSB RUHI

0663*  JSB RESET

0664*  RETURN

0665*  JMP SWAP1, I

0666*  00704 126654R
PAGE 0022 #01 << SUBROUTINE SWAP1 -- SWAP HI-CORE FOR "SWAP2" >>

0688*
0689* PWHI - WRITES/READS HI-CORE TO/from WOPR WREA ON DISK
0690* DEPENDING ON PMPCD
0691*
0692* 0693*
0694 00705 000000 PWHI NOP
0695 00706 016003 JSB EXEC
0696 00707 000716P DEF +7
0697 00710 000750P DEF PMPCD
0698 00711 000055 DEF +2
0699 00712 100757P DEF SHHIF2.I
0700 00713 000753P DEF BFSZ
0701 00714 000647R DEF TP1
0702 00715 000751P DEF PSFCT
0703*
0704 00716 126705P JMP PWHI.I
0705*
0706* REQUEST X TRACK FROM WOPR WREA
0707*
0708 00717 000000 GTTRK NOP
0709 00720 016003 JSB EXEC
0710 00721 000727P DEF +6
0711 00722 000057 DEF +4
0712 00723 000730R DEF, TCONS
0713 00724 000647P DEF TRK
0714 00725 000731P DEF LUN
0715 00726 000732R DEF S-TRY
0716*
0717* RETURN
0718*
0719* 0720 00727 126717R JMP GTTRK.I
0721*
0722* CONSTANTS FOR REQUESTING TPACKS
0723*
0724 00730 100001 TCONS OGT 100001
0725 00731 000000 LUN NOP
0726 00732 000000 S-TRK NOP
PAGE 0023 #01 ← SUBROUTINE SWAP1 ← SWAP HI-CORE FOR "SWAP2" →

0730*     RESET ← SECT TO #(SECTORS)-1
          DEALLOCATE WORK-AREA
0731+ 0732  00733  000000  PESET NOP
0733  00734  050511  LDH MSECT
0734  00735  003000  CMH
0735  00736  072630R  3TH SECT
0736  00737  002400  CLH
0737  00738  072650R  3TH TRNCY
0738+ 0740  00741  SET (PTPK) TO 0
0741* 0742+ 0743  00741  066747R  LDB PTPK
0744  00742  016003%  JSB E:EC
0745  00743  000745P  DEF H+2
0746  00744  000746P  DEF M19
0747+ 0749*  RETURN
0750+ 0751  00745  126733P  JMP PESET, I
0752* 0753  00746  177785  M19  DEC -19
0754  00747  000267  RTRK OCT 26↑
PAGE 0024 #01 << SUBROUTINE SWAP1 -- SWAP HI-CORE FOR "SWAP2" >>

0756*
0757+ STOPAGE REQUIRED IN SWAP1
0758+
0759 00750 000001 PWRCD OCT 1
0760 00751 000000 RSECT OCT 0
0761 00752 000016 F14 DEC 14
0762 00753 000000 BFSZ HOP
0763 00754 051527 FHIME HSC 3 SWAP2
  00755 040520
  00756 031040
0764 00757 024000 SWAP2 OCT 24000
0765 00757 PP EOU SWAP2
0766*
0767*
0768* EQUIVALENCE, EUSED
0769*
0770+

0772 00000 A EOU 0
0773 00001 B EOU 1
0774 00053 EOU 53B
0775 00100 LWAM EOU 100B
0776 00511 MSECT EOU 511B
  00760 051116
  00761 043505
0777 END
+* NO ERRORS +*
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address 1</th>
<th>Address 2</th>
<th>Address 3</th>
<th>Address 4</th>
<th>Address 5</th>
<th>Address 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSKIO</td>
<td>00774</td>
<td>00503</td>
<td>00504</td>
<td>00655</td>
<td>00698</td>
<td>00711</td>
</tr>
<tr>
<td>ENTP</td>
<td>00076</td>
<td>00111</td>
<td>00355</td>
<td>00618</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAUS</td>
<td>00080</td>
<td>00310</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=AGE</td>
<td>00208</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=ARH</td>
<td>00207</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>00772</td>
<td>00145</td>
<td>00193</td>
<td>00258</td>
<td>00377</td>
<td>00386</td>
</tr>
<tr>
<td>ADD</td>
<td>00471</td>
<td>00430</td>
<td>00446</td>
<td>00450</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>00773</td>
<td>00408</td>
<td>00447</td>
<td>00516</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BASAD</td>
<td>00336</td>
<td>00143</td>
<td>00385</td>
<td>00427</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BASI</td>
<td>00335</td>
<td>00168</td>
<td>00273</td>
<td>00320</td>
<td>00411</td>
<td>00426</td>
</tr>
<tr>
<td>BFSZ</td>
<td>00762</td>
<td>00631</td>
<td>00657</td>
<td>00700</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUFF</td>
<td>00339</td>
<td>00289</td>
<td>00291</td>
<td>00292</td>
<td>00293</td>
<td>00294</td>
</tr>
<tr>
<td></td>
<td>00407</td>
<td>00588</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLEAR</td>
<td>00400</td>
<td>00066</td>
<td>00251</td>
<td>00401</td>
<td>00420</td>
<td>00465</td>
</tr>
<tr>
<td>CLSFF</td>
<td>00574</td>
<td>00067</td>
<td>00575</td>
<td>00596</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNT</td>
<td>00470</td>
<td>00422</td>
<td>00452</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E LP</td>
<td>00516</td>
<td>00520</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPTY</td>
<td>00483</td>
<td>00445</td>
<td>00529</td>
<td>00579</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERR0</td>
<td>00077</td>
<td>00209</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERRP</td>
<td>00207</td>
<td>00184</td>
<td>00265</td>
<td>00380</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXIT</td>
<td>00078</td>
<td>00501</td>
<td>00652</td>
<td>00695</td>
<td>00709</td>
<td>00744</td>
</tr>
<tr>
<td>FLAG</td>
<td>00101</td>
<td>00231</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNAME</td>
<td>00763</td>
<td>00658</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOUR</td>
<td>00601</td>
<td>00361</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTRK</td>
<td>00536</td>
<td>00496</td>
<td>00556</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTTRK</td>
<td>00708</td>
<td>00544</td>
<td>00640</td>
<td>00720</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>00100</td>
<td>00274</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Address 1</td>
<td>Address 2</td>
<td>Address 3</td>
<td>Address 4</td>
<td>Address 5</td>
<td>Address 6</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>QIMAX</td>
<td>00614</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IND</td>
<td>00099</td>
<td>00172</td>
<td>00300</td>
<td>00302</td>
<td>00368</td>
<td></td>
</tr>
<tr>
<td>INYRT</td>
<td>00079</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITAG</td>
<td>00103</td>
<td>00254</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QITG</td>
<td>00613</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IVFTA</td>
<td>00314</td>
<td>00068</td>
<td>00241</td>
<td>00315</td>
<td>00325</td>
<td></td>
</tr>
<tr>
<td>LIST</td>
<td>00353</td>
<td>00065</td>
<td>00351</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOOP</td>
<td>00444</td>
<td></td>
<td>00454</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUN</td>
<td>00725</td>
<td></td>
<td>00714</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LWAN</td>
<td>00775</td>
<td></td>
<td>00630</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M19</td>
<td>00753</td>
<td></td>
<td>00745</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M201</td>
<td>00602</td>
<td></td>
<td>00192</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAXT</td>
<td>00104</td>
<td></td>
<td>00256</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSECT</td>
<td>00776</td>
<td></td>
<td>00487</td>
<td>00733</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O.2</td>
<td>00251</td>
<td></td>
<td>00235</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVFLW</td>
<td>00231</td>
<td></td>
<td>00130</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P128</td>
<td>00561</td>
<td></td>
<td>00506</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PP</td>
<td>00765</td>
<td></td>
<td>00628</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td>00106</td>
<td>00064</td>
<td>00150</td>
<td>00216</td>
<td>00307</td>
<td>00362</td>
</tr>
<tr>
<td>R14</td>
<td>00761</td>
<td></td>
<td>00654</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RANGE</td>
<td>00161</td>
<td></td>
<td>00123</td>
<td>00198</td>
<td>00373</td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>00732</td>
<td></td>
<td>00681</td>
<td>00751</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSECT</td>
<td>00760</td>
<td></td>
<td>00659</td>
<td>00702</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTRK</td>
<td>00754</td>
<td></td>
<td>00743</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUHI</td>
<td>00694</td>
<td></td>
<td>00645</td>
<td>00679</td>
<td>00704</td>
<td></td>
</tr>
<tr>
<td>RURCD</td>
<td>00759</td>
<td></td>
<td>00639</td>
<td>00677</td>
<td>00697</td>
<td></td>
</tr>
<tr>
<td>STRK</td>
<td>00726</td>
<td></td>
<td>00715</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCCTR</td>
<td>00472</td>
<td>00474</td>
<td>00505</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSKIO</td>
<td>CROSS-REFERENCE SYMBOL TABLE</td>
<td>PAGE 0003</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------</td>
<td>-----------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCTP0</td>
<td>00505 00514 00524</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCTR1</td>
<td>00474 00435 00450 00525</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCTPL</td>
<td>00473 00444 00518</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECT</td>
<td>00562 00485 00485 00508 00546 00587 00735</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMHPI</td>
<td>00616 00069 00685</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWAP2</td>
<td>00764 00656 00685 00699 00765</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>THGS</td>
<td>00102 00272</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCONS</td>
<td>00724 00712</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGS</td>
<td>00612 00619</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRACK</td>
<td>00611 00551 00552 00738</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRK</td>
<td>00610 00507 00547 00666 00701 00713</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAL</td>
<td>00098 00112 00144</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
0001
SETHI  R 000000
SWAP2  P 014000
EXEC   X 000001
PRINT  :: 000002
PPSZ   P 000025
P15    P 000026
LWAM   000100
FNAMP  P 000027
LOOP   P 014045
PESET  P 014101
LPRST  R 014105
SHVE   P 014115
LPSV   R 014122
HADDR  P 014131
COUNT  R 014132
LIST   R 014133
IMAX   R 014134
ITAG   P 014135
N19    R 014136
RPYCD  P 014137
PP     R 014140
SIZE   R 014141
TAGS   R 014142
TRACK  R 014143
TRK    R 014144
RUL0   R 014145
A      000000
B      000001
FUAM   0000254
UBFUA  0000255
N100   R 014157
RSECT  000053
       000053
** NO ERRORS**
005+ 0005+ MAIN PROGRAM "SETHI" & SUBROUTINE "SWAP2"
0006+ 0010+ WRITTEN AT MCMaster UNIVERSITY, FEBRUARY 1975
0007+ 0011+ BY C. A. BRYCE, FOR A MSc PROJECT
0008+ 0012+
0009+ 0013+
0010+ "SETHI" IS A MAIN PROGRAM WHICH ALLOWS THE
0011+ SAVING OF A CONFIGURED VERSION
0012+ OF "SWAP2" INTO A DATA FILE
0013+ CALLED "SWAP2" THIS IS PORT
0014+ OF THE SWAPPING PROCESS IN THE
0015+ PLOTTEF PACKAGE (VERSION II)
0016+
0017+ METHOD
0018+
0019+ "SETHI" IS A SIMPLE PROGRAM WHICH CONFIGURES
0020+ HI-CORE, SO THAT IT CAN BE COPIED TO A DATA
0021+ FILE
0022+ THE CORE TO BE SAVED IS THAT CORE PAST THE
0023+ POINT AT WHICH CORE IS TO BE PARTITIONED
0024+ (CURRENTLY THE PARTITION POINT IS 24000B.)
0025+ TO CONFIGURE THIS CORE, A COPY OF THE BASE-
0026+ PAGE LINKS MUST BE MADE AT THE EXTREME END
0027+ OF HI-CORE. THEN AND ONLY THEN CAN AN "EXEC"
0028+ CALL BE MADE TO WRITE HI-CORE TO DISK
0029+
0030+
key variables

PP - partition point where hi-core begins, address of "SWAP2"

LWAM - a system constant - address of the last word available in memory for the user

BSZ - buffer size. This is calculated by determining the number of words above the partition point. It is then used in writing hi-core to disk.

N100 - constant - 100B

declare entry points "SETHI" & "SWAP2"

ENT SETHI
ENT SWAP2

declare externals "EXEC" & "PRINT"

EXT EXEC
EXT PRINT
ENTRY POINT "SETHI"

MAIN PROGRAM

SETHI NOP

BFSZ = LWAM - PP

BUFFER-SIZE = LWAM - PARTITIONPOINT

LDA PP
CMA,INA
ADA LWAM
STA BFSZ

WE SAVE LINK AREA AT LWAM - 200B

LDB LWAM
ADB N100
ADB N100

'SAVE' IS A SUBROUTINE LOCATED IN "SWAP2"

THAT COPIES 100B WORDS FROM LINK AREA

TO AREA STARTING WITH ADDRESS [E-REG]

JSB SAVE
0108+   NOW WRITE HI-CORE TO DISK ONTO A FILE
0110+   CALLED "SWAP2"
0112+
0113+
0114 00011 01601N  JSB EXEC
0115 00012 000021P  DEF ++
0116 00013 000026P  DEF P1G
0117 00014 000055  DEF +:
0118 00015 014090P  DEF SWAP2
0119 00016 000025P  DEF FR32
0120 00017 000027P  DEF FR1ME
0121 00020 000053  DEF PSE"T
0122+

0124+
0125+   STOP
0126+
0127 00021 01601N  JSB EXEC
0128 00022 000024P  DEF ++
0129 00023 000061  DEF +6
0130 00024 026021P  JMP ++3
0131+
0133+  STORAGE REQUIRED FOR "SETHI" AND
0135+   ALSO NOT NEEDED FOR "SWAP2";
0137+  
0138+  NOTE  SINCE "SETHI" WILL NOT EXIST
0139+    WHEN "SWAP2" IS EXECUTED ANYTHING
0140+    NEEDED BY "SWAP2" MUST BE LOCAL TO
0141+    "SWAP2".  (IE = ABOVE PP)
0142+  
0143+    THE REVERSE IS NOT TRUE.  AND INFACT
0144+    THE ROUTINE "SAVE" IS USED BY "SETHI"
0145+    WHEN IT LIES ABOVE THE PARTITION
0146+    POINT
0147+  
0148  00025  000000  BFSZ NOP
0149  00026  000017  P15 DEC 15
0150+  
0151+  LWAM IS A BASE PAGE CONSTANT THAT IS
0152+     AT ABSOLUTE ADDRESS 1000
0153+  
0154  00100  LWAM EDU 1000
0155  00027  051527  FILENAME ASC 3  SWAP2
0156+     00030  040520
0157+     00031  031040
0158+  

0159*
0160*
0161* "SWAP2" IS A ROUTINE THAT IS CALLED FROM "SWAP1"
0162* IN THE PLOTTER PACKAGE "PLOT" CALLS
0163* "SWAP1" WHICH IN TURNS SAVES THE
0164* EXISTING HI-CORE THEN IT COPIES IN
0165* "SWAP2" AND TRANSFERS EXECUTION
0166* TO "SWAP2". "SWAP2" IS EXPECTED TO
0167* RESTORE IT'S BASE PAGE LINK HERE AFTER
0168* SHAVING THE BASE PAGE LINKS OF "PLOT" &
0169* "SWAP1"
0170* THEN WHEN "SWAP2" HAS RECONFIGURED ITSELF
0171* IT MUST SAVE LO-CORE OUT ONTO DISK, SO THAT
0172* THIS AREA CAN BE USED AFTER IT DOES THIS
0173* IT SHIFTS THE "TAGS()" ARRAY TO 1ST WORD
0174* AVAILABLE IN USER MEMORY (FWAM), AND
0175* READS "LIST()" ARRAY RIGHT IN AFTER THIS
0176* IN CORE. THEN AFTER SETTING THINGS UP
0177* IT CALLS "PRINT", THEN IT HAS TO RESTORE
0178* LO-CORE AND BASE PAGE LINKS SO IT CAN
0179* SUCCESSFULLY RETURN
0180*
0181* KEY VARIABLES
0182*
0183* TRK — NEXT AVAILABLE TRACK ON DISK
0184* TRACK — TRACK ADDRESS OF START OF
0185* "LIST()" ARRAY
0186* TAGS — ADDRESS OF TAG ARRAY, THE TAGS
0187* THAT POINT INTO "LIST()"
0188* ITAG — INTEGER VALUE OF # OF STRINGS.
0189* IN "LIST()" OR SIZE OF "TAGS()"
0190* IMAX — INTEGER VALUE OF LAST VALUE OF
0191* "I" IN PLOT ROUTINE, OR SIZE
0192* IN WORDS OF "LIST()"
0193* FWAM — SYSTEM CONSTANT — ADDRESS OF
0194* 1ST WORD AVAILABLE TO USER IN
0195* MEMORY (AT ADDRESS 2548)
PAGE 0008 #01  < SWAP2 -- RESTORE LINKS / CALL PRINT / RETURN >

0204* SIZE -- VALUE CALCULATED IN "SWAP2" OF
0205* SIZE OF LO-COPE [FWAM,PP]
0206*
0207* UBFWA -- SYSTEM CONSTANT -- ADDRESS OF 1ST
0208* WORD IN LINK AREA
0209*
0210* LIST -- ADDRESS OF WHERE "LIST" AREA
0211* IS LOADED INTO LO-COPE FROM DISK.
0212*
0213* COUNT, ADDR
0214* -- COUNTPS WHD INDEX USED IN "SWAP2"
0215*
0216*

0218* 0219*
0220* FORCE "SWAP2" TO BE LOADED AT 'FWAM'+14000B
0221*
0222* SINCE 'FWAM' = 10000B
0223*
0224* "SWAP2" IS LOADED AT 24000B
0225*
0226 14000 ORG SETHI+14000B

0227*
0228* THIS IS WHERE SWAP2 IS LOADED
0229* 0230*
0231* 0232*
0233 14000 00000 SWAP2 NDP &
0234* 0235*
0236* GET ADDRESS OF ARGUMENT LIST [B-REG]
0237*
0238* THEN INCREMENT FOR PROPER RETURN
0239*
0240*
0241 14001 166000R LDB SWAP2, I
0242 14002 036000R ISZ SWAP2
GET ARGUMENT LIST AND GET REQUIRED VALUES & ADDRESSES

TPK VALUE NEXT AVAILABLE TRACK ON DISK

TRACK VALUE TRACK ADDRESS OF START OF LIST

TAGS ADDRESS OF START OF TAGS + HPRAX

ITAG VALUE SIZE OF TAGS

IMAX VALUE SIZE OF LIST

LDA B, I

INB

LDA B, I

STA TRACK

INB

LDA B, I

STA TAGS

INB

LDA B, I

STA ITAG

INB

LDA B, I

LDA A, I

STA IMAX
PAGE 0010 #01 << SWAP2 -- RESTORE LINKS / CALL PRINT / RETURN >>

0284*
0285*
0286* SAVE 1ST 100B WORDS IN LINK AREA
0287* 
0288* HT 'LWAM'-100B
0289*
0290+

0292 14024 064100 LDB LWAM
0293 14025 046157R ADD N100
0294*
0295* USE 'SAVE' AGAIN TO SAVE LINK AREA
0296*
0297 14026 016115R JSB SAVE
0298*

0300*
0301* NOW USE 'RESET' TO SET LINK AREA
0302* TO CONTENTS OF 'LWAM'-200B
0303* SAVE, STARTING ADDRESS IN "ADDR"
0304*
0305 14027 064100 LDB LWAM
0306 14030 046157R ADD N100
0307 14031 046157R ADD N100
0308 14032 076131R STB ADDR
0309*
0310 14033 016101R JSB RESET
0311*

0313*
0314*
0315* NOW CALCULATE SIZE OF LO-CORE
0316*
0317* SIZE = PP - FWAM
0318*
0319*
0320 14034 060254 LDA FWAM
0321 14035 003004 CMN, INA
0322 14036 042140R ADA PP
0323 14037 072141R STA SIZE
0324*
0326*  SAVE LO-CORE ON DISK WITH A CALL
0327*  TO "RWLO"
0328*
0329*  JSB RWLO
0330*
0331*  14040 016145R
0332  14040 016145R   JSB RWLO
0333*
0334*  NOW MOVE "TAGS()" ARRAY DOWN TO 'FWAM'
0335*  LDA ITAGS
0336  14041 062135R
0337  14042 003004
0338  14043 072132R
0339  14044 0064254
0340  14044 0064254
0341  14045 162142R
0342  14045 162142R  LOOP
0343  LDA TAGS, I
0344  14045 170001
0345  14046 170001
0346*  STA B, I
0347  14047 006004
0348  14050 036142R
0349  14051 036132R
0350  14052 026045R
0351  JMP LOOP
0352*  JUST FOR EXTRA PROTECTION PUT 1 WORD
0353*  OF ZERO'S BETWEEN "TAGS()" & "LIST()"
0354*  CLA
0355  14053 002400
0356  14054 170001
0357*  STA B, I
0358*
0359  14055 006004
0360  14055 006004
0361*  INB
0362*
0363*  B-REG CONTAINS STARTING ADDRESS OF "LIST()"
0364*  JMP LIST
0365*  SO SAVE IT IN "LIST"
0366*
0367*  14055 006004
0368*  14055 006004
0369  14056 076133R
0370  14056 076133R   STB LIST
PAGE 0012 #01 << SWAP2.-- RESTORE LINKS / CALL PRINT / RETURN >>

0371*
0372* READ IN "LIST( )" ARRAY FROM DISK
0373*
0374*
0375*

0377 14057 016001X JSB EXEC
0378 14060 014067R DEF ++7
0379 14061 000054 DEF +1
0380 14062 000055 DEF +2
0381 14063 114123R DEF LIST, I
0382 14064 014134R DEF IMAX
0383 14065 014143R DEF TRACK
0384 14066 000053 DEF RSEC1

0386*
0387*
0388* CALL PRINT(LIST, TAGS, ITAG)
0389*
0390* NOTE - 'LIST' IS ADDRESS - SO INDIRECT
0391*    'FUAM' IS ADDRESS WHERE TAGS() IS NOW
0392*
0393*
0394*
0395 14067 016002X JSB PRINT
0396 14070 014074R DEF ++4
0397 14071 114133R DEF LIST, I
0398 14072 100254 DEF FUAM, I
0399 14073 014135R DEF ITAG
0400*

0402*
0403* NOW WE RESEt LO-CORE
0404*
0405* 1ST SET 'RWRC' TO 1 FOR
0406* A READ REQUEST
0407*
0408*
0409 14074 002404 CLA, INA
0410 14075 072137R STA RWRCD
0412*    READ LO-CORE BACK IN
0413*    JSB RULO
0414*
0415 14076 016145R
0416*

0418*    JSB RESET
0419*
0420*    PUT LINK AREA BACK AS IT WAS
0421*    WHEN "SWAP2" WAS CALLED
0422*
0423*
0424 14077 016101R
0425*

0426*    NOW... RETURN
0427*
0428*    JMP SWAP2,I
0429*
0430 14100 126000R
0433*  
0434*  RESET -- COPIES A 100B WORD BUFFER
0435*     FROM 'ADDR' TO LINK AREA
0436* 
0438 14101 000000  RESET HOP
0439*  
0440*  SET COUNT TO -100B
0441*  SET B-REG WITH 'UBFWA'
0442*  
0443 14102 062157R  LDA M100
0444 14103 072132R  STA COUNT
0445*  
0446 14104 064255L  LDB UBFWA
0447*  
0449*  
0450*  A-REG <= [ADDR]
0451*  
0452 14105 162131R LPRST LDA ADDR.I
0454*  
0455*  
0456*  'DO A PRIVELED WRITE INTO PROTECTED
0457*  CORE ( WITHOUT ENCOUNTERING MEMORY
0458*  PROTECT )
0459*  
0460*  A-REG => [ B-REG ] & INB
0461*  
0462 14106 016001X  JSB EXEC
0463 14107 014111R  DEF ++2
0464 14110 014136R  DEF M19
0466*  
0467*  
0468*  NEXT 'ADDR'
0469*  
0470*  'FINISHED' ?? [COUNT]
0471*  
0472 14111 036131R  ISZ ADDR
0473 14112 036132R  ISZ COUNT
0474 14113 026105R  JMP LPRST
0475*  
0476 14114 126101R  JMP RESET, I
0477*  

---
0473*  
0480*  SAVE -- DOES THE REVERSE OF 'RESET'
0481*  BUT HAS NO PROBLEM READING FROM
0482*  PROTECTED COPE
0483*  
0485  14115 000000 SAVE NOP
0487*  
0488*  
0489*  SET COUNT TO -100B &
0490*  'ADDR' TO 'UBFWA'
0491*  
0492  14116 062157R  LDA N100
0493  14117 072132R  STA COUNT
0494*  
0495  14120 060255  LDA UBFWA
0496  14121 072131R  STA ADDR
0497*  
0498*  B SET BEFORE CALL TO THIS ROUTINE
0499*  
0500*  
0501*  A-REG <= [ADDR]
0502*  
0503  14122 162131R LPSV  LDA ADDR, I
0504*  
0505*  A-REG => [ B-REG ]
0506*  
0507  14123 170001  STA B, I
0508*  
0509*  NEXT 'ADDR', 'B-REG'
0510*  
0511*  DONE ?? (COUNT)
0512*  
0513*  
0514  14124 036131R  ISZ ADDR
0515  14125 034001  ISZ B
0516  14126 036132R  ISZ COUNT
0517  14127 026122R  JMP LPSV
0518*  
0519*  RETURN
0520*  
0521*  
0522  14130 126115R  JMP SAVE, I
0523*  

0525*
0526*
0527* STORAGE FOR RESET & SAVE
0528*
0529 14131 000000 ADDR HOP
0530 14132 000000 COUNT HOP
0531 14133 000000 LIST HOP
0532 14134 000000 IMIX HOP
0533 14135 000000 ITAG HOP
0534 14136 177755 H19 DEC -19
0535 14137 000002 RURCD OCT 2
0536 14140 014000R PP DEF SWAP2
0537 14141 000000 SIZE HOP
0538 14142 000000 TAGS HOP
0539 14143 000000 TRACK HOP
0540 14144 000000 TRK HOP

0542*
0543*
0544* 'RWLO' WRITES OR READS LO-CORE
0545*
0546* DEPENDING ON 'RURCD'
0547*
0548* INITIALLY 'RURCD'=2 (WRITE)
0549*
0550 14145 000000 RWLO HOP
0551 14146 016001X JSB EXEC
0552 14147 014156R DEF **+7
0553 14150 014137R DEF RURCD
0554 14151 000055 DEF +2
0555 14152 100254 DEF FWAM, I
0556 14153 014141R DEF SIZE
0557 14154 014144R DEF TRK
0558 14155 000053 DEF RSECT

0560 14156 126145R JMP RWLO, I
EQUIVALENCES
A & B REGISTERS
FWAM, UBFWA, RSECT,

A EQU 0
B EQU 1
FWAM EQU 254B
UBFWA EQU 255B
N100 OCT -100
RSECT EQU 53B
EQU 53B
END SETHI

** NO ERRORS**
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00570</td>
</tr>
<tr>
<td>ADDR</td>
<td>00529</td>
</tr>
<tr>
<td>B</td>
<td>00571</td>
</tr>
<tr>
<td>BFSZ</td>
<td>00148</td>
</tr>
<tr>
<td>COUNT</td>
<td>00530</td>
</tr>
<tr>
<td>EXEC</td>
<td>00067</td>
</tr>
<tr>
<td>FNAME</td>
<td>00155</td>
</tr>
<tr>
<td>FNAM</td>
<td>00573</td>
</tr>
<tr>
<td>IMAX</td>
<td>00532</td>
</tr>
<tr>
<td>ITAG</td>
<td>00533</td>
</tr>
<tr>
<td>LIST</td>
<td>00531</td>
</tr>
<tr>
<td>LOOP</td>
<td>00344</td>
</tr>
<tr>
<td>LPRST</td>
<td>00452</td>
</tr>
<tr>
<td>LPSV</td>
<td>00503</td>
</tr>
<tr>
<td>LUAM</td>
<td>00154</td>
</tr>
<tr>
<td>N100</td>
<td>00576</td>
</tr>
<tr>
<td>N19</td>
<td>00534</td>
</tr>
<tr>
<td>PP</td>
<td>00536</td>
</tr>
<tr>
<td>PRINT</td>
<td>00068</td>
</tr>
<tr>
<td>R15</td>
<td>00149</td>
</tr>
<tr>
<td>RESET</td>
<td>00438</td>
</tr>
<tr>
<td>RSECT</td>
<td>00577</td>
</tr>
<tr>
<td>RW</td>
<td>00555</td>
</tr>
<tr>
<td>SETHI</td>
<td>CROSS-REFERENCE SYMBOL TABLE</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>SETHI</td>
<td>00078</td>
</tr>
<tr>
<td>SIZE</td>
<td>00537</td>
</tr>
<tr>
<td>SWHP2</td>
<td>00273</td>
</tr>
<tr>
<td>THGS</td>
<td>00538</td>
</tr>
<tr>
<td>TPWC</td>
<td>00529</td>
</tr>
<tr>
<td>TPH</td>
<td>00540</td>
</tr>
<tr>
<td>UBFWA</td>
<td>00574</td>
</tr>
</tbody>
</table>
SUBROUTINE GRAPH - MARCH 1, 1975

FTH4.L

SUBROUTINE GRAPH : XMIN, XMAX, XSCAL,
1 YMIN, YSCAL,
2 HOFN, F

C
C SUBROUTINE "GRAPH"

C
C WRITTEN AT McMaster UNIVERSITY, FEBRUARY 1975
C BY C A Bryce, FOR A MSC PROJECT.
C
C "GRAPH" IS A UTILITY PROVIDED IN CONJUNCTION
C WITH A PLOTTING PACKAGE USED ON
C AN ELECTROSTATIC PRINTER, THAT
C PERMITS THE PLOTTING OF FUNCTIONS
C THIS ROUTINE WILL GRAPH FUNCTIONS
C FROM "XMIN" TO "XMAX" USING A
C SCALE OF "XSCAL" LENGTHWISE
C DOWN THE PAGE IT WILL USE
C "YMIN" AND TAKING "YSCAL"
C CALCULATE A "YMAX". IN ADDITION
C IT WILL GRAPH UP TO 10 FUNCTIONS
C SIMULTANEOUSLY.
C
C "GRAPH" USES 4 SUBROUTINES
C
1) EXEC - THE SYSTEM EXECUTIVE IS USED TO
C GENERATE NEEDED PAGE EJECTS
C
2) INSRT - A ROUTINE THAT ADDS NECESSARY BITS
C INTO OUTPUT BUFFER "LINE()"
C
3) BFOUT - THE ROUTINE THAT PUTS "LINE()" OUT
C TO THE PRINTER/PLOTTER
C
4) F - THIS ROUTINE MUST BE SUPPLIED BY
C THE USER, AND PASSED BY NAME TO
C "GRAPH" THROUGH THE EXTERNAL "F"
C THIS ROUTINE HAS 2 ARGUMENTS,
C THE 1ST AN INTEGER ( <= 11 )
C THAT SPECIFIES WHICH FUNCTION TO
C EVALUATE, AND THE 2ND A REAL
WHICH IS THE VALUE TO BE USED IN THE
PARTICULAR FUNCTION THE USER
SUPPLIES IN MAIN PROGRAM THAT CALLS
"GRAPH" & THE FUNCTION "F:1:0" THAT
IS IN TURN CALLED FROM "GRAPH"

METHOD

AFTER CALCULATING CONVERSION FACTORS
"DELT:" & "DELTY", REPORTING "MIN" USED.
& PUTTING OUT TOP BORDERS. "GRAPH" GETS
DOWN TO WORK AND BASICALLY BY TAKING
3 CONSECUTIVE VALUES OF THE FUNCTIONS
Y-LAST, Y-PRESENT & Y-NEXT, THE ROUTINE
INTERPOLATES BACK AND FORWARD TO GET A
FAMILY OF VALUES FOR THIS FUNCTION AND
THIS VALUE OF : AS UP TO 10 FUNCTIONS
CAN BE PUT OUT AT ONCE. THE 3 VALUES
ARE SAVED IN X(L), X(P) & X(N)

IN ADDITION AT EACH VALUE OF X
THE SIDE BORDERS AND POSSIBLY THE X-AXIS
IS PLOTTED ALSO THERE IS A CHECK TO
CATCH THE OCCURRENCE OF THE Y-AXIS, AND
PLOT IT

WHEN "XMAX" IS REACHED, THE VALUES OF
"YMIN", "YMAX", "XMAX", "XSCAL",
& OF "YSCAL" ARE REPORTED

KEY VARIABLES
XMIN - REAL VARIABLE FOR MINIMUM X
XMAX - REAL VARIABLE FOR MAXIMUM X
XSCAL - REAL VARIABLE FOR X-SCALE TO USE
YMIN - REAL VARIABLE FOR MINIMUM Y
YSCAL - REAL VARIABLE FOR Y-SCALE TO USE
NDFN - INTEGER NUMBER OF FUNCTIONS
SUBROUTINE GRAPH    MAR 1, 1975    PAGE 3

F    FUNCTION EXTERNAL PASSED TO "GRAPH"

REAL XMIN, XMAX, XSCALE
REAL YMIN, YSCALE
INTEGER INP, IYP
EXTERNAL F

LOCAL VARIABLES

MAX    INTEGER CONSTANT FOR THE NUMBER
OF POINTS AVAILABLE IN LINE

CNFCT    REAL CONSTANT FOR CONVERSION
FACTOR NEEDED TO CONVERT X & Y
INTO INCHES IN PLOTTER UNITS

LINDEX    INTEGER BUFFER USED TO
STORE FUNCTION IMAGES

DELTX    REAL VALUE OF STEP 1 LINE
IN X-DIRECTION

DELTY    REAL VALUE OF A STEP IN
Y-DIRECTION

YMAX    LOCAL REAL VALUE CALCULATED
FROM "YMIN" & "YSCALE"

X    REAL VALUE OF CURRENT X

XAXIS    INDEX OF X-AXIS IN "LINE()"

XL    REAL VALUE OF LAST X

J    INDEX USED TO FLAG FUNCTIONS

Y    INDEX USED FOR "LINE()"

IYL()    INTEGER ARRAY SAVING LAST Y
FOR JTH FUNCTION

ICY()    INTEGER ARRAY SAVING PRESENT Y
SUBROUTINE GRAPH --- MARCH 1, 1975

C FOR JTH FUNCTION
C
C IWK(J) = INTEGER ARBITRARY SAVING NEXT Y
C FOR JTH FUNCTION
C
C IYPJ, IYPJB, IYPJF
C = INTEGER INDEXES USED
C FOR ITH INTERPOLATION
C
C RPMF(5) = IYPJ = IYPJB
C
C INTEGER LINC, IYL, IYP, ITH
C DIMENSION LINC (35), IYL (10)
C DIMENSION IYP (10), ITH (10)
C INTEGER IMAX, JMAX, Y, IMIN, IMAX
C INTEGER IYPJ, IYPJB, IYPJF
C REAL DELT, DELTY, IMAX, JMAX

C INITIALIZE DATA
C
DATA MAX, 550
DATA CHFCT, 72.5
C
C
C**********************************************************************
C
C START OF SUBROUTINE "GRAPH"
C
C FIRST - CHECK TO SEE IF # OF FUNCTIONS IS
C IN CORRECT RANGE -- [ 1, 10 ]
C PEPOLY WITH APPROPRIATE ERPOP MESSAGE
C IF NECESSARY AND RETURN
C
C IF ( NOFN GT 10 AND NOFN LE 10 ) GO TO 10
C
C WRITE * , N, NOFN
C 29 FORMAT ( 25H ERPOP IN CALL TO "PRINT", 1 , 25H # OF FUNCTIONS IN CALL = , 15 )
C RETURN
C
C**********************************************************************
FROM "YSCAL" & "YSCAL" DETERMINE THE
NECESSARY VALUES FOR "DELTX" & "DELTY"
RESPECTIVELY.

ALSO DETERMINE THE PROPER INDEX FOR
"XMIN" IF "XMIN" NOT IN [ 1, "MAX" ]
SET "XMIN" TO 1 AND HIDE IT ON LEFT
BORDER.

\[ \text{DELTX} = 10 / \left( \text{YSCAL} \times \text{CNPT} \right) \]
\[ \text{DELTY} = 10 / \left( \text{YSCAL} \times \text{CNPT} \right) \]
\[ \text{XMIN} = 1 \]
\[ Y = \begin{cases} 
\text{IFILI} < \text{YMIN} & \text{DELTY} \\
\text{IF} > Y \text{GE 1 AND } Y \text{ LE MAX} & \text{XMIN} = Y 
\end{cases} \]

SKIP TO TOP OF NEXT PAGE WITH
EXECUTIVE CALL

THEN REPORT "XMIN" VALUE USED &
PUT OUT TOP BORDER

CALL EXEC ( 3, 11108, 63 )

WRITE ( 8, 100 ) XMIN

\[ \text{RFORMAT} \times 30X, 1 \times \text{XMIN} = 1 \times 10.3, / \]

DO 15 \[ Y = 1, 35 \]

\[ \text{LINE} < \text{Y} = 1777778 \]
CALL BFOUT < \text{LINE} >

INITIALIZE "IYL()" & "IYP()" WITH
VALUES OF FUNCTIONS FOR X EQUAL TO
"XMIN"-1 & TO "XMIN" RESPECTIVELY

\[ \text{XL} = \text{XMIN} - \text{DELTX} \]
\[ X \times \text{XMIN} \]

DO 20 \[ J = 1, \text{NOFN} \]
IYL(J) = IFILI < (F(J, XL)-YMIN),DELTY>

\[ \text{IYP}(J) = \text{IFILI} < (F(J, X)-YMIN), \text{DELTY} > \]

GET NEXT VALUE OF X, AND CALCULATE
SUBROUTINE GRAPH --- MARCH 1, 1975

"IYH : " 'FOR EACH FUNCTION

30 X = X + DELT:

DO 31 J = 1, NOFN
31 IYN(J) = IFI..(F(J,X)-YMIN).DELTY

IS THIS WHERE THE Y-AXIS BELONGS ???

IF = IS SITUATION -- X-DELTX = 0 0 0
- X HAS CHANGED SIGN WHICH MEANS THAT
X + X-DELTX IS TIVE

IF SO PUT OUT Y-AXIS

IF < X+(X-DELTX) GE 0 0 \ GO TO 33

DO 32 Y = 1, 35.
32 LINE (Y) = 177777B

GO TO 50

*****************************************************************

FOR EACH FUNCTION PUT NEEDED POINTS
INTO "LINE( )"

NOTE
- IN ORDER TO GET A CONTINUOUS GRAPH
FOR EACH FUNCTION, "GRAPH" INTERPOLATES
BACKWARDS AND THEN FORWARDS HALFWAY.
AND IT IS THIS RANGE THAT IS INCLUDED
NOT JUST THE Y, VALUE

33 DO 40 J = 1, NOFN
     IYPJ = IYP(J)
     IYPJB = IYPJ - (IYPJ-IYL(J))/2
     IYPJF = IYPJ + (IYN(J)-IYPJ)/2
     IMIN = MIN0 (MAX, IYPJB, IYPJ, IYPJF)
     IMAX = MAX0 (-1, IYPJB, IYPJ, IYPJF)

     IF (IMIN.LT.1) IMIN = 1
SUBROUTINE GRPH

IF (IMAX GT MAX) IMAX = MAX

DO 35 Y = IMIN, IMAX
   35 CALL INSRT (LINE, Y)

CONTINUE

NOW INCLUDE THE X-AXIS ALONG WITH
THE LEFT & RIGHT BORDERS

CALL INSRT (LINE, 1)
CALL INSRT (LINE, XMIN)
CALL INSRT (LINE, XMAX)

BUFFER OUT CONTENTS OF "LINE()"

CALL Bfout (LINE)

NOW SET "IYL()" == "IYP()
& "IYP()" == "IYN()"

UNLESS ALL THE GRAPHING HAS BEEN
DONE (X > XMAX) PROCESS FOR
NEXT X ( => 30)

DO 51 J = 1, NOFN
   IYL (J) = IYP (J)
   IYP (J) = IYN (J)
   IF (X.LE.XMAX) GO TO 30:

PUT OUT BOTTOM BORDER AND THEN
REPORT "YMIN", "XMAX", "YMAX",
& "XSCAL", AND "YSCAL" THAT
WAS USED TO CREATE GRAPH.

SKIP TO TOP OF NEXT PAGE AND
RETURN
DO 60 Y = 1, 35
60 LINE (Y) = 1777778
CALL BFSOUT (LINE)

C

TMAX = YMIN + FL0W * MAX * DELTY
WRITE (8, 101) YMIN, YMAX, YMAX
101 FORMAT (1X, 2F6.3, 2G6.3)

C

WRITE (8, 102) NSCAL, VSCAL
102 FORMAT (1X, 3H SCALE USED, 12X)
1 23HIN X-DIRECTION DOWN
2 10H1 UNIT = , G10 3.
3 THINCHES = , 12X.
4 23HIN Y-DIRECTION H-CROSS
5 10H1 UNIT = , G10 3.
6 THINCHES

CALL EXEC (3, 11108, 63)

RETURN

C

C

C

C

C

C

C

END

END$
APPENDIX E

Further Examples of Plotter Output

To further illustrate what the plot package is capable of at McMaster, seven more harmonographs (see 4.1.2.) are provided.
REFERENCES

1. Hewlett-Packard 2100A - Reference Manual,
   Hewlett-Packard Corporation,
   11000 Wolfe Road,
   Cupertino, California 95014.
   (HP 02100 - 90001)

2. Hewlett-Packard Assembler Manual,
   Hewlett-Packard Corporation,
   11000 Wolfe Road,
   Cupertino, California 95014.
   (HP 02116 - 9014)

3. Hewlett-Packard Fortran IV Reference Manual,
   Hewlett-Packard Corporation,
   11000 Wolfe Road,
   Cupertino, California 95014.
   (HP 5951-1321)

4. Moving-Head Disc. Operating System,
   Hewlett-Packard Corporation,
   11000 Wolfe Road,
   Cupertino, California 95014.
   (HP 02116 - 91779)

5. DOSM Operating System Listings,
   Hewlett-Packard Corporation,
   11000 Wolfe Road,
   Cupertino, California 95014.
   (HP 24225 - 6000X Rev.D)

6. Benson-Lehner Plotting System for CDC 6400
   Data Processing and Computing Centre,
   McMaster University,
   Hamilton, Ontario.
   (Publication 5.2.1.)

7. Software - Practice and Experience,
   Volume 2 (1972)
   Computer Recreations,
   Wiley Interscience.
   (Pages 293 to 297)

8. Investigations of Available Sorting Algorithms
   and their Behavior for Different Orderings,
   McMaster University,
   Hamilton, Ontario.