MACINTOSH - BRIDGEPORT
COMMUNICATIONS
CAD/CAM

A Thesis Presented to the Faculty of the
College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Dinesh Dhamija
November, 1988
# TABLE OF CONTENTS

Table of Contents .............................................. i
List of Illustrations ........................................... ii
Acknowledgements ................................................. iii

## INTRODUCTION
- INTERFACING THE MACINTOSH COMPUTER TO THE BRIDGEPORT COMMUNICATIONS ................................. 1
- PRELIMINARY RESEARCH - PARABOLIC REFLECTORS .................................................................................. 10
  - THE PROGRAM ....................................................... 21
    - GENERICS ......................................................... 23
    - SPECIFICS ......................................................... 24

## THE SOFTWARE
- THE GENERIC STRUCTURE ............................................ 32
- THE LIGHTSPEED PASCAL COMPILER .................................. 34
- THE LIGHTSPEED PASCAL PROJECT ................................. 38

## SOFTWARE DESCRIPTION
- THE DATA ARRAY ..................................................... 43
- UNIT DEFINITIONS AND EXPLANATIONS .......................... 44
  - MacCamGlobals ................................................. 47
  - UNIT MiscellaneousProcs ...................................... 47
  - PROCEDURE INITHAND ........................................... 49
  - PROCEDURE INITCROSS .......................................... 49
  - PROCEDURE INITPENCIL .......................................... 50
  - PROCEDURE CLEARALL ......................................... 50
  - PROCEDURE ERASEDIALOG .................................... 51
PROCEDURE DRAWSCALE
PROCEDURE DRAWZICON
PROCEDURE DRAWDIALOG

UNIT MoreProcs;

PROCEDURE RECDRAW
PROCEDURE PlotDot
PROCEDURE PlotGrid
PROCEDURE DrawBox
PROCEDURE SetUp
PROCEDURE YesNo
PROCEDURE LoadDrawing

UNIT SaveDraw

PROCEDURE SaveAll

UNIT FrontView

PROCEDURE FrView
PROCEDURE HangAround
PROCEDURE PressAKey
PROCEDURE Done

UNIT BridgePort;

PROCEDURE Post

PROGRAM MACCAMLP1;

PROCEDURE SHOWXY
PROCEDURE SCALESELECT
PROCEDURE TOOLCHANGE
PROCEDURE GridSet
<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCEDURE WindowInit</td>
<td>92</td>
</tr>
<tr>
<td>PROCEDURE Zset</td>
<td>93</td>
</tr>
<tr>
<td>PROCEDURE ToolSet</td>
<td>94</td>
</tr>
<tr>
<td>PROCEDURE Mtrack</td>
<td>98</td>
</tr>
<tr>
<td>PROCEDURE FrDecision</td>
<td>98</td>
</tr>
<tr>
<td>PROCEDURE FrameMill</td>
<td>99</td>
</tr>
<tr>
<td>PROCEDURE DrillHole</td>
<td>101</td>
</tr>
<tr>
<td>PROCEDURE CircleMill</td>
<td>102</td>
</tr>
<tr>
<td>PROCEDURE CutterComp</td>
<td>104</td>
</tr>
<tr>
<td>PROCEDURE CompOff</td>
<td>106</td>
</tr>
<tr>
<td>PROCEDURE DrawLine</td>
<td>107</td>
</tr>
<tr>
<td>PROCEDURE Line</td>
<td>107</td>
</tr>
<tr>
<td>PROCEDURE SetSnap</td>
<td>110</td>
</tr>
<tr>
<td>PROCEDURE Zdepth</td>
<td>110</td>
</tr>
<tr>
<td>THE MAIN PROCEDURE</td>
<td>111</td>
</tr>
<tr>
<td>THE FRONT VIEW ALGORITHM</td>
<td>114</td>
</tr>
<tr>
<td>TESTING AND DEBUGGING</td>
<td>122</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>124</td>
</tr>
<tr>
<td>EXAMPLE 1</td>
<td>124</td>
</tr>
<tr>
<td>EXAMPLE 2</td>
<td>129</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>133</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>134</td>
</tr>
<tr>
<td>APPENDIX C</td>
<td>136</td>
</tr>
<tr>
<td>APPENDIX D</td>
<td>151</td>
</tr>
<tr>
<td>APPENDIX E</td>
<td>157</td>
</tr>
<tr>
<td>APPENDIX F</td>
<td>163</td>
</tr>
</tbody>
</table>
SOURCE CODE
USER'S MANUAL
PARABOLA CODE

Appended
Appended
Appended
LIST OF ILLUSTRATIONS

Current Software vs MacCam........................................2
Project Outline.........................................................7
Macintosh-Bridgeport Communication Ports..................11
Bridgeport Control Characters.................................15
Summary of Communication Parameters.......................16
Macintosh Bridgeport Cable Wiring............................18
Parabola Geometry......................................................22
Parabola Machining - the concept............................25
Structure of Procedure Post......................................28
Structure of the Data Array.......................................45
The Front View Display - Screen Layout.......................118
The Front View Display - Example.............................121
Example 1...............................................................125
Example 2...............................................................130
MacCam Menu Structure...........................................132A
RS-232 Character Frame............................................145
Acknowledgements

The author wishes to take a moment here to express gratitude to those people that helped make this project a reality.

This project would never have been without the constant guidance of Dr. Urieli, Associate Professor of Mechanical Engineering, Ohio University. Dr. Urieli's constant questioning of each algorithm in the software led the author through new avenues in the design of algorithms. His help in providing guidance in the selection of reference books and at times the books themselves were invaluable. The author also appreciates the detailed reviews that Dr. Urieli has given this document through each revision, making it more readable and consistent in style.

A very special thanks is due to Mr. John A. Deno, Assistant Professor of Industrial Technology, for introducing me to the intricacies of the Bridgeport CNC machine and explaining the programming in detail. These discussions were invaluable in the writing of the code generator. The author really appreciates the extra time that Mr. Deno spent, next to the Bridgeport along with the author, sharing the frustrations of an improper download, or a wayward tool seeming to have a mind of it's own, and the constructive suggestions that helped remove these bugs.

Thanks to Dr. Mohamad Deghani, Assistant Professor of Mechanical Engineering, for reviewing the manuscript and watching the demonstrations of the software, asking questions about the heart of the main algorithms. Thanks to him for associating himself with the project in a relatively short time.

Thanks to Mr. Timothy Sexton, Assistant Professor of Industrial Technology for so willingly agreeing to part with the Macintosh on his desk for long periods of time. The entire project was done on that machine, itself an indication of how long Mr. Sexton had to part with the use of the Mac.
INTRODUCTION

The main task of this project was to initiate communications between the Macintosh\(^1\) personal computer and the Bridgeport\(^2\) CNC Milling machine and write a CAD/CAM package that is simple to use. NC machines have been in use since the fifties to machine various components to a high degree of accuracy.

The first NC machines used programs written on punched tape that the machine read as it executed the program. This was terribly inconvenient since an error in punching the tape led to reprogramming the entire tape or splicing in the correction physically. Paper tape NC's were also unsuitable for mass production operations as the tape had to be rewound before each run. The advent of sophisticated computer technology has brought about considerable change in the CNC machine tools available today.

In the current time, CNC machines may be programmed remotely and the program is then downloaded into the CNC machine via communications ports. The current machines are equipped with Random Access Memories and are capable of multiple program runs without any rewinding of paper tapes and other problems.

Until the early 1980's, computers were still limited to large corporations due the expense in acquiring and maintaining

\(^1\) Apple, Macintosh are trademarks of Apple Computer Inc.

\(^2\) (C) Bridgeport Machines Inc.
CURRENT SOFTWARE VS MAC CAM

CURRENT SOFTWARE

USER

CAD SOFTWARE

DXF OR IGES

CAM PROGRAM

G CODES

MAC CAM

USER

CAD SOFTWARE (IN MAC CAM)

G CODES
them. Thus CNC machining in its most flexible form was limited to large production lines, in corporations that could afford large systems. The introduction of the personal computer in 1980 considerably changed the situation. Small job shops were able to program and control their machines with the same ease as the large corporations.

At that time, software was cumbersome to use and the hardware was not developed enough to make the personal computer a tool that every person in the machine shop could use. It required skilled operators. The keyboard used to be the only input device, making it difficult to define the geometry of the part to be machined.

At the present time, there is an abundance of microcomputers with extremely powerful software and sophisticated hardware that fulfill the task of designing the part and machining it with considerable ease.

The tradition of the software currently available is to separate the design and machining parts of the CAD/CAM process. This is an excellent idea since the user may focus his complete attention on the design and drawing of the part and then take care of the machining aspects. This affords the user considerable power while designing the part. The computer assisted drafting software is completely responsible
for the generation of the part geometry. Some examples of CAD software available for the Macintosh computer are AUTOCA0\textsuperscript{3}, VERSACAD\textsuperscript{4}, MINICAD\textsuperscript{5}, to mention a few. This geometry is also used to make detail drawings for the shop floor, and in larger systems, it is used to view completed three dimensional models of the final product, using three dimensional solid modelling techniques. The geometry, once generated, is then used to create an IGES\textsuperscript{6} file or a DXF\textsuperscript{7} file. These files are then used in the second stage to import the geometry into a CAM program.

The second stage is the CAM program, where the tool path is generated and the program for the machine tool is actually generated. The use of the IGES or DXF files gives the user independence to choose the CAD software, without any restrictions being imposed by the CAM software. Within the CAM software environment, the first step is to import the geometry. The tool paths are then explicitly specified by using a digitizer or a mouse. Tool paths are generated based on breaking down the geometry into logical components. The

\begin{itemize}
\item \textsuperscript{3} (C) Autodesk Inc.
\item \textsuperscript{4} (C) VersaCad Corporation
\item \textsuperscript{5} (C) Diehl Graphsoft
\item \textsuperscript{6} International Graphics Exchange Specification - Graphical File Format
\item \textsuperscript{7} Data Exchange Format - Graphic File format
\end{itemize}
user assumes complete responsibility for the breaking down of the geometry into smaller segments and the location and timing of all tool changes. The post processor is then used to generate the program for the NC machine. Most CAM software are written using standard G codes\(^8\). This affords them the power of being machine independent. This machine independence is gained at the expense of increased program complexity and length.

The process described above is excellent for complex geometry, where powerful CAD software is invaluable, and is also very well suited for Research and development environments. In the small machine shops, the operator is forced to learn the functioning of the CAD software, even if his job does not call for the use of software as powerful. After mastering the CAD software, he still has to deal with the task of learning to use the CAM software. The small shop operator is unlikely to do any detail part drawing or complex simulations. The basic aim of the operator is to generate the tool paths with minimum effort.

The aim of this project is to explore the possibility of the design of one such software system. This software allows the user to generate a geometry within it's environment. This

\(^8\) G code Description is given in detail elsewhere in the manual
geometry is actually a pseudo geometry, since the user is really defining the tool path in the guise of geometry. This eliminates the need to separate the process of generating the geometry and specifying the tool paths. The software was designed to generate the program for the Bridgeport Series I Vertical Milling machine with an R2E4 control, operating under the BOSS 9 operating system. This program is unique in the sense that it uses the canned cycles in the Bridgeport, making the programs short and extremely readable after the post processor has generated it.

The entire project was divided into three phases.

1) Interfacing the Macintosh computer to the Bridgeport vertical mill. This phase in itself was divided into two parts. The first sub-phase was connecting the Macintosh to the Bridgeport's editing port, to allow remote editing. The second sub-phase was to interface into the Bridgeport's communication port for uploading and downloading files to and from the Bridgeport. Mockterminal⁹ was the first package to be tried out, since it is a desk accessory and it would have been handy to use a desk accessory as the communications package. There were some problems however, as discussed

⁹ (C) C. E. Software
PROJECT OUTLINE

PHASE 1
INTERFACING MACINTOSH BRIDGEPORT

PHASE 2
PRELIMINARY RESEARCH PARABOLA MACHINING

PHASE 3
SOFTWARE DEVELOPMENT

PHASE 4
DEBUGGING AND TESTING
in the communications section. Versaterm\textsuperscript{10} was also considered, but the final choice for the communication was Red Ryder\textsuperscript{11}.

2) The second phase was to do some preliminary research on the structure of the post processor algorithm. Since the final post processor would have to deal with several different G codes, it was thought practical to experiment with a program with a limited number of options. Thus software was written\textsuperscript{12} to have the user input the Focal length and the diameter of the parabola, and then generate the geometry on the screen. Then if the user wishes, the G codes to machine the parabolic shape were generated automatically by the software. This phase was also used to check the interfacing done in phase 1 of the project. The G codes were downloaded to the Bridgeport and the parabolic shape was machined.

3) The next phase was the writing of the main software itself. The initial development of the main program was done using the Macintosh Pascal interpreter. The Final program is presented as a Lightspeed Pascal application

\textsuperscript{10} (C) Lonnie R. Albeck

\textsuperscript{11} Version 10.0 by Scott Watson, (C) The FreeSoft Company

\textsuperscript{12} The original software was written by the author. The input section was later modified by Dr. Urieli for a presentation to his class, to make it fit into his class theme.
so that the user does not have bother with the source code and the Pascal interpreter, every time the software is used. The task was to develop the post processor, the user interface and the graphics interface. It was also designed with the Macintosh's mouse very much in mind. All inputs to the system are mouse oriented, except for loading and saving files where the user has to use the keyboard to name the file to be saved or loaded.

4) The final stage involved the testing of the software written. This phase was actually almost concurrent with phase 3, since the magnitude of the software would make debugging a nightmare if it were done at the end of the code generation.

Following is a detailed report of all the phases that are mentioned above. It lists all the venues explored and the various problems encountered in the process. It is then followed by a user's manual for the software.
The Macintosh and the Bridgeport communicate through serial ports. The typical communication standard is RS-232 standard. RS-232 is designated as the EIA\(^1\) standard. It is also known as the CCIT\(^2\) V.24 standard. In this document, the term RS-232 is used while referring to the interface. RS-232 defines the electrical signal characteristics, mechanical interface and the functional description of the interchange circuits. Included in Appendix C, is a complete description of the RS-232 communication standard, and the terms used in defining the standard. If the reader is unfamiliar with the RS-232 protocol, it is recommended that appendix C be reviewed before reading this section.

The Bridgeport CNC vertical mill has two ports. Port A is the editor port and port B is the upload/download port. Through port A, the user is connected directly to the Bridgeport internal editor (BOSS 9) and can remotely edit the information in the Bridgeport's buffer.

The first task was to determine the communication parameters and to get the computer to communicate with the editor.

---

\(^1\) Electronic Industries Association, See Appendix C

\(^2\) Consultive Committee International Telegraph and Telephone - A United Nations group that specifies International Standards.
bits, no parity, one stop bit, full duplex transmission. This setting was successful and the Macintosh computer could be used as a remote editing terminal. All editing functions were accessible through the Macintosh keyboard. The computer was functioning as a dumb terminal providing remote access to the Bridgeport controller.

On trying to work with port B, the upload/download port, there were some problems. It was determined that the Bridgeport settings for this port were 7 data bits, even parity, 1 stop bit and a programmable baud rate that was chosen to be 2400 for this connection. It was possible to receive files from the Bridgeport to the Macintosh using this format. On trying to upload any file, however, there was an instant error message generated on the Bridgeport. The machine would not even accept files previously downloaded from it. Several different parameters were tried but none seemed to work. Eventually it was decided to use Red Ryder as the communications package since the cause of the trouble seemed to be the software handshaking protocol used by Mockterminal.

When using Red Ryder, it was necessary to establish communications with the editor port first since this was familiar ground. It took a while to get through the many powerful features that Red Ryder has to offer, but finally
communications to the editor port was completely established through Red Ryder. Several different settings were tried over the next few days, with moderate success. Again, it was possible to download files from the Bridgeport to the Macintosh. The downloaded files would begin with a box character, and end with a similar character.

From the author's past experience with interfacing the IBM PC to the Bridgeport, the codes were interpreted as follows. The beginning box character was interpreted to be a CONTROL R character, otherwise known as the ASCII character 18, the DC2 general purpose toggle code. This is the character that signals to the waiting device that transmission from the remote device is beginning. Following the CONTROL R character, the data itself is sent. To terminate the transmission in a legal manner, the Bridgeport uses a CONTROL D, followed by a CONTROL T character. The CONTROL D is the ASCII character 4, and is known as the EOT character, EOT standing for end of text. Then the CONTROL T character is another General purpose toggle character, called the DC4 character, which is ASCII 20. This terminates the communication session between the devices. In the Macintosh character set, these special control characters are represented by the box character. The Bridgeport was automatically inserting these characters in the file and in a text editor on the Macintosh they would appear as boxes.
After several days of experimentation with Red Ryder, the communication parameters that would finally work on port B, the upload/download port were, a baud rate of 2400 (this would work with any baud rate upto 9600, the Bridgeport's limit), 7 data bits, even parity, one stop bit and full duplex transmission. Using these parameters, it was possible to receive files from the Bridgeport, and send back any file that was received previously from the machine. There was still a problem with files created by the user on the Macintosh text editor. The problem was thought to be the CONTROL characters. A futile search was conducted to find a way of generating those codes from the Macintosh keyboard. At a certain stage, it was decided to cut and paste those characters from a file previously sent by the Bridgeport onto a user created file via the Macintosh clipboard. This would at least give an indication if the right approach was being taken. This endeavour met with success and user created files could also be uploaded to the Bridgeport without any errors. It was still impossible to generate those codes from the keyboard. The search for the mystery keys was eventually abandoned since the software that was going to generate the G codes was going to be written in Pascal. It would be very easy to insert a

WRITELN(CHR(18));
BRIDGEPORT CONTROL CHARACTERS

\^R \{ASCII 18\}

G...X...Y...Z........;
G...X...Y...Z........;
...
...
...
...
...
...
G...X...Y...Z........;
\^D\{ASCII 14\}\^T\{ASCII4\}
SUMMARY OF COMMUNICATION PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baud Rate</td>
<td>1200 - 9600</td>
</tr>
<tr>
<td>Parity</td>
<td>Even</td>
</tr>
<tr>
<td>Data Bits</td>
<td>7 Bit ASCII</td>
</tr>
<tr>
<td>Stop Bit</td>
<td>1</td>
</tr>
</tbody>
</table>
to generate the ASCII 18, the control R character. Other characters could be generated in a similar manner. To check the validity of this method, a small pascal program was written where the program would open a previously created text file and insert the appropriate characters in the right places and rewrite the file to disk. This was tested by downloading the modified file back into the Bridgeport. The procedure was successful.

Thus, it was possible to upload and download files to the Bridgeport successfully and the editor port was also communicating flawlessly. It was later discovered, that whether the user was using the editor port or the upload/download port, communication settings of 7 data bits, Even parity, one start bit, full duplex transmission and any baud rate selected by the user would allow perfect communication.

The following paragraph gives the cable specifications used in the communication. It was determined that the Bridgeport used only software handshaking and thus the hardware handshaking lines could conveniently be ignored. This left 4 pins to be dealt with, the signal ground, the frame ground, the transmit and the receive lines. There was already a cable existing from the Bridgeport to an IBM PC, a connection that had previously been established by the author. This cable to
MACINTOSH BRIDGEPORT CONNECTIONS

DB25 CONNECTOR  DB9 CONNECTER

<table>
<thead>
<tr>
<th>FRAME GROUND</th>
<th>1</th>
<th>1 FRAME GROUND</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSMIT (TxD)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>RECEIVE (RxD)</td>
<td>3</td>
<td>3 SIGNAL GROUND</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5 RECEIVE (RxD)</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>SIGNAL GROUND</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9 TRANSMIT (TxD)</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

DB25 CONNECTOR - MALE  DB9 CONNECTOR - MALE
FRONT VIEW               FRONT VIEW

1 13
14 25
6 5
9
the IBM PC terminated in a regular DB25 pin connector in the room in front of the Bridgeport. It was decided to build a cord to patch into that DB 25 connector, so that the user may decide to use the IBM PC as the remote terminal by directly connecting the DB 25 connector to the serial port of the PC or to use the Macintosh by plugging in this custom designed cable from the modem port of the Macintosh into the DB 25 connector that is wired into the Bridgeport on the other end.

The DB 25 connector had pin 1 as the frame ground, pin 7 as the signal ground, pin 2 as transmit and pin 3 as the receive lines. On the Macintosh modem port, it was found that Pin 1 was the frame ground (also referred to as the chassis ground), pin 3 was the signal ground, pin 5 was the transmit data line and pin 9 was the receive data line. The cable was wired as follows. Pin 1 on the Macintosh to pin 1 on the DB 25 connector, frame ground to frame ground; pin 3 on the Macintosh to pin 7 on the DB25 connector, signal ground to signal ground; pin 5 on the Macintosh to pin 3 on the DB25 connector, transmit data on the Macintosh data to receive data on the DB25 connector; pin 9 on the Macintosh to pin 2 on the DB25 connector; receive data on the Macintosh to transmit data on the connector.

---

3 The source for the Macintosh Modem port connections is the VersaTerm Manual
At this point, the communication problems between the Macintosh and the Bridgeport were solved and the author was ready to move on to the next stage of doing some preliminary research on the post processor, as detailed in the next section.
PRELIMINARY RESEARCH - PARABOLIC REFLECTORS

This chapter deals with writing a program in Macintosh Pascal to have the user input the focal length and diameter of a parabola and have the computer draw out the parabola on the screen. The user then has the option of generating a CNC program to machine the parabola displayed on the screen.

This work is the preliminary research that was done before approaching the more complex task of writing software that would allow the user to input any type of geometry into the computer in graphical form and have the software generate the codes required by the NC machines to machine the part. The Macintosh computer was used because of its more natural and intuitive graphical interface, which allowed the programmer to concentrate on the functional aspects of the software, rather than the detail involved in generating graphics of an acceptable quality.

This program is written in 3 parts. Part 1 deals with user inputs on the parabola parameters, part two deals with generating the parabola on the screen, appropriately scaled, to enable the user to visualize the end product, and if acceptable, to generate a data file which was used to write the codes for the Bridgeport Series I CNC Milling Machine.
PRELIMINARY RESEARCH

- MACHINING PARABOLAS

\[ x^2 = 4 \times \text{Focus} \times z \]
If the user is unfamiliar with G codes and other CNC codes, it is recommended that appendix D be reviewed before reading this section. The appendix gives a description of the codes and the CNC machine programming language.

THE PROGRAM

GENERICS

This program is written in Macintosh Pascal to allow the user to view graphically, different parabola geometries on the screen and then to generate the G codes for the Bridgeport milling machine operating in the BOSS 9 environment.

The first part of the program allows the user to choose the dimensions of the parabola interactively. The user types in the focal length, the diameter of the parabola, and the number of points to be plotted (i.e. the resolution) and the computer produces a graphical display of the parabola. The parabola is then saved to disk as a set of x-y coordinates into a file called Sfile.

In the second part of the program, the user calls up the coordinates from Sfile and uses the least squares approximation to determine the focus and the diameter of the parabola. The parabola is then plotted on the screen and the user is then asked if he wishes to machine this geometry. If
the answer is yes then the user responds to the computer prompts defining machining parameters.

The user is prompted to type in a program name that is no longer the 4 numeric characters. This is in conformance to the Bridgeport format of naming programs. The user is then asked to enter the tool number, the tool diameter, the spindle speed, the feed rate and the maximum blank size. Based on this information and the coordinates stored in Sfile, the program generates a file called Gfile that has the G codes ready to download to the Bridgeport.

SPECIFICS
The variable declarations for the program are performed as usual and the procedure definitions are located above the main program.

The main program begins with a call to the procedure getobject. This procedure is primarily responsible for reading in the coordinates of the parabola into arrays. The next step is a call to the procedure PlotObject. This procedure re-plots the parabola on the screen. The next step is a call to the procedure FindFocus. This procedure uses the least squares technique to find the focus and the diameter of the parabola. At this point, the user is asked if he wants to generate the G codes into a Gfile. A positive reply
PARABOLA MACHINING

- The concept
initiates a call to the procedure 'Post'. Post is discussed elaborately below.

The procedure Post was written to generate the G codes for parabolas of any diameters and focal lengths. The general assumption made by Post is the Z depth is 0 at the top surface of the blank and is increasingly negative as the tool plunges deeper into the part, in keeping with the conventions of CNC machining practice. Thus to machine the parabola, Post relies on the number of points the user has picked to define the parabola. The greater the number of points picked, the higher the resolution of the parabola. The procedure divides the distance D into the same number of divisions as the number of points chosen by the user. Then it defines an imaginary X-Y plane at each of the depths and computes the radius of the parabola at that depth. In this case, the radius is picked from Sfile. Starting at that radius, the procedure generates CNC code to move the tool in a circular path cutting the material to the specified depth. The radius is then repetitively incremented by a fraction of the tool radius and the circular path is generated until the radius becomes equal to the blank radius. This indicates that no more material is to be removed at the given Z depth. The procedure then rapids the tool back to the start radius for the next Z depth and plunges the tool to this new depth and begins the procedure again. The start radius gets
progressively larger as the Z depth increases, reducing the number of circular radial increments required before the blank radius is achieved.

As mentioned before, the more the number of points to plot the parabola, the higher the resolution of the resultant die. A ball end mill is the preferred cutter as this reduces the step like finish of the die and gives the finished part more of a curvature between steps.

The procedure begins with querying the user about the program name, the tool number, tool diameter, spindle speed, feed rate and the blank size. The maximum blank size is then divided by two to convert it to radial dimensions. The plunge rate, or the rate at which the tool is fed into the material in the Z direction is then decided upon as half the feed rate by convention. This is to make sure that the tool does not seize or get damaged by thrusting it into the material due to too rapid a feed. A prompt is output to the screen telling the user that the processing of the Gfile is commencing. The very first character that is written into the file is the ASCII 18 which is the CONTROL R character, the Start of Transmission character in standard RS232 communications protocol. This is the first character that is downloaded into the Bridgeport and it initialize the communication between the Macintosh and the Bridgeport. The second line that is
STRUCTURE OF PROCEDURE POST

\[ Z := Z_1 \]

\[ X := \text{First radius at } Z_1 \]

FOR LOOP := 2 TO N DO \{N is \# of steps for a given z\}

BEGIN

REPEAT

{Counterclockwise Circle(radius \( X \))}

\{X := ToolDiameter/2 + X\}

UNTIL (\( X \geq R_{\text{max}} \) at a given Z depth)

\[ Z := \text{Next Z depth} \]

\[ X := \text{Minimum radius at the Z depth} \]

END; \{End the FOR loop\}
written is the G0G90TnDnSnM26 command line. This instructs the Bridgeport to rapid to the home position, all coordinates in the program are absolute (G90), the tool number is specified, the tool diameter, the spindle speed and the M26 is an instruction to the machine to move to the clear point and await a tool change. The N in the beginning of each line is an N code which is the line number in the program. The N code may be omitted without any loss to the program, but it is useful for doing online editing using the Bridgeport editor.

The main body of Post has two loops. The FOR loop has nested within it, a REPEAT UNTIL loop. The FOR loop runs from 2 to n, where n is the number of points that the user has chosen to define the parabola. The very first point is omitted as this is a radius of 0 at a Z depth of 0. The X, Y, and the Z are set as the first step within the FOR loop and the repeat loop is started thereafter. The FOR loop defines the X-Y plane at a specific Z depth, and the REPEAT UNTIL loop defines the radius of the cut. Since all moves are modal, or are unchanged until reset by a similar instruction, the Z may be set just once for each depth.

The REPEAT loop within the FOR loop begins with a check to see if CNTR, the counter variable is 4. This is done because the center of the part has to be defined to the Bridgeport
only once at the beginning. At N4, the very first G2, or counter clockwise move is encountered. Thus the center of the circular path to be followed has to be defined here. This is done by following the X and Y coordinates by I and J coordinates. The I and J refer to the center of the circular path that is to be generated and the X and Y coordinates denote the start/end point of the tool. The radius of the path is then automatically computed by the milling machine from this information. Any G2 instruction following this first one may be issued without the I, J coordinates and it remains modal from there on. The G2 instruction is continuously issued, incrementing the radius by the radius of the tool, this move being specified by the G1 instruction between the G2, until the maximum blank radius has been reached. It was necessary to include the IF statement to check that the X radius did not exceed the maximum blank radius. This was most likely to happen as the start value of X is determined by the number of points picked by the user, and this may lead to a non standard decimal number. This number, when incremented by the tool radius may almost certainly never equal the maximum blank radius. Thus this check ensures that if the X value has indeed taken a greater value than the blank radius it is reset to be equal to the blank radius value. This will also allow the exit condition of the repeat loop to take effect, without having the tool to go through any extra motions.
On exit from the REPEAT loop, the command to home the tool and turn the spindle off is given (M30). The ASCII characters 4 (ETX, end of transmission) and 14 (DC2, general purpose toggle signal) are appended to the Gfile to terminate communication with the Bridgeport in a normal manner during downloading. The program thus ends execution. A copy of the listing follows the MacCam listing.

The Gfile that is generated as a result of the above program is a pure ASCII file and is downloaded to the Bridgeport for execution through the RS232 port using Red Ryder, a communications package. This process has been explained on the section on communication.
THE SOFTWARE

The successful completion of the Bridgeport - Macintosh interface led to the start of the development of the CAD/CAM software.

As explained in the introduction, this software was written in a way such that small machine shop operators can quickly generate tool paths for their parts. The purpose of this CAD/CAM program is to serve as an uncomplicated, yet powerful tool in the hands of the machine shop operator. Since, the program layout is unique, and the approach is new, this version has been kept simple. It is two and a half dimensional\(^1\).

The user interface was also written to be completely independent of the Macintosh interface. The reason for this was that this being a new concept, once proved to be effective, could very easily be changed into any other programming language or computer system with minimal changes. The use of arrays instead of packed records and lists also is to maintain the system independence of the software. As this program stands, with modifications of the actual graphics drawing commands, the program is capable of executing on any system, including the IBM PC, using Turbo

\(^{1}\) X and Y are the two dimensions, and variations in Z are constant. Z depth may be varied, but without simultaneous variations in X and Y directions.
Pascal. At this stage, customizing the program for the Macintosh's user interface is not a very difficult task.

Initial development of this software was done in Macintosh Pascal. This was because the ease of dealing with an interpreter, rather than a compiler and the instant editing and debug features the interpreter offered, were perfect for the initial development of new software such as this. A few days into programming the software led the author to plan to use the Lightspeed Pascal\textsuperscript{2} compiler for the final version for two major reasons. Reason one is that after the program was designed, it would be much better to present it as a compiled application instead of executing the source code within the interpreter. The second reason is the sheer speed of the compiler once the program is executing. All development of the program continued on the Macintosh Pascal interpreter until quite a late stage into the project, at which time the program became too large and unwieldy for the interpreter to handle. The size of the program would slow down the editing to an unacceptable level and the interpret time before execution seemed close to eternity. The program detailed in this chapter will focus on the Lightspeed Pascal version. This version is similar to the Macintosh Pascal version, but enhanced greatly.

\textsuperscript{2} (C) Think Technologies Inc.
Almost near the end of the project, it was found that using the Lightspeed Pascal compiler would pose a problem. The Macintosh computer does not allow the user to have a code segment larger than 32 Kilobytes. This was not even a consideration at the start of the development as the author did not expect the application code to be that large. At this stage it was necessary to use Lightspeed Pascal's ability to fragment code into smaller pieces that are easy to handle. The way in which this was handled was to separate some of the procedures into separate units. Each of these units was then placed in a different segment. During compilation, the compiler assumes the responsibility of compiling and linking up the different segments provided the segmentation was done correctly and the programs were presented in the proper build order for compilation. This process is described in detail in the section on Compilation.

In this chapter, the author will first explain the generic layout of the program, the style it was written in and then will describe specific routines and their algorithms.

THE GENERIC STRUCTURE

The purpose for writing this software was to create a program that is user friendly, easy to use, and still capable of
generating powerful programs for the Bridgeport. One of the goals was to use the Bridgeport canned cycles in the code generation to simplify the resultant program and to reduce the amount of memory taken up in the Bridgeport's buffer.

The program had to be icon driven, with as little complexity in the usage as possible. The use of the keyboard as an input device was designed to be kept as a minimum. It was decided to make the interface as graphically oriented as possible. Based on the above criterion, the software was written as described below.

This program has been broken down into as many procedures as possible, to avoid repetitive programming by allowing the same procedure to be called from several points with the correct parameters, and to make debugging easy.

The heart of this entire application is a data array. It is this array that stores the information about the user's geometry, tool changes, and all other features in the drawing. This array structure is described in a section of its own, Reference will be made to the data array in this section as the general data array. Please review the section on the array description for detailed specifications.
The entire application is driven by a main program, that is probably the smallest segment of code in the entire application. Execution begins with the main program initializing some basic parameters such as the window size, the different graphic cursor shapes, the initial Z depth plane, the snap, etc. The main procedure then calls the tool description routine within the initialization phase itself so that the user does not forget to describe the first tool he intends to use, before defining the part geometry. The parameters returned by this routine are the feed rate, the spindle speed and the plunge rate which are computed based on the selected tool diameter. This information is then passed on to another procedure that enters this information, along with the tool features into the general data array. At this point the user is asked for the size of his grid, which may range from 0 inches to 2 inches. The option of not having the grid is not presented because of the very nature of the program, that requires precision geometry.

The part origin is the next parameter to be set. The flexibility of defining the part origin or the 0,0 point in the X-Y was found necessary because this program is intended to be a general CAD/CAM program. There are times that the user may want to align the origin to the center of the part, and times where he may wish it to be the lower left corner, or yet again, at some indeterminate point on the part. When
the part origin is selected, the user may choose not to snap
to the grid by selecting the snap value as 0. This feature
was provided since there may be instances that the user wants
to snap to a grid of 0.5", but at a 0.0625" offset from the
physical grid on the screen. If snap was selected then the
user clicks the mouse button close to the position of the
desired grid point and the snap was set with that point as
reference.

At this point the main event loop begins. The main event loop
is where the computer waits for the user to select one of the
menu options. Some of the menu options are, line between two
points, continuous string of lines, cutter compensated, or
uncompensated lines, shallow hole drilling, deep hole
drilling, frame milling, circle milling, etc. The main event
loop looks for a mouse down/mouse up event in the appropriate
area. If the mouse button click is in an illegal or presently
non functional area, it is ignored, putting out a system beep
in most instances.

At the termination of the pseudo geometry generation, the
user selects the G option from the menu. This option is used
to generate the G codes for the existing geometry. On
selection of this option, the program does a complete look
up of the general data array and correspondingly generates
the G codes in Bridgeport format. As mentioned before, this
program makes use of the Bridgeport canned cycles where possible in order to compact the size of code that is being generated and to make the program more readable after generation. Facility to save and load geometry is also provided within the program. The user may choose the front view option where the computer generates a front view of the tool path. In effect, this view shows tool movement in the Z direction. The front view display may be invoked as often as desired in the geometry generation process.

THE LIGHTSPEED PASCAL COMPILER

LIGHTSPEED PASCAL, UNITS, INTERFACES AND MORE...

The following section defines the Lightspeed Pascal Compiler and it's usage. In Lightspeed Pascal, each complete Pascal program is defined in a project. Once into the Lightspeed environment, a project has to be defined before the compiler can be used. For each new application, a new project is opened, and for an existing application, the existing project is loaded from disk. The project can be thought of as a sub environment within the compiler. It is in this sub environment that the user includes all his files. Each project can have only one program. Besides this one main program, each project may have several units. These units may be called from the main program or from within other units.
A typical unit may contain several different procedures or functions. Each unit must be declared in a particular form before it can be used. The main program has all units that are in the project defined in a USES clause at the outset of the program. Procedures in other units may be addressed from within the main program as if they were in the same section and the compiler resolves the calls. An important point to note is that all variable declarations for the main program could be in a separate file. These declarations are then global and other units may USE them. The very first unit declared in the USES clause is this global declaration unit. This global file does not have anything in the implementation section. It is not necessary to define units not directly called by the main program as this is done in the units that directly address the unit in question.

The first declaration within an individual unit is an INTERFACE declaration, which defines other units that procedures in this unit may refer to. Following this, still within the interface declaration, is a list of all the procedures and functions contained within this unit. The procedures and functions are declared at this point with their parameter lists. The actual code is preceded only by the procedure name in the implementation section.
Following the INTERFACE declaration, the next declaration is the IMPLEMENTATION section. It is in this section, all the procedures are defined. These procedures begin with the name of the procedure, but the declaration of the parameters must not be repeated in the implementation section. The IMPLEMENTATION section may consist of as many procedures that the user desires, but is limited by the size of the object code that is generated, not larger than 32K.

THE LIGHTSPEED PASCAL PROJECT

This file is the highest in the hierarchical structure. The main programs and all units are housed in the project. They may be stored in a project by Build order, the order in which each of the units will be compiled or by segment number. A note about Build order, the Lightspeed compiler is critical about the order in which the files are added into the project. Just as in Macintosh Pascal, a procedure may not be called until it is defined, a unit in Lightspeed may not be used until it is previously compiled. The trick to do this is to add those units into the project first, that do not call or depend on any other units. Thus, on compilation of the code, the program compiles them in Build order and at a later stage in the program, a procedure that is called would already be compiled in a unit above it.
The formal structure of a Lightspeed program that uses units is given here.

PROGRAM progame;
   USES unitname1,unitname2,unitname3,.....

(and the procedures and the main program begin from here on)

The formal structure of a unit in a project is given as follows;

UNIT unitname;   {same as declared in the main program}

INTERFACE

USES
   otherunit1,otherunit2,.....

{The above unit names may be the same or different from the main} {program list. This one is normally a subset of the main program} {list}

PROCEDURE InthisUnit1 (param1,param2,param3 : INTEGER;
                        param4,param5 : BOOLEAN;
                        VAR param6,param7 : DTARRAY);
PROCEDURE InthisUnit2 (param1,param2,param3 : REAL;
    param4,param5 : INTEGER;
    VAR param6,param7 : BOOLEAN);

.
.
.

(all other units in this procedure are declared as above)

IMPLEMENTATION

PROCEDURE InThisUnit1;
    CONST
        ConstParam1 = value1;
        ConstParam2 = value2;
        .
        .
        .
    VAR
        LocalParam1,LocalParam2,....: INTEGER;
        LocalParam1,LocalParam2,....: REAL;
        .
        .
        .
    BEGIN
        (The procedure is defined here)
    END;
PROCEDURE InThisUnit2;

CONST

ConstParam1 = value1;
ConstParam2 = value2;
.
.

VAR

LocalParam1, LocalParam2,...: INTEGER;
LocalParam1, LocalParam2,...: REAL;
.
.

BEGIN

(The procedure is defined here)

END;
END.

(This is the end of this unit and it is saved as a separate file called UnitName1)

This concludes the discussion on the Lightspeed compiler's syntax in fragmentizing programs.

SOFTWARE DESCRIPTION
The general outline of this software has been discussed previously. In this section, it is intended to discuss the procedures and the algorithms that make up the software. The data array in which the graphic information is stored is the heart of this program and its description is a logical place to begin the software discussion.

THE DATA ARRAY

This is a universal data array in which information about the image is stored as the user is drawing the geometry. This array is used constantly throughout the program. All information about the part geometry, tool changes, speed and feed, etc., are stored in this array and is later used in the generation of the G codes as well as for saving and loading the geometry to disk.

The array name is DAT and it has been declared in a TYPE declaration in MacCamGlobals\(^3\) as an array of TYPE integer. It is a two dimensional array with 125 rows and 7 columns. The number of rows is arbitrarily designated as 125 in a variable called MAXENTITY. This defines the maximum size of the geometry. It has been deliberately declared as a constant so that it can be modified without any trouble. The 7 column

\(^3\) This is the declarations for the entire program. It is described Later
## STRUCTURE OF THE DATA ARRAY

### GEOMETRY STORAGE

<table>
<thead>
<tr>
<th>DAT</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
</table>
| 1   |    |    |    |    |    |    | G-CODE ...
| 2   |    |    |    |    |    |    | OR   |
| 3   |    |    |    |    |    |    | ENTITY FLAGS |
| 4   |    |    |    |    |    |    |     |
| 5   |    |    |    |    |    |    |     |
| 6   |    |    |    |    |    |    |     |
|     | ...| ...| ...| ...|     |     |     |
| MaxEntity |    |    |    |    |    |    |     |
width of the array was arrived at after careful planning and then some trial and error. The unique approach that this program uses to store geometry is to try and store all the parameters of a particular part of the geometry in the same row in the array. This is advantageous because as soon as the user draws the geometry, the information on that specific entity is stored in a single row in the data array. The very first column of this row is tagged with a code that will later identify the G code to be generated when the appropriate option is selected. For simplicity sake, the code in most instances is the G code number itself. In cases of tool change, etc an arbitrary flag number, usually negative, is assigned to identify the feature without much trouble and confusion.

Thus, an examination of the first column of DAT[MAXENTITY,1] will give an instant idea of the corresponding G code that will be generated for that entity. It may be stated that the first column is always reserved for the entity G code flag, and the remaining six columns of the row in question are defined by the geometry itself and their format will be defined in the appropriate section where the routine for that specific geometry is dealt with. Thus a formal definition of the structure of DAT [MAXENTITY,1] is

CONST
MAXENTITY = 125;

TYPE
dtarray = ARRAY [1..MaxEntity, 1..7] OF INTEGER;

The overall format is as follows:

<table>
<thead>
<tr>
<th>DAT</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G-CODE</td>
<td>...Parameters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ENTITY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FLAG</td>
<td></td>
<td></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>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></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>
<td></td>
</tr>
<tr>
<td>MAXENTITY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

UNIT DEFINITIONS AND EXPLANATIONS

MacCamGlobals

This is the first unit in this project. It has all the global variable declarations that are common to all the units in the project. All declarations would be done as if it were in the
main program itself and will be positioned in the interface section of the unit. There is no implementation in this unit since these are just global declarations, but the implementation and its corresponding END has to appear in the unit for it to be legal to the Lightspeed compiler.

Global variables declared are as follows;

UNIT MacCamGlobals;

INTERFACE

CONST

MaxEntity = 125;

TYPE

dtarray = ARRAY [1..MaxEntity, 1..7] OF INTEGER;

toolinfo = ARRAY [1..24,1..3] OF INTEGER;

VAR

pencil, cross, hand : cursor;
dat : dtarray;
tlno : toolinfo;
rec:rect;
x,y,xzero,yzero,gridsize,zpixel,u,v,feed,spndl,plunge,
tool,snapsizexon,yon : INTEGER;
The main program has several procedures that make up the program. These procedures are stored in different units, to conform to the Macintosh's restriction on the code segment not being larger than 32K.

This unit has several of the initialization procedures and housekeeping procedures called from within the main program. All procedures here have no parameters passed, and have thus been grouped together into MiscellaneousProcs. The only other unit that this unit communicates with is MacCamGlobals. The following discussion lists the procedures in this unit and provides a brief discussion of the structure of each procedure. The procedures in this unit are discussed below.

PROCEDURE INITHAND;
This procedure initializes the hand cursor. The pointing hand cursor is used whenever the user is required to choose an option from a menu. The hexadecimal values defining the graphics image are set, so that when the SETCURSOR procedure is called to change the cursor type, the variable HAND would already be initialized.

PROCEDURE INITCROSS;
This procedure initializes the cross cursor. The cross cursor is used for pointing at precise locations on the screen, such as the corner of a rectangle, or the center of a circle, etc. The hexadecimal values defining the graphics image are set, so that when the SETCURSOR procedure is called to change the cursor type, the variable CROSS would already be initialized.

PROCEDURE INITPENCIL;
This procedure initializes the hour glass cursor. This cursor is used when the computer indicated the user to wait for it to finish the present task. Examples of when this cursor is used is during the load and save routines. The hexadecimal values defining the graphics image are set, so that when the SETCURSOR procedure is called to change the cursor type, the variable PENCIL would already be initialized. The hour glass cursor is displayed during Load/Save operations, and when the computer is doing something other than polling the mouse. It
is named pencil as this variable was originally setup to display a pencil. Drawing with a pencil was more difficult for this application than using the cross hairs. The values were thus modified to display a hour glass instead.

**PROCEDURE CLEARALL;**

This is a simple procedure that is used to clear the entire drawing window. This is required prior to and after the display of the front view of the object, and during the redraw screen calls. The pen pattern is set to white and a PAINTRECT command is given to the entire drawing window. It is used in the initialization routine between questions and in the front view procedure to clear the display to show the front view. At the end of the front view procedure, this is again called to clear the screen before restoring the previous image.

**PROCEDURE ERASEDIALOG;**

Erasedialog is called when some interaction with the user has just terminated in the dialog area. It is designed to clear the dialog area. This is another housekeeping procedure, called from several parts of the program. It is invoked just after some user interaction in the dialog area on the right
side of the drawing window is terminated. It basically draws a white rectangle over the entire area designated the dialog area.

PROCEDURE DRAWScale;

This is the generic procedure that is used during interactive input, to enable the user to select dimensions for the various features supported by the software. It draws the 0 to +4 scale in the dialog box to select tool diameters, circle and fillet radii and the like. This procedure is also only responsible for the actual display of the scale. The calling program assumes responsibility for the prompt that goes with it and then the value that is selected. A FOR - DO loop does the actual scale generation as outlined below.

FOR i := 1 to 32 DO
BEGIN
{Moveto startpoint +i*disp}
{Is this the fourth line?}
(i/8)=ROUND(i/8)}
{IF YES, Draw a quarter inch marker}
{Is this the eighth line?}
(i/8)=ROUND(i/8)}

See user instructions for specific size of the dialog area
{IF YES, Draw an inch marker line}

{IF both above conditions are False,

Draw a regular line}

END;

Every Fourth line is drawn twice as long to indicate half an inch, and every eighth line is drawn even longer to define a full inch marker. The algorithm checks to see if \( (i/4) = \text{ROUND}(i/4) \), where \( i \) is the running variable. This allows every fourth line to be flagged down by the IF statement. Similarly a check for \( (i/8) = \text{ROUND}(i/8) \) flags down every eighth count, for the definition of an inch marker. The variable 'stp' is responsible for the spacing between each of the eighth inch markers.

PROCEDURE DRAWZICON;

This procedure draws the Icon that represent Z depth setting on the main draw window. The reason this procedure is separated from the rest is that this icon is designed to flash when selected. Flashing the icon is basically done by drawing the image with a different font and pen size, so this procedure is called twice in quick succession with different pensize and font settings, with a clear instruction between the calls, producing the effect of a flash.
PROCEDURE DRAWDIALOG;

Drawdialog is the procedure called to draw the dialog boxes in the dialog area whenever any user interaction is desired. This procedure is called frequently from all parts of the program. It is the one that is responsible for drawing the Macintosh like dialog boxes on the right hand side of the drawing window whenever a user interaction is required. The procedure that calls this routine takes responsibility for setting up the titles and the words that go into the dialog box before calling this procedure and also accept responsibility for polling the response. This procedure just draws the dialog box.

This summarizes the procedures in the Unit MiscellaneousProcs.

UNIT MoreProcs;

This is the next unit in the MacCam project. Unit moreprocs has several setup procedures, dialog procedures and the load drawing procedure. This unit communicates with 'miscellaneousProcs', for its house keeping routines and the 'SaveDraw' unit that is responsible for saving a drawing. The SaveDraw is called if the user opts to save current drawing
before loading a new one. The procedures used in this unit and their declarations are as follows.

PROCEDURE RECDRAW (rec : RECT;
    radius,ofset,pnsize : INTEGER;
    recpat : PATTERN);

When the user selects the frame mill option from the menu, the screen drawing of the rectangle is handled by this procedure. This procedure is called by the FrameMill procedure to do the graphical display for the frame to be milled. The size of the frame is passed in the variable 'rec', that consists of the top left and the bottom right coordinates of the rectangle. The variable 'radius' contains the radius of the fillets. If there are no fillets, radius is passed on as zero. 'xl' and 'yl' are two local variables to compute the length and the breadth of the rectangle. This is then used in calls to procedure LINE to draw the actual lengths.

This procedure is called twice for each frame selected, once to represent the actual size of the frame to be machined, and the second time to draw the representation of the tool path on the screen. It is also used to indicate graphically if the
frame is an outside or an inside frame, or a pocket. For example, if an inside pocket was being machined, the pen pattern would be set to gray, and the offset would have a positive value (3 pixels here). Thus, in computing the length and breadth of the rectangle, the size is smaller by 3 pixels all around. This gray 3 pixel short rectangle represents the tool is on the inside of the rectangle. This is followed by another call to the same procedure RecDraw, but the offset is zero and the pen pattern is black. This rectangle is then the original size, representing the size of the finished frame. The same procedure is followed for frame pockets, except the pen pattern is dark gray and the pen size is 2 pixels thick. Likewise, for an outside frame, the offset takes a value of -3 pixels, so the overall length of the frame is increased when drawn with a gray pattern.

The order in which the original pattern and the gray pattern rectangles are drawn in is very important. This is especially so when taking into consideration the fillets.

The fillet drawing routines within the procedure first use a pen pattern of white and positioning themselves on each successive corner of the rectangle, erase a portion of the rectangle drawn equal to the radius of the fillet, in both

---

5 See the users manual for screen representations of inside, outside, pocket mills
the horizontal and vertical directions. This effectively
draws the frame with corners trimmed equivalent to the radius
of the fillet. Then, the variable 'arc' which is of type RECT
is repeatedly redefined to draw the four fillets in the
proper orientation. It may be noted here that the Macintosh
treats 0 degrees as the 12 o'clock position and positive
angles are clockwise, contradictory to the regular convention
of 0 degrees being at 3 o'clock position and positive angles
being counter clockwise.

If the outer rectangle for the frame was drawn first, then
the erasure of the fillet drawing routines on the inner
rectangle would erase part of the outer rectangle's pixels,
resulting in a tarnished image. Thus the order of calling
RecDraw twice successively, is always such that the smaller
rectangle gets drawn first, irrespective of it's pattern or
pen width.

PROCEDURE PlotDot (grx, gry : INTEGER);

This is a simple procedure that is called by the grid drawing
routine to plot a grid on the screen. The two parameters
'grx' and 'gry' define the position at which the desired
point is to be plotted.

PROCEDURE PlotGrid (gridsize : INTEGER);
Plotgrid is a simple procedure that is responsible for the actual plotting of the grid on the screen. This procedure is accessed through the procedure setup. The gridsize selected by the user is passed on as a parameter to this procedure. This grid starts from an X value of 48 so as not to draw the grid in the menu area. The upper bounds for the X and Y values of the grid are 400 and 210 respectively.

There are two Repeat - Until loops nested in one another to produce the grid, the inner one plots the X axis of the grid, for a given value of Y, in increments of gridsize, resets itself and plots the X axis of the gird for the next increment of Y, also incremented by a factor of GridSize. This loop continues until the Y value is greater than or equal to it's bounds. Discussed below is the structure of the REPEAT - UNTIL loops.

BEGIN

(Set X coordinate to upper left of drawing window.)

(Set Y coordinate to lower right of drawing window)

REPEAT

REPEAT

{PLOT x,y}

(Increment X by gridsize)

(Continue this loop...
UNTIL (X limit is reached in the drawing window)
    (Increment to the next Y coordinate)
UNTIL (Y coordinate has reached it's limit)
END; (The grid generation is complete)

PROCEDURE DrawBox (x,y,BoxSize : INTEGER);

This procedure is called repeatedly to draw the boxes around
the Icons in the menu area. The X and Y values give the
coordinates of the top left of the box, and the BoxSize gives
the size of the square box to be drawn. With this
information, simple calls to the line routine are used to
generate the box.

PROCEDURE SetUp (Var x,y,gridsize : INTEGER);

This is one of the main initialization procedures and it sets
up the menus and the icons before the user can begin his
drawing. It receives the values of X,Y as parameters from the
main program, and their values refer to the top left of the
topmost leftmost box in the menu. The start values of X and
Y are 1. It was decided to keep these values as variables so
that the menu may be relocated without any problems
initially. However, as the program progressed through several
stages, the author felt that the spot selected for the menus
initially was the most convenient and relative programming for the rest of the menu was given up. At a later stage though, if so desired, the program may easily be modified to make all menu references relative so that by varying the X and Y parameters, the user may relocate the menu anywhere on the screen.

This procedure begins with setting the BoxSize to 20 pixels, the pen pattern to black and the textsize to 18 for any icons that are represented by a single letter. Another pair of Repeat - Until loops nested are used here to generate the boxes. The outer loop controls the number of boxes in the Y direction and the inner one controls the number of boxes in the X direction. The delimiters are selected as pixel values for ease of programming. For instance, in the X direction, the box starts out with a value of 1 and X is incremented by 22 to define the second box on the same Y value. This leaves a gap of one pixel between boxes since the start value of X was 1 and the box size is 20, this is a total of 21 pixels and the second box begins at 22. The sentinel is set for a value of X greater than 23 and this occurs soon after 2 boxes have been drawing for the same value of Y. X is reset to 1 before Y is incremented to its next value, by a factor of 22. The Sentinel for the Y value is 155 Giving a total of 6 rows of boxes in the Y direction.
Once the boxes are set up, little icons representing the supported functions are drawn in the appropriate boxes. These icons are well documented in the program listing and are self explanatory.

PROCEDURE YesNo (Var Result : BOOLEAN);

Procedure YesNo is a general purpose procedure that is called when a user response of Yes or No is required from any point in the program. Examples of where it is used are if cutter compensation is desired in the line routines, or the user wishes to save a drawing.

The procedure sets the cursor to a pointing hand and calls DrawDialog to draw the dialog box and to 'Fill in the Blanks' to suit the query. Choice is a local boolean variable that defines if a valid choice is made by the user. The mouse is continuously polled until a valid response is obtained, at which point choice, which defaults to FALSE, is set TRUE, irrespective of the selection. Result is another parameter that determines what the choice was, TRUE if yes, or FALSE if no. Choice being set to TRUE stops the polling of the mouse for a response. Additionally if the user selects a point out of the areas drawn by draw box, the Macintosh
system beep is sounded to indicate an invalid choice. At the end of the selection, the cursor is reset to the cross hair cursor, the dialog box is erased and the result of the YesNo prompt is sent back in the parameter 'Result'.

PROCEDURE LoadDrawing (UserName : BOOLEAN;
    Var snap : BOOLEAN;
    Var snapsize, gridsize, feed,
        xzero, yzero, u,
        v : INTEGER;
    Var TlNo : ToolInfo;
    Var Dat : Dtarray);

This is a general purpose program that the software uses to save the main data array and some additional information to disk. There are two possible situations in which load drawing may be called. One is that the user has opted to load a new drawing from the menu, and the other is if the load drawing procedure is being accessed by the program itself for a redraw, or a redraw after the front view has been displayed.

The major difference in the two types of calls is that if it is a user call then the user supplies the filename that is to be retrieved and is prompted if he wishes to save the current drawing. The prompt to save the current drawing
appears only if the index variable $U^6$ has a value greater than 1. This is done because a prompt of "Save drawing?", is redundant when there is nothing drawn on the screen, just after a boot up for instance. The save drawing question is also supported by the YesNo procedure defined above.

Now if the call to LoadDrawing was user invoked, then the program begins with a call to the SaveAll procedure in the SaveDraw unit described later, and the user is prompted for the name of the file to be loaded.

If the call to LoadDrawing was system invoked, then the above process is bypassed and the load filename is automatically set to 'temp'. This is because, whether the system invokes this procedure after a front view display or after a screen refresh request, the first thing either of those procedures does is to make a copy of the current drawing under the name temp, and then proceeds to manipulate the display.

Once the filename is determined, irrespective of it's origin, the procedure clears the drawing window, and indicates that it is processing the load instruction. The cursor is set to the hourglass shape. All the fixed parameters are loaded into

---

$^6$ U is the number of entity counter
the memory first. The array 'dat' and the array 'tlno' are the next to be replenished with information from the disk file. Once the load is complete, the screen is cleared by procedure clearall, and procedure SetUp is called to redraw the menu. 'Xhole' and 'Yhole' are the temporary variables used to define the start position of the first box in SetUp. This is to avoid additional local declarations, and optimize the variable use in the procedures.

From this point on, there is a complex compound IF statement that looks at the first column entry of the array 'dat' for successive row values as generated by the running index variable 'i'. Depending on the first value, the program proceeds to retrieve other parameters from other columns in the same row, and regenerates the image. The structure of the 'dat' array is described in the relevant drawing procedure and is not repeated here to avoid repetition. The drawing algorithms are also very similar to the original drawing algorithms and are thus not discussed here.

UNIT SaveDraw;

7 For a definition of the fixed parameters and the structure of the datafile, refer to the description of the SaveAll procedure in Unit SaveDraw.
This unit primarily houses the save routine for this project. The procedure used for saving the files is called 'SaveAll'. This unit needs to communicate only with MacCamGlobals for reading in the global values. It does not call or access any other unit or procedure. The procedure declaration is as follows:

```
PROCEDURE SaveAll (UserName : BOOLEAN;
    Var snap : BOOLEAN;
    Var snapsize, gridsize, feed, xzero, yzero, u, v : INTEGER;
    Var T1No : TOOLINFO;
    Var Dat : DTARRAY);
```

Local declarations within this procedure are the variables 'DataFile' of type text, the datafile in which the geometry is to be stored, 'FileName' of type string, the variable used to store the data file name, and 'i', 'j', local running index variables.

The procedure begins with a test of the boolean variable 'UserName'. If this variable is TRUE, then the user is prompted for a filename under which he wishes to save his drawing. If 'UserName' is FALSE then the program interprets this to be a system call and assigns the variable filename
with the name 'temp'. The cursor is set to the hour glass shape to indicate that the system is busy.

The format of the datafile is as follows; the very first part of the data file consists of housekeeping parameters such as snap, snapsize, zero positions, and the like. This is followed by a dump of the contents of the array 'dat' and then of the array 'tlno'. The exact format in the order it is saved is

SNAP : Boolean variable signifying the status of snap in this drawing, TRUE indicates that snap was on, and FALSE indicates that was off
SNAPSIZE : Integer variable indicating the size of the snap if it was on
GRIDSIZE : Integer Variable indicating the grid size in pixels
FEED : The feed rate of the machine in inches per minute
XZERO : The X coordinate of the part origin
YZERO : The Y coordinate of the part origin
U : Running index for 'dat' array
V : Running index for 'tlno' array
DAT : Array storing the actual geometry of the part and information of the tool number,
tool diameter and spindle speeds. This array is written to the disk file with the help of two index variables. The first variable 'i' is the row counter in the array and the second index variable 'j' is the column counter. 'i' runs from 1 through 'u', since u has the number of entries made into the drawing and 'j' runs from 1 through 7, in accordance with the dimension of the array itself.

This data array keeps track of the tool information of each tool change. It has a dimension of 24 rows and 3 columns. The 24 rows are a direct consequence of the fact that the Bridgeport machine supports 24 different tools in it's tool chart. The running variable 'v' keeps track of how many of the tools have been used in the particular drawing. The first column keeps track of the tool number corresponding to the Bridgeport, the second column stores the tool diameter. The tool diameter is multiplied by 10000 to make it an integer value for storage in an integer array, so that it would be convenient to store all information about
a particular tool in one array. The value of the tool diameter, when retrieved, is again divided by 10000 to yield the tool diameter. The third column stores the spindle speed in rpm.

This concludes the definition of procedure SaveAll and the unit SaveDraw.

UNIT FrontView;

Unit front view is a self contained unit containing all the procedures that are exclusively used by the front view generating routines. This unit is unique in the manner in which it generates the front view of the tool paths of the part to be machined. It must be remembered that the very concept of this program is to eliminate the separate generation of the tool path and the geometry. Thus the front view is also in reality, pseudo geometry, almost the tool path. The unit interfaces to several other units during it's procedure calls. The units that are declared in this procedure are as follows:

MacCamGlobals, MiscellaneousProcs, SaveDraw, MoreProcs
The reason for calls to the above units will become apparent in the following procedure discussion. MacCamGlobals is, as always, the global variable declarator.

PROCEDURE FrView (Snap : BOOLEAN;
    snapSize, GridSize, Feed, Xzero,
    Yzero, u, v, plunge : INTEGER;
    Dat : DTARRAY;
    TlNo : ToolInfo);

This unit is one of the highlights of this program. It is also the most important procedure in this unit. All other procedures in this unit are designed as support procedures to this procedure. A lot of thought had to go into the development of this procedure. This front view generator would have been simple to write in a specific program or in a program where the geometry is bound by predetermined specifications. The matter is complicated further by the fact that the user may choose to begin his drawing anywhere in the drawing window, and also make his drawing of any size in the window. To further complicate matters, the user is free to choose the zero origin of his part. It need not be in the center of the part or the lower left, but could be anywhere, inside or outside the part.
The approach that was followed to overcome the above hurdles is as follows. The frontview algorithm searches through the entire database for the maximum and minimum bounds of the X and Y values for any entity in the drawing, irrespective of its shape or function. In the event that this entity is a circular object, only the center point and the radius is stored in the array. In this case, the algorithm frames the circle in a rectangle and checks for the maximum bounds. The boolean variable 'FirstVal' flags down the very first values of x,y,z and assigns them to the minimum and maximum storage variables, 'minx', 'maxx', 'miny', 'maxy', 'minz', 'maxz'. All comparison of the subsequent variables is done with respect to this initial setting. This was necessary since the minimum values of each of the variables was being sought. An arbitrary start value of 0 would have caused an erroneous result if negative values existed in the search. This is particularly true for Z values, which are negative into the part.

At the conclusion of the above search, an imaginary rectangle is defined, enclosing all the entities in the drawing. The top left of this rectangle is (minx,miny) and the bottom right is (maxx,maxy). Now these parameters are used to determine the scale factor for the top view in the front view display algorithm. The X and the Y scale factors are computed in such a manner that the entire imaginary rectangle is scaled to fit a maximum X value of three hundred and ninety,
and a minimum Y value of eighteen\(^8\). A particular point is arbitrarily picked on the screen to be the minimum X value point and the maximum Y value point of the top view. The entire object is scaled and translated with respect to this point so that the location of the top view in the front view algorithm is always determinate. This allows the program better control of the drawing window on the screen. In this program the two variables determining this point are 'alwaysminx' and 'alwaysmaxy'. They have been declared as constants and have been assigned the values of 100, and 145 respectively. These values have been chosen very carefully so that the entire top view occupies a little more than the top half of the drawing window, leaving space for the front view in the bottom half. Another variable 'alwayszeroz' is set up to represent \(Z = 0\) in the front view. This position is also always determinate and the rest of the front view is drawn relative to this point. The value of this variable is 155, but as a rule of the thumb, it is alwaysmaxy + 10 to ensure a 10 pixel separation between the top and the front views. Thus the drawing top view will always be scaled to fit the screen and translated to have it's lower left corner on the position (alwaysminx,alwaysmaxy) and the front view will always have the \(Z = 0\) plane on the Macintosh's Y coordinate of 155, which is the variable 'alwayszeroz'.

---

\(^8\) 390 and 18 are the two numeric values that exist in the computation of xscale and yscale.
The maximum X length of the part is easily computed from

\[ x_{\text{length}} = \text{max}x - \text{min}x \]

and the Y length is also computed from the maximum and minimum values as

\[ y_{\text{length}} = \text{max}y - \text{min}y \]

The X axis scale factor is then computed as

\[ x_{\text{scale}} = \frac{390 - \text{alwaysmin}x}{x_{\text{length}}} \]

and the Y axis scale factor is also computed as

\[ y_{\text{scale}} = \frac{\text{alwaysmax}y - 18}{y_{\text{length}}} \]

The lesser value of the two scales computed is the overall scale factor of the drawing and the entire drawing is scaled up or down to this value, depending on whether 'scale' is > 1 or 'scale' < 1.

At this point, the object is scaled to the right size, but is still not in the right position on the screen. The translation to the point 'alwaysminx', 'alwaysmaxy' has still
not been performed. To compute this translation factor, the extreme values of X and Y that were found in the previous algorithm are first scaled. This is achieved by multiplying the \texttt{'minx'}, \texttt{'maxx'}, \texttt{'miny'}, \texttt{'maxy'}, \texttt{'minz'}, \texttt{'maxz'} values with the computed scale factor. This gives the extreme bounds of the scaled image. The scaled values are rounded off because the values represent pixels that cannot have real values. The X axis translation factor is then computed as

\[ x_{\text{trans}} = \text{alwaysminx} - s_{\text{minx}} \]

where
- \( x_{\text{trans}} \) : X axis translation value
- \( \text{alwaysminx} \) : Always the minimum X value point
- \( s_{\text{minx}} \) : The scaled minimum X position of the image

Similarly for the Y axis translation factor

\[ y_{\text{trans}} = \text{alwaysmaxy} - s_{\text{maxy}} \]

where
- \( y_{\text{trans}} \) : Y axis translation value
- \( \text{alwaysmaxy} \) : Always the maximum Y value point
- \( s_{\text{maxy}} \) : The scaled maximum Y position of the image
The signs for the translation are automatically taken care of when the above translation values are computed depending on whether they are negative or positive. The values of Xzero and Yzero are not important at this point because the program does not scale the image in inches, but does so in a pixel manipulation manner that does not require the initial position to be manipulated. At a later stage, the same routine may be modified to take into consideration the Xzero and Yzero points to allow the user the flexibility of scaling up or scaling down his creation (drawing). This has not been implemented in the current version in the interests of time.

From this point, the drawing of the image is routine. The top view is drawn by polling the first column of the array 'dat' and determining what kind of a function it is. Based on this, the corresponding parameters are extracted from other columns and the image is drawn just as explained in the respective drawing algorithms, with one difference.

Each coordinate in each axis is transformed based on the following transformation.

\[ \text{param1} \times \text{scale} + \text{trans} \]
for example, to translate a line from x1,y1 to x2,y2 the parameters passed to the line routine are

\[
\begin{align*}
\text{NewX1} &= x1 \times \text{scale} + \text{xtrans;} \\
\text{NewY1} &= y1 \times \text{scale} + \text{ytrans;} \\
\text{NewX2} &= x2 \times \text{scale} + \text{xtrans;} \\
\text{NewY2} &= y2 \times \text{scale} + \text{ytrans;}
\end{align*}
\]

MoveTo (NewX1,NewY1);
LineTo (NewX2,NewY2);

In the interpretation of the G0 and the G1 routine, the program looks for a G0. It memorizes this point as the beginning of a linear move in the variables 'previoustopx' and 'previoustopy'. The next G1 that the program interprets, it draws a line from the translated x,y coordinates associated with the G1 to the values previously memorized for the G0 move. As soon as it draws the line it memorizes the new x,y values associated with the G1 for the x1,y1 of the next segment, so that the line string may be generated with ease. For the front view, the Y coordinate is ignored. The zdepth is read from the array and is added to the 'alwayszeroz' value to give the displacement from the plane
of zero depth. A negative sign on Z takes values of Z in the positive direction into consideration. A similar procedure as explained above is followed to draw the other functions to scale on the screen. It may be noted here that all circles and holes are treated as rectangles for graphical drawing purposes since the Macintosh Pascal and the Lightspeed compiler use a variable of type RECT to specify arcs. This makes scaling and translation of the circles and holes a very easy task.

The user has the option to step through the front view generation. This is a very useful option since the algorithm draws an entity in the top view and then in the front view, with a digital read out of the actual Z depth of that entity. The user may then click the button to continue and the computer updates the top and the front view with the next entity, and pauses for the user to click the button again. This presents a nice visual effect and a better understanding of the information the program is trying to present. The question whether the user would like to 'Step - Step' through the drawing is handled by procedure YesNo. If the user wishes the step mode to be active, then the algorithm calls the

---

9 Negative z depths are stored as positive pixel values for ease of interpretation, the Unit post is responsible for converting them to negative values for the Bridgeport.
procedure PressAKey after each successive entity is drawn. PressAkey is explained below. At the end of the complete picture generation, the algorithm calls the procedure Done, to display that the algorithm has finished it's display, and the user may press any key to continue. This ends the definition of the FrontView Algorithm.

PROCEDURE HangAround;

This is a simple parameterless procedure that is called during the front view generation in the step step mode to wait for the user to click the mouse button. It does not matter where the mouse button is clicked as long as one complete mouse down, mouse up sequence is executed. It is also called by the procedure Done, to wait for user acknowledgement before returning control to the main program at the termination of the front view routines.

PROCEDURE PressAKey;

This procedure is called upon when the user wishes to step through the generation of the front view. It is this procedure that displays the Z depth on the screen in the step mode, and it is also this procedure that displays the 'Click Button to Continue' prompt before calling Procedure HangAround to wait for the user's response. In the event the
user does not wish the Step Step mode, this procedure is completely bypassed.

PROCEDURE Done;

This is the last procedure in this unit. It is called at the end of the front view generation irrespective of whether the user has opted for the Step - Step mode or not. It also calls on the procedure HangAround to wait for the user's response before returning to the main program.

NOTE:

It is the main program's responsibility to save the drawing as it currently exists into a file called temp and to clear the screen before the front view procedure is called. Likewise, the calling program also accepts housekeeping responsibility after front view has terminated execution by clearing the drawing Window and calling LoadDrawing to regenerate the drawing using the file temp, just as it existed before the Front view routine was called.

UNIT BridgePort;

This unit contains the post processor procedure for the Bridgeport. The procedure post is the only procedure for this
unit. This unit only uses MacCamGlobals. It is completely independent apart from the global declarations.

PROCEDURE Post (Zpixel, Xzero, Yzero, u, v, feed, plunge : INTEGER;
                  dat : DTARRAY;
                  TlNo: TOOLINFO);

Procedure post again depends on the array Dat, Tlno, and the running indices, u, and v, to generate the G codes. Several local variables are used in 'Post' because data in the arrays is stored in as compact a form as possible. Thus the conversion from pixels to inches, positive to negative in the case of Z depths, conversion of tool diameters to real decimal values and the like are performed within 'Post'. Frame milling G codes, for instance require the center of the rectangle and the unsigned lengths in the X and the Y directions to generate the tool paths. Data in the array is stored as the top left and bottom right corner of the rectangle for compactness. This conversion is also made by 'Post'.

'Post' begins with a constant declarator for the colon. This is to be appended to the name of the Bridgeport program that results out of post's execution. The first prompt in this procedure is the name for the Bridgeport program. The rules
of the Bridgeport say that this should be no more than 4 characters long and should be numeric. The name should be preceded by a colon and this is automatically done by the program. The procedure then resets a file called G file, to which the output will be sent. Any previous GFile that exists will be erased when post begins it's execution. If the user wished to save any previous Gfile, then he should have renamed it at the system level before starting this application. This is really not a problem if the user has saved the geometry from within the application, since the file may be retrieved and the post processor will regenerate the G codes in under a minute.

The First character that gets written into Gfile is the ASCII 18 character, or the Control R character. The reason for this is explained in the Interfacing section. This is followed by the writing of a colon and the users filename for the Bridgeport. At this point the post processor begins processing of the data arrays to generate the G codes.

The entire process is controlled by a FOR - DO loop from 1 to U times, U being the running index counting the number of entries into the data arrays. The first column of the dat array is polled for the G code type (DAT[i,1]). Based on the code in the first column, the processing of the G file begins. 'Cntr' is a variable that is incremented every time
something is written to Gfile, so that the N codes may be
generated correctly. All writes to Gfile may be assumed to
have a

\[ Cntr = Cntr + 1; \]

statement before the actual write unless otherwise specified,
and this will be omitted in the following discussion for
clarity.

\[ \text{IF dat}[i,1] = 0 \text{ or } \text{dat } [i,1] = 1 \text{ THEN} \]

This indicates that the user intended to perform a moveto a
point or a linear move from some point. The first step here
is to convert the x,y,z values in columns 2,3,4 respectively
into inches. Note in the program listing, the sign of the Y
is reversed since the Macintosh has Y = 0 on the top of the
screen, Y becoming increasingly positive as the user moves
down the screen, contradictory to the way in which the
program was written with Y = 0 at the bottom of the screen
and Y becoming increasingly positive as the user moves up the
screen, as in the cartesian coordinate system. The format of
the G0 code is

\[ \text{GOX}___Y___Z____; \]
At this point, the decision tree is broken down even further. If the code was a G0, then a G0 to the specified X and Y coordinate is performed, 0.05 inches above the part surface. This is to prepare the tool to plunge into the part. The very next code generated automatically is a G1, to plunge into the Z depth desired, as retrieved from column 4 of the G0 row in dat. This is done at the plunge rate that is designated by the program to be half that of the feed rate. The format of the G1 code is

\[
\text{G1X}_Y \text{Z}_F
\]

If the code was a G1, then a G1 is written to the file at the specified X, Y, Z values, at the specified feed rate.

\[
\text{IF dat[i,1] = 175 or dat [i,1] = 176 or dat[i,1] = 177 THEN}
\]

This signifies a circle mill routine. The very first task in this routine is to retrieve the x,y coordinates of the circle center, the radius of the circle and the desired Z depth.

At this point, if the code is a G 175 or a G 176, the same line is generated since all parameters are identical, except for the G code number. This is retrieved from the data array to maintain generality. A G 175 represents an outside circle
mill, and a G 176 represents an inside circle mill. The format of this instruction is

G175(176)X__Y__R__Z__Z__P__P__F__F__F__;

where,

- **x,y**: center of the circle
- **R**: Radius of the circle
- **Z**: First Z is the full depth of the mill
- **Z**: Second Z is the step depth of the mill
- **P**: First P is the tool entry clearance\(^{10}\)
- **P**: Second P is the finish allowance
- **F**: First F is the mill feed
- **F**: Second F is the finish feed
- **F**: Third F is the Plunge feed

If the code was a G 176, representing a pocket, then using the same values of x,y,r,z, code is written to Gfile in the following format.

G177X__Y__R__Z__Z__P__P__F__F__F__;

where,

- **x,y**: center of the circle

\(^{10}\) Note: The sum of the entry clearance and the finish allowance must be less than the circle radius
R  :  Radius of the circle
Z  :  First Z is the full depth of the mill
Z  :  Second Z is the step depth of the mill
P  :  First P is the step over, the amount the tool
     will step over when machining the pocket
P  :  Second P is the tool entry clearance
P  :  Third P is the finish allowance
F  :  First F is the mill feed
F  :  Second F is the finish feed
F  :  Third F is the Plunge feed

IF dat[i,1] = -10 THEN

This is a code invented by the author to flag a tool change
within post. Besides the 'cntr' being updated here, 'tcnt'
is also updated to signify an increment in the array T1No.
The format of the instruction generated based on the tool
change code is

G0G90T___D___S___M26;

Where,

G0G90:  is a rapid absolute move to the clear point
for a tool change.
T  :  is the tool number corresponding to the
    Bridgeport's tool chart
D : is the diameter of the tool
S : is the spindle speed
M26 : is the miscellaneous code for a move to clear point for tool change.

IF dat[i,1] = 170 or dat [i,1] = 171 or dat[i,1] = 172 THEN

This indicates a frame mill or a rectangular pocket. Irrespective of the code number, the first part of this algorithm is to compute the center of the designated frame, and its unsigned lengths in the X and the Y directions. All the values must also be converted into inches as discussed for the other procedures. 'Xc,Yc' are the computed centers of the frame and 'Xl,Yl' are the unsigned lengths in the X and the Y direction. The 170 or 171 codes are identical in format except for the G code number. G170 signifies an outside frame and a G171 signifies an inside frame. The format of the command is

\[ G170(171)X____Y____Z____X____Y____R____Z____Z____ \]

\[ P____F____P____F____F____; \]

where

\[ X,Y : \] the center of the frame to be milled
\[ Z : \] is the clearance plane
\[ X,Y : \] is the unsigned length and width of the frame
If a G172 is generated, this indicates a rectangular pocket.
In this case the following code is generated.

```gcode
G170(171)X___Y___Z___X___Y___R___Z___Z___
P___F___P___P___F___F___;
```

where

- **X,Y**: the center of the frame to be milled
- **Z**: is the clearance plane
- **X,Y**: is the unsigned length and width of the frame
- **R**: Radius of the fillet (0 if no fillets)
- **Z**: First Z is the full depth of the frame
- **Z**: Second Z is the step depth of the frame
- **P**: First P is the step over for each successive frame
- **P**: Second P is the initial entry clearance
- **P**: Third P is the finish allowance
F : First F is the mill feed
F : Second F is the finish feed
F : Third F is the plunge feed

This concludes the discussion on the frame mill post processor routines.

IF dat[i,1] = 81 or dat [i,1] = 83 THEN

This indicates hole drilling. G 81 is a hole drilling canned cycle and G 83 is a deep hole peck drilling cycle. If the drilling depth is specified as 0.5 or greater then the program automatically selects the peck drilling cycle. Irrespective of the code, the x,y coordinates are converted into inches, and the Z depth is also converted into inches. The format of the G 81 code is as follows;

G81X____Y____Z____F____;

where
X,Y : are the coordinates of the holes to be drilled
Z : is the Z depth of the hole
F : is the feed rate used to drill the hole

In the case of a G83 command for deep hole drilling, the format is
where

- **X, Y** : are the coordinates of the deep hole
- **Z** : First Z is the total Z depth to be drilled
- **Z** : Second Z is the first peck depth
- **Z** : Third Z is the subsequent peck depths

This concludes the discussion of the drilling cycles.

**IF dat[i,1] = 40 or dat[i,1] = 41 or dat [i,1] = 42 THEN**

This indicates that the user has opted for cutter compensation to be turned on or off. If the code is a 41 or 42, then this is the cutter compensation turn-on point. The corresponding x,y coordinates are read off from the array and the G code is generated. If the code was a 41 or a 42, then the format is

\[ G0G41(42)X_{---}Y_{---}Z_{---}; \]

where

- **X, Y** : is the cutter compensation turn-on point
- **Z** : is the depth it is turned on at.
A G 41 constitutes outside compensation and a G 42 constitutes inside compensation. A corresponding G 40 turns off cutter compensation and it has the exact same format as shown above.

END IF.

At the termination of the generation of G codes for all the entities, a G0 to X0 Y0 Z1.0 is written to move the tool above the part. An M30 instruction is also appended as is conventional at the end of a NC program. This instruction resets the tool to a clear point and rewinds the program for future use. The very last line written to the Gfile is the ASCII characters 4 and 14, the Control D, Control T characters. This is to allow smooth termination of communication with the Bridgeport. Refer the section on the Interfacing of the Macintosh and the Bridgeport for an explanation of these codes.

PROGRAM MACCAMLP1;

USES MacCamGlobals, MiscellaneousProcs, MoreProcs, SaveDraw, FrontView, BridgePort
This is the one and only main program within the project. As mentioned before, the initial development of this program was done in Macintosh Pascal before the program was fragmentized into smaller segments. This is the reason that several procedures are still in the main program unit. They may be relocated with ease, but the author chose to leave them alone for convenience. The main program body will be discussed at the end of this section after a discussion of all the procedures that are accommodated in this unit.

PROCEDURE SHOWXY (x,y : REAL);

This is a simple procedure that is used by several of the mouse tracking locations within the program. It receives an x,y real value that has been converted into inches with respect to the xzero, yzero origin the user has previously picked. This value is displayed constantly, every time the mouse is moved, at the bottom right side of the screen. It provides a means of instantly tracking the position of the mouse. It uses a PaintRect call to erase the area before updating the position of the mouse, and this causes a flickering effect on the screen. This is unavoidable without resorting to some Macintosh operating system calls and this has deliberately avoided to maintain the transportability of the program. In the case of certain selections, where the flickering is unacceptable, the update of the mouse position
is provided only if the user holds down the mouse button and then changes the mouse position. This enables him to see the update only when desired and not lose the flexibility of having a concrete readout of the current position of his mouse.

PROCEDURE SCALESELECT ( Var PixelVal : INTEGER);

This procedure is called from several parts of the program when the user is expected to select a dimension. Some examples of calling this procedure are to select the diameter of the tool, to select the radius of the circle mill, or to indicate the depth to which a hole is to be drilled. This routine returns to the calling program a single pixel value that represents the selected dimension. This value may easily be converted into inches by dividing it by 32, which is this program's reference pixel number for an inch. The routine DrawScale is invoked from within this routine to draw the scale in the dialog area. The user is limited to clicking within a reasonable distance from the scale. A click outside this region is answered with a system beep. The ShowXY procedure is passed a 0 value for X and a Z value for it's Y display, to enable it to display Z depths in a digital form.

PROCEDURE TOOLCHANGE ( Var dat : DTARRAY;
Var tlno : TOOLINFO;
Var u,v,tool,spndl : INTEGER;
Var ToolDia : REAL);

This procedure is called at initialization to set the start tool number and its associated parameters. It can also be called from within the program, where an icon exists to invoke the tool change routine. On the invocation of the drill hole routine, the user is once again given a chance to call this routine, as the diameter of this tool will determine the diameter of the hole that will be drilled.

The first thing the user selects is the tool number. This is done by clicking in the appropriate box out of the 24 boxes drawn in the dialog area, representing the maximum number of tools possible in the Bridgeport tool chart.

Once the appropriate tool number is selected, the user is prompted to select the diameter of the tool. This is done with the help of the ScaleSelect procedure defined above. The tool diameter is multiplied by 10000 to store it as an integer in the array tlno, and it is later converted to a real value by the procedure post.

PROCEDURE GridSet ( Var GridSize : INTEGER);
This procedure is invoked at the beginning of the program to set the gridsize. It is responsible for the drawing of the dialog boxes and the prompt since the box is not in the regular dialog area. This routine even has its own version of scaleselect built in, in a horizontal manner for a flavor of difference. This was one of the very early procedures that was developed, and at that time, the existence and importance of a separate ScaleSelect routine was not felt. Even the so called dialog area was not definitely assigned at this stage, explaining the non standard placement of this procedure's prompt. The main return from this procedure is the value of the variable 'GridSize' that is used in many areas within the entire program.

PROCEDURE WindowInit;

This is a parameterless procedure that is called at the start of execution of the program. Unlike the Macintosh Pascal interpreter, the Lightspeed compiler does not store the attributes of the windows along with the file. The windows are to be explicitly defined. The text window is defined first and is made to appear on the lower part of the screen. The values are stored in a RECT TYPE variable and the top, left and bottom right coordinates of the text window is set as (2,288) for the top left and (512,342) for the bottom right.
Likewise for the drawing window, the top left corner was set in a RECT TYPE variable as (2,40) and the bottom right was set at (512,290). There is a slight overlap of the drawing window on the text window, and this is intentional, to gain a few pixels in the drawing window. The user does not want to see TEXT as the title in the text window, and this is overlapped by the drawing window, increasing it's size by several pixels. A call to the procedure HideAll ensures that no windows are open on the display, and the drawing and text windows are selectively opened with calls to SetTextRect, and SetDrawingRect. This is followed by calls to ShowText and ShowDrawing to display the windows.

PROCEDURE Zset (zpixel: INTEGER);

This is the routine that is accessible through the icons to change the reference Zdepth at any point in the program. The user must set this Zdepth before calling the linedraw routine, for example. The reason ScaleSelect was not used here is that this option had to allow for positive values to move the quill of the Bridgeport up, and for negative values to move the quill down, or plunge the cutter into the part. This scale is oriented just as it would be on the Bridgeport, with positive values being upwards, and negative values being downward.
PROCEDURE ToolSet ( Var feed, spndl, Plunge : INTEGER);

This procedure is called once at the beginning of a new part right after the tool selection to inform the computer as to which material is being machined. This information is used to extract from its basic database, mill feeds and spindle speeds as functions of the selected tool diameter. This procedure has been written in a general fashion, with the user having the option of picking his own material, if a suitable one on the list does not exist. In doing so, the user assumes complete responsibility for determining the correct cutting speeds and chip loads for that material. The variable TotalMat is incremented by one to include this new material in its database, and on the next time, the material is displayed as if it were one of the original choices.

This procedure uses two disk files for its operation. They are 'MatDataFile' and 'MatFile'. Matfile contains a list of the materials information is available for, and this information is transferred into an array called 'Material' for ease of manipulation. MatDataFile contains the corresponding cutting speeds in surface feet per minute and the chip load for the corresponding material in 'MatFile'. This information is also transferred into an array called 'tool' at the invocation of this procedure.
The actual spindle speed in rpm is a function of several parameters. Examining metal cutting from a micro point of view, the cutting edge of the tool actually shears of the metal at the point of contact. Based on several factors, there is an optimum speed at which the metal is sheared for each material. This is the cutting speed in surface feet per minute. The spindle speed is a function of this cutting speed. The relation of this spindle speed with the cutting speed is given by

$$SS = \frac{SFM \times 4}{D}$$

where

- $SS$ = Spindle Speed in rpm
- $SFM$ = Cutting speed in surface feet per minute
- $D$ = Cutter diameter in inches

Some typical cutting speeds for different materials is given in the table below.

<table>
<thead>
<tr>
<th>MATERIAL</th>
<th>CUTTING SPEED (SFM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aluminum</td>
<td>300</td>
</tr>
<tr>
<td>Carbon Steel</td>
<td>100</td>
</tr>
<tr>
<td>Brass</td>
<td>200</td>
</tr>
<tr>
<td>Cast Iron</td>
<td>75</td>
</tr>
</tbody>
</table>
The feed rate is a function of the chip load. The chip load is related to the amount of material removed per cut per cutting tooth. The relationship is defined as

\[ F = CL \times NT \times RPM \]

Where

- \( F \) = Feed rate in inches per minute
- \( CL \) = Chip load
- \( NT \) = number of teeth
- \( RPM \) = the spindle speed in rpm

Chip loads for some materials are listed in the table below.

<table>
<thead>
<tr>
<th>MATERIAL</th>
<th>CHIP LOAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aluminum</td>
<td>0.004</td>
</tr>
<tr>
<td>Steel</td>
<td>0.003</td>
</tr>
<tr>
<td>Wax</td>
<td>0.004</td>
</tr>
</tbody>
</table>

After having initialized the basic arrays to be used, the program sets up the screen with the material name and it's parameters alongside. On the left of each material there is

\[ 11 \] This is an approximation. No data was available for machine wax
a small check box drawn for the user to click his choice. The last check box drawn is to allow the user to pick his own material, invoking the special part of the program that re-establishes the 'Matfile' and 'MatDataFile'.

The algorithm to check for a valid check box click was also written in a general manner to allow the incorporation of new choices without user intervention or programming changes. Thus when the user adds a new material to the list, there is a new check box added to display that material and the polling routine also polls for that new check box. This is done as a function of totalmat, the variable keeping track of the number of materials in the datafile.

Based on the user's selection of the material, the program designates values for the variables 'spndl', 'feed', 'plunge' from within its database for use in the rest of the program. 'spndl' is the spindle speed as calculated, 'feed' is the computed feed rate, and 'plunge' is the plunge rate which is half the feed rate as a rule of the thumb. If the user has more than 8 materials on the current list, then some materials must be purged before the system allows the addition of new materials. This is done by direct editing of the ASCII files, Matfile and MatDataFile using any ASCII editor.
PROCEDURE Mtrack (Snap : BOOLEAN;
SnapSize, Xzero, Yzero : INTEGER;
var X, Y : INTEGER);

This is a general purpose mouse tracking procedure. At many instances within the program, the user clicks the mouse at a desired X, Y coordinate. Instead of having individual routines to do essentially the same thing, this common procedure is accessed. The Snap, and Snapsize variables are used by the routine to determine if snap is to be used while tracking the mouse. Xzero, and Yzero are passed so that Mtrack may call the procedure ShowXY, to display it's current coordinates in inches. The final selection by the user is returned to the calling routine in the variables X, and Y. The heart of the tracking routine is two Repeat - Until Loops. The first one detects a mouse down and the second one detects a mouse up event.

PROCEDURE FrDecision (Var Pocket, Inside : BOOLEAN);

This procedure is invoked from within the frame mill and circle mill routines to determine the user's wishes. There are two variables that are returned to the calling procedure. Pocket having a values of TRUE indicates that the user would like to machine a pocket, and the result of inside is inconsequential. If Pocket is FALSE, this indicates that the
user wants to do a frame mill, or a circle mill. If inside is true, this indicates that the user is interested in machining an inside frame or circle. Likewise, a FALSE value for inside and pocket indicates the user's desire for an outside frame or circle. This procedure calls DrawDialog after setting up the choices and the titles in the dialog area. It has a variable called choice, to flag when a complete legal choice is made so that the procedure may stop polling the mouse and return to the calling routine. This routine ends with setting the variables as explained above, erasing the dialog box, resetting the cursor to a cross type cursor and returning the results to the parent routine.

PROCEDURE FrameMill (Snap : BOOLEAN;
                        SnapSize,Xzero,yzero,zpixel : INTEGER;
                        Var u : INTEGER;
                        Var dat : DTARRAY);

This procedure is the main procedure for rectangular frame milling. It is within this procedure, that G170, G171, and G172's are generated. This procedure begins with a call to procedure FrDecision, as described above. The three possibilities of the results are as follows.

If Pocket was TRUE, then the user intended to machine a pocket and no further boolean checks are necessary and the program proceeds to the next stage.
If Pocket was FALSE, then the user intended to do a frame mill. Two possibilities exist here, an outside frame mill or an inside framemill. If the boolean variable 'Inside' is TRUE, this indicates an inside frame, and if it is FALSE, this indicates an outside frame.

From this point, processing the rest of the information is almost identical for all the frame routines. The user is asked to pick a radius for the fillet, and the result is picked up by a call to scaleselect. If no fillets are desired, the user may respond with a corner radius of 0. After the radius is determined, the user is prompted to pick the total depth of the frame to be machined. The result is again acquired with a call to scaleselect. Once the radius and the depth are determined, the program asks the user to select the top left and the bottom right of the rectangle to be machined. A call to Mtrack returns the top left and bottom right coordinates. The format in which this information is stored in the array for future retrieval is as follows:

\[
\begin{align*}
\text{DAT}[i,2] &: \quad \text{left coordinate of rectangle} \\
\text{DAT}[i,3] &: \quad \text{top coordinate of rectangle} \\
\text{DAT}[i,4] &: \quad \text{right coordinate of rectangle} \\
\text{DAT}[i,5] &: \quad \text{bottom coordinate of rectangle} \\
\text{DAT}[i,6] &: \quad \text{Radius of the Fillets}
\end{align*}
\]
DAT[i,7] : Total Z depth of the frame

At this point, depending on the type of frame or pocket to be machined, the pen size, the pen pattern, and the offsets are set appropriately, and the procedure RecDraw is called to do the graphical display of the frame on the screen.

PROCEDURE DrillHole (snap : BOOLEAN;
SnapSize : INTEGER;
Var u, xzero, yzero : INTEGER;
Var Dat : DTARRAY;
Var toolDia : REAL);

This procedure is responsible for all the hole drilling in the program. There are two possibilities, if the depth of the hole is less than 0.5 inch, then a G 81 is generated for a straight hole drill. If the depth is greater than 0.5 inch, then a G83 is generated for a peck drilling cycle.

The procedure begins with setting up the dialog area to ask the user if he wishes a tool change before specifying a drill cycle. The diameter of the current tool is displayed on the top. If the user wishes a tool change, the procedure ToolChange is called before proceeding with the drill routine.
The first step in the drill hole routine is to ask the user for the depth of the hole to be drilled. The value is input into the with a call to scaleselect. The user is then prompted to select the location of the hole using the mouse. This X, Y location is read with a call to the Mtrack routine. The format in which the information for a drilled hole is stored in the array DAT is as follows:

\[
\begin{align*}
\text{DAT}[i,2] & : \text{ X coordinate of the hole center} \\
\text{DAT}[i,3] & : \text{ Y coordinate of the hole center} \\
\text{DAT}[i,4] & : \text{ Depth of the hole to be drilled} \\
\text{DAT}[i,5] & : \text{ Diameter of the hole to be drilled, in effect the current tool diameter.}
\end{align*}
\]

The user has to go through the entire procedure to place multiple holes, because this does not tie up the user to one hole depth. A future enhancement could be to ask the user if this was multiple hole drill, or a bolt circle drill and accommodate those G codes as well. This concludes the discussion of the procedure to drill holes.

```pascal
PROCEDURE CircleMill ( snap : BOOLEAN;
    SnapSize, xzero, Yzero,
    Zpixel : INTEGER;
    Var u : INTEGER;
    Var dat : DTARRAY;
```
This procedure is very similar to the Frame mill routine. It has the same features and a need for the same variables.

This procedure begins with a call to procedure FrDecision, as described above. The three possibilities of the results are as follows.

If Pocket was TRUE, then the user intended to machine a pocket and no further boolean checks are necessary and the program proceeds to the next stage.

If Pocket was FALSE, then the user intended to do a circle mill. Two possibilities exist here, an outside circle mill or an inside circle mill. If the boolean variable 'Inside' was TRUE, this indicated an inside circle, and if it was FALSE, this indicated an outside circle. These boolean results are provided by a call to the procedure FrDecision.

From this point, processing the rest of the information is almost identical for all the circle routines. The user is asked to select the radius of the circle, and the result is input through scaleselect. The user is then asked to select the depth of the hole, this information also being acquired through a call to the routine ScaleSelect. The user is then asked to indicate the center of the circle, the values of the
X and Y coordinates being retrieved by a call to Mtrack. The format in which the above information is stored is as follows:

\[
\begin{align*}
\text{DAT}[i,2] & : \quad \text{X coordinate of the hole} \\
\text{DAT}[i,3] & : \quad \text{Y coordinate of the hole} \\
\text{DAT}[i,4] & : \quad \text{Radius of the Hole} \\
\text{DAT}[i,5] & : \quad \text{Final depth of the hole}
\end{align*}
\]

The possible assignments for \( \text{DAT}[i,1] \) are 175, for an outside circle, 176 for an inside circle and 177 for a circular pocket mill. The program first draws the actual sized circle without offsets and using a pensize of 1 and a pen pattern of black. An IF statement after this drawing determines how the circle to indicate the tool path will be drawn in terms of offset, pen pattern, and pen size. This concludes the discussion of the CircleMill procedure.

```
PROCEDURE CutterComp (Snapsize, xzero, yzero : INTEGER;
                      Var u, xon, yon : INTEGER;
                      Snap : BOOLEAN;
                      Var Dat : DTARRAY;
```

This procedure is called from within the LineDraw routine if the user has decided upon cutter compensation. This procedure is responsible for determining the tool position, whether it
is on the left of the part or the right of the part being machined. It also stores the users choice into the array 'dat' for later retrieval by the post processor.

Drawdialog is invoked, after setting up the parameters, to determine if the tool is on the left or on the right of the part being machined. The user has the opportunity to back out of this option if he desires. After determining the tool side, the direction in which the user draws his polygon will determine whether the milling will be climb milling or conventional milling. If the user has opted for cutter compensation, then he has to select a cutter compensation turn on point. This turn on point is selected using the mouse and this value of X and Y is tagged on to the G 41 or G 42 code that is generated depending on the type of compensation selected. The format in which this information is stored is as follows:

\[
\begin{align*}
\text{DAT}[i,2] & : \quad \text{X coordinate of cutter compensation turn on point} \\
\text{DAT}[i,3] & : \quad \text{Y coordinate of the cutter compensation turn on point}
\end{align*}
\]

\text{DAT}[i,1] takes on the Value 41 or 42 depending on whether the tool is to the left of the part or to the right. The variables 'xon, and yon' are copies of the compensation turn
on point so that when compensation is turned off, it is done with respect to the same point. This concludes the discussion of procedure CutterComp.

PROCEDURE CompOff ( Var u, xon, yon : INTEGER;
                      Var dat : DTARRAY);

This procedure is accessed to turn off the cutter compensation feature that the user may have selected. This is achieved via a G40 code, with respect to some X, Y value. The turn on point in procedure CutterComp was stored as 'xon', 'yon'. This point is also used to turn off the cutter compensation. This routine is called on exiting from the line draw routine. The format of the information stored is as follows:

\[
\begin{align*}
\text{DAT}[i,1] & : 40; \text{signifying the G 40 code} \\
\text{DAT}[i,2] & : \text{X coordinate of Cutter compensation turn off point} \\
\text{DAT}[i,3] & : \text{Y coordinate of Cutter compensation turn on point}
\end{align*}
\]

This routine does not output any information to the screen but influences the resulting G codes. This concludes the discussion on procedure CompOff.
PROCEDURE DrawLine ( xold, yold, x, y : INTEGER;
     Comp : BOOLEAN);

This procedure is used to draw lines on the screen. It is the only routine relied upon by the line and polygon generation routine to do the graphical display. This procedure moves to the coordinate (xold, yold) and draws a line to the current (x, y) coordinate. If the user has opted for cutter compensation, then the lines are drawn in gray to distinguish them from uncompensated paths.

PROCEDURE Line ( Snap : BOOLEAN;
     Var Comp : BOOLEAN;
     SnapSize, GridSize, X, Y, Zpixel,
     xzero, yzero : INTEGER;
     Var u,v : INTEGER;
     Var dat : DTARRAY;

This routine handles all the line drawing in the program. It is the master sub routine among the various routines that support the line drawing feature. The procedure begins with highlighting the line icon to show that this mode is in effect. The user is then prompted via the dialog box if cutter compensation is desired. The task of getting the response is assigned to the procedure YesNo. If compensation is desired, procedure CutterComp is called, as explained
above. If not, the cursor is set to the cross hairs and the user begins the line drawing.

The main event loop in the line routine flags down the very first point that is selected after entering this procedure. This is because the first point is reached by a MoveTo statement instead of a Lineto. Thus it is a G0 code instead of a G1. The user may exit the line procedure at any time by selecting the cross marked icon. If snap has been selected, the routine maintains the snap during mouse polling. The format in which the data for the very first call, a G0 is stored is as follows;

\[
\begin{align*}
\text{DAT}[i,1] & : \text{ is given a code 0, for a G0} \\
\text{DAT}[i,2] & : \text{ X coordinate of the point} \\
\text{DAT}[i,3] & : \text{ Y coordinate of the point} \\
\text{DAT}[i,4] & : \text{ Z depth at the point, in effect the first plunge depth}
\end{align*}
\]

The setting of the above values is controlled by a boolean local variable called 'FirstFl' that is set true on entering line draw. As soon as the above parameters are set, 'FirstFl' is set false, so that a second G0 is not invoked within the same call to the procedure. The values of X and Y from the above algorithm are also stored as xold, yold so that a Moveto this old position can be executed before a line is
drawn with the Lineto procedure. In the polling loop in the line draw routine, as long as the user keeps the mouse button held down, a 'rubberband' line is always displayed on the screen from the previously selected point to the current mouse position. This gives the user a better idea of the line he is about to place. The mouse button has to be held down for this mode to be active. The line is repeatedly redrawn and erased, as long as the button is held down. The erasure of the line is achieved by changing the pen mode to do a BOOLEAN XOR and the line is redrawn.

If the point selected is not the first one in this routine, then the DrawLine routine draws the line from the old X and Y position to the new one selected. The information of this line is stored as follows.

\[
\begin{align*}
\text{DAT}[i,1] & : \text{ is given a code 1, for a G1} \\
\text{DAT}[i,2] & : \text{ X coordinate of the point} \\
\text{DAT}[i,3] & : \text{ Y coordinate of the point} \\
\text{DAT}[i,4] & : \text{ Z depth at the point}
\end{align*}
\]

As soon as the information is stored, the current X and Y values are transferred into the variables 'xold', and 'yold' so that a continuous string of lines may be generated until the user decides to quit the routine. This concludes the discussion of the procedure Line.
PROCEDURE SetSnap ( Var SnapSize : INTEGER;
                   Var Snap : BOOLEAN);

This procedure is used to set the snap, or remove it after it has been set. It may be selected from the main icon menu. When this routine is invoked, the user is asked to select the snapSize, via the dialog box and the routine ScaleSelect. If the user selects a non zero value for snap then the variable 'snap' is set to true and the snap size is recorded in the variable 'SnapSize'. These are then passed on as parameters to other routines. A word about the relation between the part origin and the snap is in order here. If the user selects the part origin exactly lined up with one of the grid points, the snap will also be aligned to the grid. If the part origin is set offset from the grid points, then the snap will also be offset by the same degree from the grid units.

PROCEDURE Zdepth ( Var Zpixel : INTEGER);

This procedure is called from the main icon level to set the current Z depth of the system. The Z icon is highlighted by drawing it in a pensize of 3, and the user's request is serviced. The procedure calls Zset to set up the scale to be used for selecting the depth. Zset has been discussed previously. The mouse is then polled for a valid click.
reasonably close to the scale drawn. An invalid click gives rise to a system beep and discarding of the illegal click.  
ShowXy is used to display a digital indication of the Zdepth when the mouse button is held down. The X value of ShowXy is held at zero and the normally Y coordinate is used to display the current Z value.

The Z depth selected from this procedure is returned to the requesting routine in the parameter Zpixel. This value is in pixels and it is the responsibility of the calling routine to convert it into inches by dividing by 32.

THE MAIN PROCEDURE

This description pertains to the main program of this application. The main program begins with calls to WindowInit, InitHand, InitPencil, InitCross, all of which have been previously discussed. The user selects the snap size to be used initially. A snap size of 0 effectively turns off the snap feature. The snap may be reset at any time within the program by clicking on the snap icon. The Zdepth is initially set to zero. The toolset procedure is called to set the chip loads and the cutting speeds to be used with the first tool. All the array indices such as u and v are set to their initial default values. The gridsize is also set to zero and the tool change procedure is called. Procedure
toolchange returns the spindle speed, the feed rate and the plunge rate based on the tool diameter of the selected tool. The Gridset procedure is called next. Procedure Setup is called to set up the screen and draw the icons and the grids on the screen. The user is then prompted for the part origin. The polling for the part origin is done locally.

Once the above setup operation is completed, the procedure goes into the main eventloop. This loop polls the mouse and looks for clicks in the valid menu area. On detection of a valid click, it calls the appropriate procedure to service the user's request. The calls to the various procedures are self explanatory and have been dealt with where the individual routines have been discussed and will not be repeated here. On selecting the exit Icon, the program flashes the E in the icon by redrawing it in the outline font and exits the application. This concludes the discussion of the main procedure.
THE FRONT VIEW ALGORITHM

The front view algorithm is a highlight of this program. It is designed to allow the user to view the geometry description in the front view and actually see a visual display of the geometry with the Z plane being prominent.

The need for this unit was felt during initial program development, as early as the preliminary research stage as it was difficult to see if the tool motions would indeed generate the required parabola. The screen display would draw a parabolic shape from the data provided by the user, with no relation to the tool motion or the G codes that were generated subsequently. Thus a close examination of the resultant code, or a dry run on the bridgeport was the only way to see if the Z movements were correct. In terms of the general software, it was anticipated that this problem would multiply in magnitude since geometries were not predefined, as in the case of the parabola. Thus it was imperative that a front view algorithm be included.

The author chose the front view generation as the simulation technique, rather than the unofficial standard of tool path simulation. This gives the user a clear representation of the geometry in orthographic projection as well as accomplish the basic objective of examining the tool paths.
At the conception phase, it was decided that the algorithm would be separated from the mainstream of the program, still maintaining full integration with the software. It was planned initially, to continuously and automatically generate the front view at the bottom half of the screen. This idea was abandoned as it was found that this occupied too much valuable screen space, it would delay the geometry generation, and the fact that the user may not want to have a continuous front view display. It was thus decided to include the front view generator as an icon in the main menu. This enables the option to be invoked, as many times as the user desires without actually occupying screen space.

Conceptually, when the front view option is invoked, the procedure saves the current drawing to disk as a file called 'Temp'. It then erases the screen and redraws the entire drawing window, dedicating it to the front view. At the end of viewing, the software completely erases the drawing window again and loads the file 'temp' from disk and recreates the drawing. The user is allowed the option of stepping through the front view generation or seeing the final result. The step option additionally displays the current Z depth as a digital readout at the lower left of the screen through the step.
The first problem that this algorithm presented was the actual generation of the front view. One of the initial alternatives considered was to use a rotation matrix and transform the data array into a set of coordinates that represent the object in its front view. This method would have worked, but it was too expensive to implement in terms of CPU time, and speed. The rotation matrix would consist of 1, -1, and 0 since the rotation was 90 degrees.

Giving the problem further thought, it was decided to make a mini interpreter to display the front view. Since the original plan was to use the first column of the data array\(^1\) as a tag to indicate the type of G code being generated. For instance, a G1 is a straight line, a G91 is a drill hole, a G171 is a frame mill and so on. These same tags could be used, not only to generate the G codes, but also the geometry front view. The main body of this procedure has a loop that has several routines built into it as follows.

```
LOOP
  IF TAG=1 THEN
    BEGIN
      (The G1 handling routine)
      END
```

\(^1\) Please see the DATA ARRAY sub heading in the software description section
ELSE IF TAG=81 THEN
    BEGIN
    {The Drill Hole Routine}
    END
ELSE IF TAG=171 THEN
    BEGIN
    {The Frame Mill Routine}
    END
ELSE IF .............
    .
    .
    .
    END IF
END LOOP

Each routine then has a algorithm built in to handle the graphics to generate the front view for that particular move. The generic approach of all these algorithms is to ignore the Y coordinate in the data array and to make use of the Z coordinate in the generation. A more detailed explanation of these algorithms is provided in the software description section.

The other major problem to consider while designing this algorithm was the location of the front view on the screen. For completeness sake and to make the front view more
FRONT VIEW DISPLAY

Top View Area

-------------------

Front View Area

Z Depth Display
Z DEPTH  =  0.625"  

Prompt Area
meaningful, it was decided to simultaneously display an entity in the front view and the top view at the same time. For obvious reasons, the front view display had to be scaled to fit the drawing window, or this would lead to problems on extremely large drawings, or on small ones.

The screen was arbitrarily divided into two halves, lengthwise. The top half would display the top view, or the drawing as drawn by the user, and the bottom half, the front view. On screen this demarcation is shown by a dotted line. This line also serves as the zero reference on the front view, that is \( z = 0 \). The position of this line is fixed, irrespective of the size of the image, and the image is force scaled to fit this region. This ensures that the program can handle any situation that the user might find himself in.

The scaling is handled by an algorithm that first scans the data array and sets the very first \( x,y,z \) coordinates into variables 'minx', 'maxx', 'miny', 'maxy', 'minz', 'maxz'. The first \( x \) coordinate goes into min and max \( x \) and so on. With this initial set up, the complete data array is scanned and coordinate values are compared to the above variables. If a correspondingly smaller or larger value is located for a specific coordinate, then the variable is updated to this new value. At the end of this scan, the corresponding variables have the minimum \( x,y,z \) coordinate values of the
drawing. These points can then be used to create an imaginary window in the \(X,y\) plane, and the \(x,z\) plane, so that the maximum attributes of the drawing are known. These are also used to determine the scale factor as outlined in the procedure description in software description section.

Once the scale factor is determined, the 'xzero', 'yzero' points, or the part origin that the user specifies in the beginning, are the first points to be scaled. They are multiplied by the computed scale factor. The variables mentioned above are also scaled. Once these 'X - Y', 'X - Z' limits are scaled. The point that represents the lower left limit of the drawing is moved to a predetermined location the screen by a simple translation. This enables the software to define a base point for the rest of the image to be generated.

The above settings being determined, the algorithm proceeds with the actual image generation as explained previously, and in the software description section.
FRONT VIEW DISPLAY

Z = 0
TESTING AND DEBUGGING

Testing and debugging this software at the final stage would have been an impossible task due to it's magnitude and level of interaction between the several modules. The approach that was followed to test this software was to test and debug each module extensively as it was written. This enabled the author to make sure that the software was performing as it should in a simple and effective manner.

Initial tests on the G code files generated were done by making the software generate very simple geometry and comparing this with versions that were coded by hand. This was done for all the features that the program supported. As the program passed initial tests, the geometry was made a bit more complex each time to see if the program was corrupted by interaction of more routines. Some routine bugs were found and fixed as they occurred.

These G files were then downloaded to the Bridgeport and the file was run with no Z movement to observe the tool paths that resulted from the geometry. Once it was certain that the program was functioning without any problems, the parts were machined using machine wax. They were found to be accurate.
In the case of canned cycles, the tool entry clearance, the step depths and the step over in the case of the pockets were found by trial and error. The program, at this stage, codes these values based on constant internal definitions.

Cutter compensation is still a minor problem. In the case of closed polygons, the cutter does not compensate the very first and the very last move of the compensated path. The author is working on this problem and this is the only known bug in the system. Cutter compensation works fine in other instances. At this point, the solution is to define the geometry of the compensated path and not use the internal function in the case of closed polygons.

Most of the other problems that occurred are described in the relevant section in the software definition, and in the communications section.
EXAMPLES

The following section details the steps followed to create the geometry in examples 1 and 2. It does not attempt to justify or describe the choices as this is done adequately in the user's manual and the software description. This section is intended as a 'getting started' section to give a new user a feel for the MacCam system.

EXAMPLE 1

- Start the MacCam application by double clicking on the application icon

- Select the material as wax (or any other material to be machined)

- Select tool number 1

- Tool diameter 0.375"

- Select grid size as 0.5"

- Select the part origin as the center of the drawing

- Pick the snap option from the menu and set snap to 0.25"
**Example 1: Not to Scale**

1112

N1G0G90T1D0.5000S2400M26
N2G172X2.5000Y2.5000Z.05X5.0000Y5.0000R0.2500Z0.2500Z.125P.25P.25P.01 F29.F15.F15.
N3G177X2.5000Y2.5000Z.05R1.2500Z0.3750Z0.125P.25P.25P.01F29.F15.F 15.
N4G0X2.0000Y4.0000Z0.05
N5G1Z0.0000F15.
N6G1X3.0000Y4.0000Z0.0000F29.
N7G1X4.0000Y3.0000Z0.0000F29.
N8G1X4.0000Y2.0000Z0.0000F29.
N9G1X3.0000Y1.0000Z0.0000F29.
N10G1X2.0000Y1.0000Z0.0000F29.
N11G1X1.0000Y2.0000Z0.0000F29.
N12G1X1.0000Y3.0000Z0.0000F29.
N13G1X2.0000Y4.0000Z0.0000F29.
N14G0G90T2D0.2500S4200M26
N15G0X4.5000Y4.5000Z0.05
N16G83X4.5000Y4.5000Z0.7500Z0.25Z0.125F50.
N17G0X0.5000Y0.5000Z0.05
N18G83X0.5000Y0.5000Z0.7500Z0.25Z0.125F50.
N19G0G90T3D0.3750S3200M26
N20G0X0.5000Y4.5000Z0.05
N21G83X0.5000Y4.5000Z1.0000Z0.25Z0.125F38.
N22G0X4.5000Y0.5000Z0.05
N23G83X4.5000Y0.5000Z1.0000Z0.25Z0.125F38.
N24G0X0.0Y0.0Z1.0M30
- Pick the Frame icon from the menu. Select Frame from the dialog box, and 'Inside' when prompted for inside or outside frame.

- Indicate the radius of the fillets, the total depth of the channel, the top left corner and the bottom right corner of the frame. This completes the frame definition.

- Select Circle Mill option from the icons.

- Select pocket, from the frame or pocket dialog. Select the radius of the circle to be milled, and the total depth. Indicate the center of the pocket on the screen. This completes the circular pocket definition.

- Pick the drill hole icon.

- Select Change tool option. Select tool #2. Indicate the diameter of the tool. Indicate the depth of the top left hole. Click the top left hole center.

- Pick the drill hole icon.

- Select the place option and indicate the depth of the bottom right hole. Indicate the location of the hole on the
- Pick the drill hole icon.

- Select the change tool option

- Select tool number 2 and indicate the depth of the upper right hole.

- Indicate the location of the hole.

- Pick the place hole option from the icons.

- Indicate the depth of the bottom left hole and it's location.

- Select the Line draw option from the menu.

- Indicate cutter compensation off

- Draw the hexagonal path

- Click on the cross icon to exit the line draw menu

- Click the save icon to save the drawing, giving it a suitable name
- Click the Front view icon to see the front view of the tool paths

- Click the 'G' icon to generate the G codes.

This writes a file called 'GFile' to disk which is ready to download on to the bridgeport.
EXAMPLE 2:

- Start the MacCam application by double clicking on the application icon

- Select the material as wax (or any other material to be machined)

- Select tool number 1

- Tool diameter 0.375"

- Select grid size as 0.5"

- Select the part origin as the lower left of the drawing

- Pick the snap option from the menu and set snap to 0.25"

- Pick the Frame icon from the menu. Select Frame from the dialog box, and 'Inside' when prompted for inside or outside frame.

- Indicate the radius of the fillets, the total depth of the channel, the top left corner and the bottom right corner of the frame. This completes the frame definition.
EXAMPLE 2: Not to Scale

:1212
N1G0G90T1D0.2500S1600M26
N2G171X4.5000Y2.5000Z.05X9.0000Y5.0000R0.2812Z0.2500Z.125P0.5P.01
N4G0X1.0000Y3.5000Z0.05
N5G1Z-0.1875F7.
N6G1X2.0000Y4.5000Z-0.1875F14.
N7G1X6.0000Y4.5000Z-0.1875F14.
N8G1X7.5000Y3.0000Z-0.1875F14.
N9G1X7.5000Y2.5000Z-0.1875F14.
N10G1X6.5000Y1.0000Z-0.1875F14.
N11G1X2.0000Y1.0000Z-0.1875F14.
N12G1X1.0000Y2.0000Z-0.1875F14.
N13G1X1.0000Y3.5000Z-0.1875F14.
N14G0X5.0000Y1.5000Z0.05
N15G1Z-0.4062F7.
N16G1X5.0000Y4.0000Z-0.4062F14.
N17G1X5.5000Y4.0000Z-0.4062F14.
N18G1X6.0000Y3.5000Z-0.4062F14.
N19G1X6.0000Y1.5000Z-0.4062F14.
N20G1X5.0000Y1.5000Z-0.4062F14.
N21G0G90T2D0.5000S800M26
N22G0X0.5000Y4.5000Z0.05
N23G83X0.5000Y4.5000Z0.5000Z0.25Z0.125F7.
N24G0X0.5000Y0.5000Z0.05
N25G83X0.5000Y0.5000Z1.0000Z0.25Z0.125F7.
N26G0X0.0Y0.0Z1.0M30
- Select Circle Mill option from the icons.

- Select pocket, from the frame or pocket dialog. Select the radius of the circle to be milled, and the total depth. Indicate the center of the pocket on the screen. This completes the circular pocket definition.

- Pick the drill hole icon.

- Select Change tool option. Select tool #2. Indicate the diameter of the tool. Indicate the depth of the top left hole. Click the top left hole center.

- Pick the drill hole icon.

- Select the place option and indicate the depth of the bottom left hole. Indicate the location of the hole on the screen.

- Select the line draw icon.

- Select no cutter compensation, and draw the outer polygon

- Select the exit line option by clicking the cross icon
- Repeat the above steps for the inner polygon

- Click the save icon to save the drawing, giving it a suitable name

- Click the Front view icon to see the front view of the tool paths

- Click the 'G' icon to generate the G codes.

This writes a file called 'GFile' to disk which is ready to download on to the Bridgeport.
MacCam MENU STRUCTURE
The computer used for this project has the following specifications

Macintosh + with 1 MB RAM
One Printer Port
One Modem/peripheral Communication Port
One 800 K Double sided Internal Disk Drive
One 800 K Double sided External Disk Drive
ImageWriter Printer.

The compilers used in the development of the application is as follows;

Macintosh Pascal Version 2.0 (For initial Development)

LightSpeed Pascal (For the final application)

The communications software used for the DNC link is

Red Ryder Version 10.0
APPENDIX B
BRIDGEPORT MACHINE SPECIFICATIONS

- Controlled 3 axes (x, y and z), rotary axis (optional)

- Simultaneous controllable axes
  - 3 axes of the four available (linear interpolation)
  - 2 axes in any plane (circular interpolation - XY, XZ, YZ)
  - X, Y axis circular, Z linear (Helical or spiral interpolation)

- Least input increment
  - Inch system : 0.0001
  - Metric system : 0.001 mm
  - Degrees : 0.001

- Least output increment
  - Inch System : 0.0001
  - Metric System : 0.00254 mm

- Maximum Programmable dimensions
  - Inch system : +/- 838.8607
  - Metric System : +/-8388.607
  - Degrees A : +/- 720.000
- Degrees C : +/-8388.607

- Part program code, EIA RS-358-A¹, 7 bit, Even parity

- Part Program format : Variable block, Word address format

- Feed rate : Programmable directly between 0.1 and 250.0 inches per minute (ipm)

- Automatic Acceleration / de-acceleration

- Controller : R2E4, Acronym for Rigid Ram, 2 horsepower spindle motor, E type control, 3 axes milling machine

- Operating System : BOSS 9 (Bridgeport Operating System Software)

¹ RS 358, commonly called the EIA code, is a subset of the USA standard code for information interchange for numerical control perforated tape
Communicating with the Bridgeport CNC vertical milling machine is much like any other device. RS 232 communications standards are the same, irrespective of the standards used. The following appendix thus discusses RS 232 communication from a general standpoint. All of the information here is pertinent to the Bridgeport also. Other communication protocols and Hardware handshaking is included in this appendix for completeness.

Computers communicate with several devices during normal use. Mostly, these devices are pre-wired by the manufacturer and require nothing from the user to make them work. However, as the user decides to add other devices to the computer at a later stage, he accepts responsibility to make sure communication is established.

Communication between devices is a simple matter, but very critical because of the speed at which they take place. Devices have to follow some basic rules to communicate. Imagine two people talking to each other at the same time about an important issue, each person so engrossed in talking that he forgets to listen. Likewise if both computers are transmitting information, data is lost since neither of them is receiving it.
To permit proper communication between devices, there has to be some standard protocol established. There are several protocols available depending on their use, they are broadly classified as

a) RS-232 Serial communications  
b) Centronics Parallel  
c) IEEE488 Instrumentation

The serial and parallel protocols are available on most microcomputers. They are commonly referred to as ports.

CENTRONICS PARALLEL INTERFACE

The parallel protocol is not supported by the Macintosh computer or the Bridgeport milling machine and has been outlined here for completeness. This protocol transmits data a byte at a time, over 8 data lines. This interface is widely used to connect printers to computers, that will be located close to each other. Parallel interfaces cannot be used for long distance communications. This is because long lengths of wire that are grouped together induce a capacitance between themselves, this together with the resistance of wires itself cause a high value of the RC time constant and causes disturbances in transmission.
IEEE488

The parallel protocol is not supported by the Macintosh computer or the Bridgeport milling machine and has been outlined here for completeness. This is a communication protocol defined by the Institute of Electrical and Electronic Engineers. It is primarily an instrumentation interface, that is, it has been designed with process instrumentation in mind. This is similar to Hewlett Packard's Hewlett Packard Interface Bus (HPIB), also known as the General Purpose Interface bus (GPIB).

RS 232 SERIAL INTERFACE

HARDWARE

This is the most commonly used interface for communicating with devices and is the protocol used for the Macintosh-Bridgeport communications. The following table shows the pin numbering on a typical DB 25 female connector, looking at it from the front. The pins important to our discussion are:

<table>
<thead>
<tr>
<th>PIN #</th>
<th>NAME</th>
<th>ABBREVIATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>Description</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>1</td>
<td>Frame Ground</td>
<td>FG</td>
</tr>
<tr>
<td>2</td>
<td>Transmit Data</td>
<td>TD or TxD</td>
</tr>
<tr>
<td>3</td>
<td>Receive Data</td>
<td>RD or RxD</td>
</tr>
<tr>
<td>4</td>
<td>Request to Send</td>
<td>RTS</td>
</tr>
<tr>
<td>5</td>
<td>Clear to Send</td>
<td>CTS</td>
</tr>
<tr>
<td>6</td>
<td>Data Set Ready</td>
<td>DSR</td>
</tr>
<tr>
<td>7</td>
<td>Signal Ground</td>
<td>SG</td>
</tr>
<tr>
<td>8</td>
<td>Data Carrier Detect</td>
<td>DCD</td>
</tr>
<tr>
<td>20</td>
<td>Data Terminal Ready</td>
<td>DTR</td>
</tr>
</tbody>
</table>

RS 232 communication is mostly asynchronous, that is the computer and the device do not have to be in synchronization with one another but they use several means to make sure data does not get lost or misinterpreted.

The most important of the above mentioned connections are Pins 1, 2, 3, and 7. These are the bare minimum to make a successful RS 232 link, and are also the only ones needed for a successful Bridgeport interface.

Pin 1 is the Frame ground or the Chassis ground. This is a protective ground just to make sure that both devices are grounded to the same potential to prevent electrical hazards.

Pin 2 is the Data transmission line. All information from the host computer to the device takes place on this line. Pin
number 2 on the device, correspondingly is the pin on which the device communicates to the host.

Pin 3 is the Data receive line, All information transmitted by the device to the host is received on this line. Consequently all information transmitted by the host is received on pin 3 of the device.

Pin 7 is the signal ground or the reference ground for the signal being transmitted on both devices.

The above definitions show that the computer and device could be set up to communicate in a simple manner. For example, the printer knows that it will have nothing to say to the computer until the computer transmits information, and the computer knows that no response will be forthcoming from the printer until it has received something first. Thus when communication begins, the computer begins transmitting without even checking its receive line. Likewise, at the end of transmission, the computer checks its receive line for a status report. This "understanding" between devices ensures successful communication between the device and the computer.

For devices, more intelligent than a printer, it is necessary to establish a better defined communication protocol. The RS 232 addresses the problem by using Handshaking lines.
Handshaking is the term used to indicate that lines, other than the common ones described above will be used in communication. These lines are pins 4, 5, 6, 8 and 20. Here is a typical session that summarizes the use of these pins.

When the computer is turned on, it asserts its DTR line. If any device is attached to the RS-232 port, it takes this to be a signal that it is attached to a live controller, the computer. The device in return responds by setting its DSR line active, informing the computer of the physical presence of a device on the other end of the cable. This is the first step.

Now when the computer is ready to transmit to the device, it first asserts its RTS line. By doing this, the computer requests the device for clearance to transmit information to it. The device responds by setting its carrier detect (DCD) active. This reassures the computer that it's request has been received by the device. Then, at the instant the device is ready to receive information, it asserts its CTS line and this informs the CPU that it is Clear to Send information. At this point data transfer begins.

It is to be noted that the exact same procedure holds when the computer is the receiver and the device is the
transmitter as may be the case with remote bar code reading equipment. In this case, it is the device that begins the above protocol by asserting its request to send line and everything that follows is as described above.

DTE AND DCE

There are two kinds of RS 232 specifications. They are Known as Data Terminal Equipment (DTE) and Data Communications Equipment (DCE). Normally, if the device and the computer had pin connections as shown above, then it would be necessary to cross pins 2 and 3 while making the cable. This connects the receive of the device to the transmit of the computer and vice versa. Devices that are termed DCE have pins 2 and 3 reversed within the device itself. Thus on DCE equipment, Pin 2 is RxD and pin 3 is TxD. This allows straight through cabling. Normally computers are set as DTE and devices are set as DCE, though this is alterable by setting of dip switches inside the machine.

SOFTWARE

The above discussion pertains to the hardware connection between computers and devices. The discussion refers to the sending and receiving of information, but does not mention the format the information uses. This is the realm of software.
Consider the transmission of the ASCII character A. This has a decimal code of 65, which is 01000001 in Binary. In the RS-232 protocol, a binary 0 represents a voltage of +3 to +25 volts, and a binary 1 is represented by a -3 to -25 volts. When the computer prepares to transmit an A, it sends a waveform similar to the figure below over its transmit line. This raises some questions. How does the receiver know when the waveform on the receive line is valid data? If it is valid data, then how does it separate a byte from the next? Is there a way to make sure that no error occurred during transmission? Can this be further confirmed by the host? This leads us to the definition of the following software parameters.

XON/XOFF

Xon - Xoff characters are software handshaking that most devices use in the absence of hardwired handshaking. In a typical situation, when the Macintosh is transmitting a file to the Bridgeport, the transmission buffer on the Bridgeport may get full. This forces the Bridgeport to empty the contents of its buffer into its RAM before accepting fresh information. At this time, if the Macintosh is not told to hold transmission while the Bridgeport is performing its housekeeping instructions, data that the computer is trying
to send will be lost for ever. Likewise, if the Bridgeport is downloading a file to the Macintosh, and the user inserts a disk in the drive, or presses the abort key, the Macintosh has to determine what event has occurred. At this point, it should be able to tell the Bridgeport to hold transmission until it is ready again.

The characters that allow for this are the XON XOFF characters, which stands for TRANSMIT ON, TRANSMIT OFF. The XON character is the CONTROL Q character, or the ASCII character 19, the DC3 general purpose device control signal 3. The XOFF character is the ASCII character CONTROL Q, which is ASCII 17, DC1, general purpose device control signal one. The XON - XOFF protocol is automatically used by the Macintosh and the Bridgeport and the user does not normally interfere in the communication. The user however gets to specify if the computer is allowed to use this protocol. In this particular situation, it is turned on since the Bridgeport uses this software handshaking.

DATA FORMAT

RS 232 devices send bytes serially on the transmit line and receive bytes serially on the receive line. Since data communication is essentially asynchronous, bytes are sent
RS 232 CHARACTER FRAME

START       PARITY STOP
T1  T2  T3  T4  T5  T6  T7  T8  T9

7 BIT ASCII

LSB  MSB
typically in packages. Each package may typically consist of a start bit, followed by a parity bit, seven data bits (eight if no parity is used) and a stop bit. These transmissions occur at some predetermined speed termed as the Baud rate.

**BAUD RATE**

Baud rate is commonly considered to be the rate at which data transmission occurs, the number of bits transmitted per second. This definition, though popular, is incorrect. Consider the example of a computer set to transmit at 9600 Baud. This indicates that if the computer transmits uninterrupted for 1 second, it would have transmitted 9600 bits per second. This may be restated to say that the duration of each bit was

\[ \frac{1}{9600} = 104 \text{ microseconds.} \]

Thus a baud rate of 9600 denotes that the duration each bit will be on line is 104 microseconds. This does not imply that 9600 bits will be transmitted in a second, since the computer may cease transmission temporarily to service some other task and then resume transmission a little while later. In reality, the wave rise time and drop time is to be taken into consideration and the computer samples the serial port and if the signal was active for at least 90% of the time during
a given 104 microsecond period, then it is a binary one, or if the signal was inactive for at least 90% of the same 104 second period, then it is a binary zero. Common Baud rates are 110, 150, 300, 1200, 2400, 4800, 9600, 19200 bits per second. Baud rates as high as 57200 are also available for special applications. Mainframe communications occur at several million bits per second.

THE START BIT

This bit indicates the start of the byte that is to follow. Generally, there is one start bit per byte transmitted. It is the first part of the package. The end of the package is appended with a stop bit that usually is the same number as the number of start bits.

THE PARITY BIT

Parity is an error checking scheme that may be used by RS 232 devices. In data transmission, there may be times that a bit may get affected by a power surge and the resulting byte is incorrect. Parity may be even, odd or none. Selection of even parity implies that all packages of bytes sent or received will always have an even number of 1 states. For example, if parity was set to even and at a given instant, the computer is transmitting the binary equivalent of the
character A, which in 7 bit ASCII is 100 0001. In this packet, there are an even number of ones, thus it is a legal byte. Consider then the transmission of the character C, which in seven bit ASCII is 100 0011. This has an odd number of ones in the package, thus the parity is set to a one, since an even number of bits must be included in the package. Suppose the data byte has an odd number of bytes, parity is set to one, so that the net package has an even number of bits. Likewise, odd parity implies the transmission or reception of an odd number of bits in each package of bytes. A none parity disables this checking and data is transmitted 8 bit ASCII instead of the 7 bit ASCII code. Modem communication protocols such as XMODEM and its variations require the use of 8 bit data transmission, since the file being transmitted is not pure ASCII, but an application or an EXE file, that has eight bit codes in it. Error checking is then done by the protocol itself, by methods such as check sums and the like.

DATA BITS

The data bits represent the actual data being transmitted. These are typically 8 bits, but may also be 7 bits for systems using the 7 bit ASCII code. If 7 Data bits are used, then the eighth bit is used as the parity bit. No parity can be used if eight data bits are specified.
STOP BIT

Stop bits denote the end of the transmission of the current byte. Stop bits are typically 1, 1.5 or 2 bits long. Certain software does not give the user independence in deciding the number of legal stop bits, but set it to be the same as the number of start bits.

DUPLEX SETTINGS AND ECHO

Another parameter to be considered in data communication is Duplex. Theoretically, Simplex denotes data transmission in one direction only. Half duplex indicates that a two way transmission is taking place but not simultaneously. Full duplex indicates that the Computer and the device communicate simultaneously, that is, transmission and reception goes on simultaneously. In a practical sense, If a computer is set at Full duplex, the character that is transmitted by the computer is received by the device and is echoed back for verification. It is this echo that is used by the computer to verify its video display. On half duplex, the character is sent to the TxD line but no echo is received from the receiver. Thus, a parameter Local Echo is set ON to update the video display. Double characters on the display indicate that Full duplex communication is in progress and the Local
echo is on as well. The normal remedy for this situation is to turn Echo off.

SUMMARY

To summarize the above discussion, several factors must be considered to make devices communicate using the RS 232 specification. The first consideration is if hardware handshaking is required or can be omitted without any effect on communication. To simplify communication, first preference must be given to omitting hardware handshaking or emulating it through software wherever possible. The format of the byte package is to be decided upon. This includes, setting the baud rate, the number of start bits, data bits, stop bits and the duplex mode. Most of these parameters may be selected on the device and the computer end and the manufacturer normally specifies the right combinations in the users manual for the device. This concludes a brief definition of the several RS 232 parameters. This is by no means comprehensive since the RS 232 protocol has considerable variations. A complete specification is available from

EIA Engineering Department
Standard Sales
2001 Eye Street N.W.
Washington D.C. 20006
(202) 457 4966
APPENDIX D
CNC CODE DESCRIPTION

G CODES

Before the working of the software is discussed, an explanation of the CNC programming language is given here. Any numerically controlled or Computer Numerically Controlled machine is programmed in G, M and N commands. G commands, or G codes as they are commonly referred to are those functions that change the mode of operation of an N/C system. G codes are specified by the EIA and even though each manufacturer may expand on them as they see fit, the base remains the same. This is like the ASCII character set. The numbers 0 to 127 decimal have a predefined character assigned to them by the American Standards Institute, but the numbers 128 to 255 may be designated by the manufacturer according to the features desired. A complete list of the G codes are given in table one.

TABLE 1:

<table>
<thead>
<tr>
<th>CODE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>G00-G03</td>
<td>Reserved for contouring only</td>
</tr>
<tr>
<td>G04</td>
<td>Dwell</td>
</tr>
<tr>
<td>G5</td>
<td>Hold</td>
</tr>
<tr>
<td>G06-G07</td>
<td>Unassigned</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>G08-G12</td>
<td>Reserved for contouring only</td>
</tr>
<tr>
<td>G13-G16</td>
<td>Axis selection</td>
</tr>
<tr>
<td>G17-G21</td>
<td>Reserved for contouring only</td>
</tr>
<tr>
<td>G22-G24</td>
<td>Unassigned</td>
</tr>
<tr>
<td>G25-G29</td>
<td>Permanently unassigned</td>
</tr>
<tr>
<td>G30-G32</td>
<td>Reserved for contouring only</td>
</tr>
<tr>
<td>G33</td>
<td>Thread cutting, constant lead</td>
</tr>
<tr>
<td>G34-G35</td>
<td>Reserved for contouring only</td>
</tr>
<tr>
<td>G36-G39</td>
<td>Reserved for control use only</td>
</tr>
<tr>
<td>G40-G49</td>
<td>Reserved for contouring only</td>
</tr>
<tr>
<td>G50-G79</td>
<td>Unassigned</td>
</tr>
<tr>
<td>G80</td>
<td>Fixed cycle cancel</td>
</tr>
<tr>
<td>G81</td>
<td>Fixed cycle # 1</td>
</tr>
<tr>
<td>G82</td>
<td>Fixed cycle # 2</td>
</tr>
<tr>
<td>G83</td>
<td>Fixed cycle # 3</td>
</tr>
<tr>
<td>G84</td>
<td>Fixed cycle # 4</td>
</tr>
<tr>
<td>G85</td>
<td>Fixed cycle # 5</td>
</tr>
<tr>
<td>G86</td>
<td>Fixed cycle # 6</td>
</tr>
<tr>
<td>G87</td>
<td>Fixed cycle # 7</td>
</tr>
<tr>
<td>G88</td>
<td>Fixed cycle # 8</td>
</tr>
<tr>
<td>G89</td>
<td>Fixed cycle # 9</td>
</tr>
<tr>
<td>G90-G99</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

(Source: Biekert and Zeigler; see reference). It has been coordinated with the corresponding table in EIA standard RS274, interchangeable perforated tape variable block format
for contouring and contouring/positioning numerically controlled machine tools. Additional commands may be programmed by the use of unassigned code numbers. Codes G50 to G79 should be assigned first. Such additional specific assignments must be listed on the format classification sheet.

The generic format of the G code is

\[ GnXcYcZcFf \]

In the above example, G is the command to the NC/CNC machine to perform a certain task. The integer number, n, that follows it defines the task. The task is basically a move from one point to another, or a series of moves. For example, a G0 indicates a rapid move from the current position to a newly specified X, Y, Z position. Such rapid moves are frequently made to move the tool to the start position before beginning the machining, moving to a clear point to enable a tool change, and so on. A G1 is a linear move at a specified feed rate. Thus if the user wishes to move in a straight line from the current position to the specified position, he enters a G1 followed by the X, Y, Z coordinates and the feed rate, or the rate at which the cutter is to transverse when moving between the two points. Similarly, a G2 is a Circular move in the clockwise direction in the x-y
plane, and a G3 is a circular move in the counter clockwise direction in the x-y plane.

The manufacturer may choose to define his own codes at a higher value to invoke canned cycles. Canned cycles are a predefined series of moves that are executed automatically when the appropriate G code is specified. An example of this is the commands G81-G87, G89 on the Bridgeport, that are defined to perform various drilling operations.

M CODES

M codes are command codes that define miscellaneous functions. Miscellaneous commands are defined as those functions that prepare the system for operation. These functions are modal, in that once set, they remain unchanged until changed by another M function. Also, M functions are sometimes acted upon before an X, Y, Z move, during an X, Y, Z move or after the move, depending on the nature of the function. The exact nature of these functions depends on the system being used. Given in Table 2 is a complete list of M codes and their functions. (Source: Biekert and Zeigler; See reference). It has been coordinated with the corresponding table in EIA standard RS274, interchangeable perforated tape variable block format for contouring and contouring/positioning numerically controlled machine tools. Additional commands may be programmed by the use of
unassigned code numbers.

<table>
<thead>
<tr>
<th>CODE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>M00</td>
<td>Program stop</td>
</tr>
<tr>
<td>M01</td>
<td>Optional (planned stop)</td>
</tr>
<tr>
<td>M02</td>
<td>End of Program</td>
</tr>
<tr>
<td>M03</td>
<td>Spindle Clockwise (CW)</td>
</tr>
<tr>
<td>M04</td>
<td>Spindle counterclockwise (CCW)</td>
</tr>
<tr>
<td>M05</td>
<td>Spindle off</td>
</tr>
<tr>
<td>M06</td>
<td>Tool change</td>
</tr>
<tr>
<td>M07</td>
<td>Coolant #2 ON</td>
</tr>
<tr>
<td>M08</td>
<td>Coolant #1 ON</td>
</tr>
<tr>
<td>M09</td>
<td>Coolant OFF</td>
</tr>
<tr>
<td>M10</td>
<td>Clamp</td>
</tr>
<tr>
<td>M11</td>
<td>Unclamp</td>
</tr>
<tr>
<td>M12</td>
<td>Unassigned</td>
</tr>
<tr>
<td>M13</td>
<td>Spindle CW &amp; coolant ON</td>
</tr>
<tr>
<td>M14</td>
<td>Spindle CCW and Coolant ON</td>
</tr>
<tr>
<td>M15</td>
<td>Motion +</td>
</tr>
<tr>
<td>M16</td>
<td>Motion -</td>
</tr>
<tr>
<td>M17-24</td>
<td>Unassigned</td>
</tr>
<tr>
<td>M25-29</td>
<td>Permanently unassigned</td>
</tr>
<tr>
<td>M30</td>
<td>End of tape</td>
</tr>
<tr>
<td>M31</td>
<td>Inter lock Bypass</td>
</tr>
<tr>
<td>M32-M35</td>
<td>Constant cutting speed</td>
</tr>
<tr>
<td>M38-M39</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>
M49-M45  Gear changes, if used
M46-M49  Reserved for control use only
M50-M99  unassigned

**F CODE**
The F code is used to define the feed rate or the rate at which the table moves to provide the correct cutting velocity for the material being machined. This feed rate is specified in inches per minute and is dependant on the material being machined.

**T CODE**
The T code is used to allow the program to directly access different tools in the automatic tool changers. If there is no tool changer, but a tool change is specified, the machine returns to a previously defined clear position and waits for the operator to change the tool manually and re-enable the machining of the part.

**S CODE**
The S code is used to specify the rotational speed of the spindle. The speed in specified in revolutions per minute (RPM).
APPENDIX E
LIST OF REFERENCES

CNC Programming - Bridgeport Machines Inc.

Macintosh Pascal Technical Appendix, by Hueras, Jon, Think Technologies, Cris Espinosa, Carol Rose, Apple Computer Inc., - (C) 1984 - Think Technologies Inc. 420 Bedford Street, Lexinton, MA 02173, (C) 1984 Apple Computer Inc. 20525 Mariani Ave, Cupertino, CA 95014

Macintosh Pascal Reference Manual, by Hueras, Jon, Think Technologies, Cris Espinosa, Carol Rose, Apple Computer Inc., - (C) 1984 - Think Technologies Inc. 420 Bedford Street, Lexinton, MA 02173, (C) 1984 Apple Computer Inc. 20525 Mariani Ave, Cupertino, CA 95014

Mastering the Macintosh Toolbox, Peatroy, David, B, and Datatech Publications, - (C) Osborne, McGraw Hill Publishers, 2600 Tenth Street, Berkeley, CA 94710

LightSpeed Pascal - User's Guide and Reference Manual - By the Think Technology team - (C) 1986 - Think Technologies Inc., 420 Bedford Street, Lexinton, MA 02173

Versaterm™ for the Macintosh - (C) Lonnie R. Abelbeck, Distributed by Peripherals Computers and supplies Inc., 2457 Perkiomen Avenue, Mt. Penn, PA 19606.


Macintosh Hands - On Pascal, by Lewis T.G., Abbas Birjandi, - (C) 1986 by Wadsworth Publishing Company, Belmont, CA 94002

Technical Introduction to the Macintosh Family, a reference manual, (C) Apple Computer Inc., 20525 Mariani Avenue, Cupertino, CA 95014, Published by Addison Wesley Publishing Company, New York


Learning to Program, Macintosh Pascal Supplement, by Urieli, Israel and Noori, Andy, (C) 1987 by Kendall/Hunt Publishing Company, Dubuque, Iowa

Macintosh Pascal User's Guide, (C) Apple Computer Inc., 20525 Mariani Ave, Cupertino, CA 95014

Macintosh Pascal 2.0 Update, (C) Apple Computer Inc., 20525 Mariani Ave, Cupertino, CA 95014

Macintosh Revealed, Volume one, Unlocking the Toolbox, By Chernicoff, Stephen - (C) 1985 by the Hayden Book Company, Berkeley, California

Introduction to Macintosh Pascal, By Simonoff, Johnathan, D., - (C) 1985 by Simonoff, Jonathan, D., Published by the Hayden Book Company, Berkeley, California
Hidden Powers of the Macintosh, By Morgan, Cristopher, L., (C) 1985 by the Waite Group, New York

Fundamental Programming With Pascal, By Starkey, Denbigh, J. and Ross, Rockford, J. - (C) 1984 by West Publishing Company, 50 West Kellog Boulevard, P.O. Box 43526, St Paul, Minnesota 55165

Programmer's Problem Solver for the IBM PC, XT & AT, by Jourdain, Robert - (C) 1985 by Brady Communications Inc., published by Prentice Hall press, One Gulf + Western Plaza, New York, New York, 10023

Introductory Methods of Numerical Analysis, by Sastry S.S., (C) 1977 by Prentice Hall of India Private Limited, M-97 Connaught Circus, New Delhi 110001

Numerical Control Student Manual, by Biekert, Russ and Ziegler, Lyle - (C) 1974 by Engineering Technology Inc., 503 East Main Street, Mahomet, IL 61853

Plane and Solid Analytic Geometry, by William F. Osgood, and William C. Graustien, (C) the Macmillian Company, New York


Fundamentals of Interactive Computer Graphics, By Foley, J.D. and Dam, Van, A. - (C) 1982 by the Addison Wesley Publishing Company, Menlo Park, California

Computer Graphics and Applications, by Dennis Harris, (C) 1984 by Dennis Harris, Published by Chapman And Hall, London, England.


Macintosh Pascal Extender, by Juni, Jack, (C) 1985 by invention Software Corporation, Ann Arbor, MI 48106

MacUser, July 1988 - Special report on Desktop Engineering, CAD/CAM/Manufacturing with EZ-Mill - (C) Ziff Davis Publishing Company, One Park Avenue, New York, New York, 10016

MacTutor, The Macintosh Programming Journal, Vol.4, No.6, June 1988, (C) by MacTutor, P.O. Box 400, Placentis, CA 92670
APPENDIX F

VARIABLE LIST IN MACCAMGLOBAL

maxentity : maximum number of entities that can be stored in the array DAT

DAT : Array of dimension Maxentity,7; is the main data storage of the entire program; stores the entire geometry and is used to generate the G codes. It is indexed by the variable U that keeps track of the number of Entities existant so far.

TOOLINFO : This array keeps track of tool changes, spindle, feed and plunge speeds when chosen. This is indexed by the variable V, that is updated at each tool change.

PENCIL : description for the hourglass cursor, signifies the program is doing something and not polling the mouse

HAND : Pointing hand cursor definition variable

CROSS : Cross hair cursor definition variable

REC : of TYPE rectangle, used as a general purpose variable to define ovals, frames
\[ x,y \quad : \quad \text{General purpose Indicator variables} \\
i \quad : \quad \text{Loop Index variable} \\
xzero,yzero \quad : \quad X,Y \text{ position of the part origin. All computations in inches are made relative to this zero position} \\
gridsize \quad : \quad \text{Variable defining grid size as selected by the user} \\
zpixel \quad : \quad \text{The current Z depth in Pixel values} \\
u \quad : \quad \text{Index variable for DAT} \\
v \quad : \quad \text{Index variable for TLNO} \\
feed \quad : \quad \text{Feed rate for the Bridgeport} \\
spndl \quad : \quad \text{Spindle speed of the Bridgeport} \\
plunge \quad : \quad \text{Plunge speed on the Bridgeport} \\
tool \quad : \quad \text{Current tool number} \\
ssize \quad : \quad \text{Current Snapsize} \\
xon,yon \quad : \quad \text{Variables indicating cutter compensation turn on point} \\
xcor,ycor \quad : \quad \text{variables to indicate the position of the cursor in inches at any given time on the display} \\
tooldia \quad : \quad \text{The tooldiameter of the current tool being used} \\
snapreply \quad : \quad \text{Character variable indicating snap status}

\[ ^1 \text{32 Pixels make one inch} \]
<table>
<thead>
<tr>
<th>SNAP</th>
<th>: Boolean indication of snap</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td>: Boolean indicating result of the YESNO procedure</td>
</tr>
<tr>
<td>COMP</td>
<td>: Boolean indicating if Cutter compensation is turned on</td>
</tr>
<tr>
<td>USERNAME</td>
<td>: Boolean indicating if the load or save routine was called by the user or within the program for housekeeping jobs</td>
</tr>
<tr>
<td>Options File (by segment)</td>
<td>Size</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>MacPasLib</td>
<td>15842</td>
</tr>
<tr>
<td>MacTraps</td>
<td>5440</td>
</tr>
<tr>
<td>MacCamglobals</td>
<td>0</td>
</tr>
<tr>
<td>savedraw</td>
<td>566</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Segment 1</strong></td>
<td>21852</td>
</tr>
<tr>
<td>MiscellaneousProcs</td>
<td>1686</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Segment 2</strong></td>
<td>1686</td>
</tr>
<tr>
<td>MoreProcs</td>
<td>5956</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Segment 3</strong></td>
<td>5956</td>
</tr>
<tr>
<td>bridgeport</td>
<td>6694</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Segment 4</strong></td>
<td>6694</td>
</tr>
<tr>
<td>Frontview</td>
<td>8814</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Segment 5</strong></td>
<td>8814</td>
</tr>
<tr>
<td>Mac-Cam19LP1</td>
<td>19494</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Segment 6</strong></td>
<td>19494</td>
</tr>
</tbody>
</table>
unit MacCamglobals;

interface

const
  maxentity = 125;

type
dtarray = array[1..maxentity, 1..7] of integer;
toolinfo = array[1..24, 1..4] of integer;

var
  pencil, hand, cross : cursor;
dat : dtarray;
tlno : toolinfo;
rec : rect;
x, y, i, xzero, yzero, gridsize, zpixel, u, v, tool, toolinfo, xon, xoff, yon : integer;
xcor, ycor, tooldia : real;
snapreply : char;
snap, result, comp, username : boolean;

implementation
end.
unit savedraw;
interface
uses
  MacCamglobals;

procedure saveall (username : boolean;
  var snap : boolean;
  var snapsize, gridsize, feed, xzero, yzero, u, v : integer;
  var tino : toolinfo;
  var dat : dtarray);

implementation

procedure saveall;

var
datafile : text;
filename : string;
i, j : integer;
begin
  If username then
    begin
      writeln('Enter the filename ');
      readln(filename);
    end
  else
    filename := 'temp';
  setcursor(pencil);
  rewrite(datafile, filename);
  writeln(datafile, snap, snapsize, gridsize, feed, xzero, yzero, u, v);
  for i := 1 to u do
    begin
      for j := 1 to 7 do
        begin
          write(datafile, dat[i, j]);
        end;
    end;
  for i := 1 to v do
    begin
      for j := 1 to 3 do
        begin
          write(datafile, tino[i, j]);
        end;
    end;
end.
unit MiscellaneousProcs;

interface
uses
  MacCamglobals;

procedure inithand;
procedure initcross;
procedure initpencil;
procedure clearall;
procedure erasedialog;
procedure drawscale;
procedure drawzicon;
procedure drawdialog;

implementation

procedure drawdialog;
begin
  pensize(3, 3);
  frameroundrect(50, 400, 100, 490, 25, 25);
  frameroundrect(120, 400, 170, 490, 25, 25);
  pensize(1, 1);
  frameroundrect(125, 405, 165, 485, 20, 20);
  frameroundrect(55, 405, 95, 485, 20, 20);
end;  {Draw Dialog}

{CURSOR INIT ROUTINES}

procedure inithand;
var
  hot : point;
i : integer;
begin
  hand.data[0] := $0000;
  hand.data[1] := $0000;
  hand.data[2] := $0000;
  hand.data[3] := $0000;
  hand.data[5] := $1900;
  hand.data[7] := $4200;
  hand.data[8] := $c7fe;
  hand.data[9] := $8c01;
  hand.data[10] := $97fe;
  hand.data[12] := $87f0;
  hand.data[14] := $c7e0;
  hand.data[15] := $7f80;
  hand.mask[0] := $0000;
  hand.mask[1] := $0000;
  hand.mask[2] := $0000;
  hand.mask[3] := $0000;
hand.mask[5] := $1F00;
hand.mask[6] := $3E00;
hand.mask[7] := $7E00;
hand.mask[8] := $FFFF;
hand.mask[9] := $FFFF;
hand.mask[10] := $FFFF;
hand.mask[12] := $FFFF;
hand.mask[14] := $FFFF;
hand.hotspot.h := 16;
hand.hotspot.v := 9;
end;

procedure initcross;
var
hot : point;
i : integer;
begin
cross.data[0] := $0080;
cross.data[1] := $0080;
cross.data[7] := $0080;
cross.data[8] := $FFFF;
cross.data[9] := $0080;
cross.data[10] := $0080;
cross.data[12] := $0080;
cross.data[14] := $0080;
cross.mask[0] := $0080;
cross.mask[1] := $0080;
cross.mask[8] := $FFFF;
cross.mask[9] := $0080;
cross.mask[10] := $0080;
cross.mask[12] := $0080;
cross.mask[14] := $0080;
cross.hotspot.h := 8;
procedure initpencil;
  var
    hot : point;
    i : integer;
  begin
    pencil.data[0] := $0000;
    pencil.data[1] := $1FF8;
    pencil.data[4] := $0BF0;
    pencil.data[5] := $05A0;
    pencil.data[6] := $04A0;
    pencil.data[7] := $05A0;
    pencil.data[8] := $05A0;
    pencil.data[9] := $05A0;
    pencil.data[10] := $05A0;
    pencil.data[12] := $13C8;
    pencil.data[14] := $4FF2;
    pencil.data[15] := $3FFC;
    pencil.mask[0] := $0000;
    pencil.mask[1] := $1FF8;
    pencil.mask[4] := $0BF0;
    pencil.mask[5] := $05A0;
    pencil.mask[6] := $04A0;
    pencil.mask[7] := $05A0;
    pencil.mask[8] := $05A0;
    pencil.mask[9] := $05A0;
    pencil.mask[10] := $05A0;
    pencil.mask[12] := $13C8;
    pencil.mask[14] := $4FF2;
    pencil.mask[15] := $3FFC;
    pencil.hotspot.h := 7;
    pencil.hotspot.v := 16;
  end;

procedure clearall;
begin
  penpat(white);
  paintrect(-1, -1, 235, 496);
end;

procedure erasedialog;
begin
  penpat(white);
paintrect(1, 400, 233, 497);
penpat(black);
end; {erasedialog}

procedure drawscale;
const
disp := 32;
var
stp, i : integer;
begin
stp := 4;
moveto(470, 35);
line(0, 128);
moveto(470, 35);
line(9, 0);
for i := 1 to 32 do
begin
moveto(470, 35 + stp);
if (i / 4) = round(i / 4) then
if (i / 8) = round(i / 8) then
line(9, 0)
else
line(6, 0)
else
line(3, 0);
stp := -stp + 4;
end;
textsize(8);
moveto(482, 39);
writtenraw("0");
moveto(482, 39 + disp);
writtenraw("1");
moveto(482, 39 + 2 * disp);
writtenraw("2");
moveto(482, 39 + 3 * disp);
writtenraw("3");
moveto(482, 39 + 4 * disp);
writtenraw("4");
pensize(1, 1);
end; {DrawScale}

procedure drawzicon;
begin
("Z" Begins)
moveto(4, 114);
line(6, 0);
line(-6, 6);
line(6, 0);
moveto(15, 114);
line(0, 14);
line(3, -3);
moveto(-3, 3);
line(-3, -3);
(Z ends)
   end;(draw z icon)

end.
unit MoreProcs;

Interface
uses
  MacCamglobals, miscellaneousProcs, savedraw;

procedure recdraw (rec : rect;
  radius, ofset, pnsize : integer;
  recpat : pattern);
procedure plotdot (grx, gry : integer);
procedure plotgrid (gridsize : integer);
procedure drawbox (x, y, boxsize : integer);
procedure setup (var x, y, gridsize : integer);
procedure yesno (var result : boolean);
procedure loaddrawing (username : boolean;
  var snap : boolean;
  var snapsize, gridsize, feed, xzero, yzero, u, v : integer;
  var tinfo : toolinfo;
  var dat : dtarray);

Implementation

procedure recdraw;
  var
    arc : rect;
    xl, yl, dia : integer;
  begin
    pensize(pnsize, pnsize);
    penpat(recpat);
    dia := radius * 2;
    xl := rec.right - rec.left - (2 * ofset);
    yl := rec.bottom - rec.top - (2 * ofset);
    moveto(rec.left + ofset, rec.top + ofset);
    line(xl, 0);
    line(0, yl);
    line(-xl, 0);
    line(0, -yl);
    penpat(white);
    moveto(rec.left + ofset, rec.top + ofset);
    line(0, radius - 1 - ofset);
    moveto(rec.left + ofset, rec.top + ofset);
    line(radius - 1 - ofset, 0);
    moveto(rec.right - ofset, rec.top + ofset);
    line(1 - radius + ofset, 0);
    moveto(rec.right - ofset, rec.top + ofset);
    line(0, radius - 1 - ofset);
    moveto(rec.right - ofset, rec.bottom - ofset);
    line(1 - radius + ofset, 0);
    moveto(rec.right - ofset, rec.bottom - ofset);
    line(0, 1 - radius + ofset);
    moveto(rec.left + ofset, rec.bottom - ofset);
line(radius - 1 - offset, 0);
moveto(rec.left + offset, rec.bottom - offset);
line(0, 1 - radius + offset);
penpat(reepat);
arc.top := rec.top + offset;
arc.left := rec.left + offset;
arc.bottom := (rec.top + dia - (2 * offset));
arc.right := (rec.left + dia - (2 * offset));
framearc(arc, 0, -90);
arc.top := rec.top + offset;
arc.left := (rec.right - dia + offset);
arc.bottom := rec.top + dia - offset;
arc.right := rec.right - offset;
framearc(arc, 0, 90);
arc.left := rec.left + offset;
arc.top := rec.bottom - dia + offset;
arc.bottom := rec.bottom - offset;
arc.right := rec.left + dia - offset;
framearc(arc, -90, -90);
arc.left := rec.right - dia + offset;
arc.top := arc.bottom - dia + offset;
arc.bottom := rec.right - offset;
arc.right := rec.right - offset;
framearc(arc, 90, 90);
pensize(1, 1);
end: {drawframe}

procedure plotdot;
begin
  moveto(grx, gry);
  line(0, 0);
end:{plotdot}

procedure plotgrid;
var
  grx, gry : integer;
begin
{set up grid}
  grx := 48;
  gry := 0;
  repeat
    repeat
      plotdot(grx, gry);
      grx := grx + gridsize;
    until grx >= 400;
    grx := 48;
    gry := gry + gridsize;
  until gry >= 242;
end; {plotgrid}

procedure drawbox;
begin
moveto(x, y);
line(boxsize, 0);
line(0, boxsize);
line(-boxsize, 0);
line(0, -boxsize);
end; {drawBox}

procedure setup;
var
grx, gry, boxsize, i, j : integer;
begin
penpat(black);
boxsize := 20;
textsize(18);
repeat
repeat
drawbox(x, y, boxsize);
x := x + 22;
until x > 23;
x := 1;
y := y + 22;
until y > 155;
{Line Begins}
moveto(5, 18);
lineto(18, 5);
{Line Ends}
{Circle Begins}
frameoval(5, 27, 18, 40);
{Circle Ends}
{redraw Begins}
Moveto(5, 40);
drawchar('R');
{redraw Ends}
{drill Begins}
frameoval(27, 27, 40, 40);
moveto(33, 29);
line(0, 8);
moveto(29, 33);
line(8, 0);
{drill Ends}
{'Tool change' Begins}
moveto(28, 63);
drawchar('T');
{'Tool change' Ends}
{frame mill begins}
moveto(6, 48);
line(10, 0);
line(0, 14);
line(-10, 0);
line(0, -14);
[frame mill ends]

['Save' Begins]
moveto(7, 85);
drawchar('S');
['Save' Ends]

['Load' Begins]
moveto(29, 85);
drawchar('L');
['Load' Ends]

['New begins']
moveto(27, 128);
drawchar('N');
['New' ends]

['Snap Begins']
penpat(ltgray);
paintrect(135, 25, 151, 41);
penpat(black);
['Snap Ends']

[Front View Begins]

penpat(gray);
moveto(2, 165);
lineto(19, 165);
penpat(black);
frameoval(160, 1, 170, 11);
frameoval(162, 3, 168, 9);
moveto(12, 160);
lineto(12, 170);
lineto(19, 168);
lineto(19, 162);
lineto(12, 160);
[End Front View]

[G Begins]
moveto(6, 107);
drawchar('G');
[G Ends]

[E Begins]
moveto(29, 107);
drawchar('E');
[E Ends]

drawzicon;
procedure yesno;
  var
    xf, yf : integer;
    choice : boolean;
  begin
    choice := false;
    drawdialog;
    moveto(426, 80);
    drawstring(' YES ');
    moveto(421, 150);
    drawstring(' NO ');
    penpat(black);
    sysbeep(8);
    sysbeep(8);
    sysbeep(8);
    setcursor(hand);
    while not choice do
      begin
        repeat
          until Button;
        repeat
          getmouse(xf, yf);
          until not button;
        if (xf > 405) and (xf < 485) and (yf > 125) and (yf < 165) then
          begin
            result := false;
            choice := true;
          end
        else if (xf > 405) and (xf < 485) and (yf > 55) and (yf < 95) then
          begin
            result := true;
            choice := true;
          end;
        if not choice then
          sysbeep(8);
      end;
    erasedialog;
    setcursor(cross);
  end;

procedure loaddrawing;
  var
    datafile : text;
    filename : string;
i, j, radius, psize, offset, xhole, yhole, rhole : integer;
rec, drawhole : rect;
recpat : pattern;

begin
if username then
begin
if u > 1 then
begin
  textsize(10);
moveto(400, 30);
textface([bold]);
drawstring("Save Drawing??");
textsize(12);
textface([]);
yesno(result);
end
else
result := false;
if result then
begin
  username := true;
saveall(username, snap, snapsize, gridsize, feed, xzero, yzero, u, v, tino, dat);
end;
writeln("Enter the name of the file to load ");
readln(filename);
end
else
  filename := 'temp';
clearall;
moveto(30, 200);
textsize(12);
drawstring("Loading file, Please wait....... ");
setcursor(pencil);
reset(datafile, filename);
readln(datafile, snap, snapsize, gridsize, feed, xzero, yzero, u, v);
for i := 1 to u do
  begin
    for j := 1 to 7 do
      begin
        read(datafile, dat[i, j]);
      end
  end;
for i := 1 to v do
  begin
    for j := 1 to 3 do
      begin
        read(datafile, tino[i, j]);
      end
  end;
xhole := 1;
yhole := 1;
clearall;
setup(xhole, yhole, gridsize);
pensize(1, 1);
for i := 1 to u do
  begin
    writeln(dat[i, 1]);
    if dat[i, 1] = 0 then
      begin
        moveto(dat[i, 2], dat[i, 3]);
      end
    else if dat[i, 1] = 1 then
      begin
        if (dat[i, 7] = 1) then
          penpat(gray)
        else
          penpat(black);
        pensize(1, 1);
        lineto(dat[i, 2], dat[i, 3]);
        penpat(black);
      end
    else if (dat[i, 1] = 172) or (dat[i, 1] = 171) or (dat[i, 1] = 170) then
      begin
        rec.left := dat[i, 2];
        rec.top := dat[i, 3];
        rec.right := dat[i, 4];
        rec.bottom := dat[i, 5];
        radius := dat[i, 6];
        if dat[i, 1] = 172 then
          begin
            ps := 2;
            recpat := dgray;
            ofset := 3;
            recdraw(rec, radius, ofset, ps, recpat);
            ofset := 0;
            recpat := black;
            ps := 1;
            recdraw(rec, radius, ofset, ps, recpat);
          end
        else if dat[i, 1] = 171 then
          begin
            ps := 1;
            recpat := gray;
            ofset := 3;
            recdraw(rec, radius, ofset, ps, recpat);
            ofset := 0;
            recpat := black;
            ps := 1;
            recdraw(rec, radius, ofset, ps, recpat);
          end
        else if dat[i, 1] = 170 then
          begin
            ofset := 0;
            recpat := black;
            ps := 1;
          end
      end
    else
      begin
        penpat(gray);
        pensize(1, 1);
        lineto(dat[i, 2], dat[i, 3]);
        penpat(black);
      end
  end
recdraw(rec, radius, offset, pnsize, recpat);
  pnsize := 1;
  recpat := gray;
  offset := -3;
  recdraw(rec, radius, offset, pnsize, recpat);
end;
penpat(black);
end
else if (dat[i, 1] = 81) or (dat[i, 1] = 83) then
begin
  xhole := dat[i, 2];
  yhole := dat[i, 3];
  rhole := round(dat[i, 5] / 10000 * 32 / 2);
  drawhole.top := yhole - rhole;
  drawhole.left := xhole - rhole;
  drawhole.right := xhole + rhole;
  drawhole.bottom := yhole + rhole;
  penpat(black);
  pensize(1, 1);
  frameoval(drawhole);
  penpat(gray);
  moveto(xhole, yhole - 2);
  lineto(xhole, yhole + 2);
  moveto(xhole - 2, yhole);
  lineto(xhole + 2, yhole);
end
else if (dat[i, 1] = 177) or (dat[i, 1] = 176) or (dat[i, 1] = 175) then
begin
  offset := 3;
  xhole := dat[i, 2];
  yhole := dat[i, 3];
  radius := dat[i, 4];
  rec.left := xhole - radius;
  rec.right := xhole + radius;
  rec.top := yhole - radius;
  rec.bottom := yhole + radius;
  penpat(black);
  frameoval(rec);
if dat[i, 1] = 177 then
begin
  penpat(dkgray);
  pensize(2, 2);
  rec.left := xhole - radius + offset;
  rec.right := xhole + radius + offset;
  rec.top := yhole - radius + offset;
  rec.bottom := yhole + radius - offset;
  frameoval(rec);
end
else if dat[i, 1] = 176 then
begin
  pensize(1, 1);
  penpat(gray);
pensize(1, 1);
rec.left := xhole - radius + offset;
rec.right := xhole + radius - offset;
rec.top := yhole - radius + offset;
rec.bottom := yhole + radius - offset;
frameoval(rec);
end

else if dat[i, 1] = 175 then
begin
 dat[u, 1] := 175;
penpat(gray);
pensize(1, 1);
rec.left := xhole - radius + offset;
rec.right := xhole + radius + offset;
rec.top := yhole - radius + offset;
rec.bottom := yhole + radius + offset;
frameoval(rec);
end (This inner if)
end; (The big grandad if)
end; (The for - do, u times)
end; (procedure loaddrawing)
end.
unit Bridgeport;

interface
uses
maceamglobals;

procedure post (zpixel, xzero, yzero, u, v, feed, plunge: integer;
                 dat : dtarray;
                 tinfo : toolinfo);

implementation

procedure post;
const
colon = ':';
var
tooldia, maxblnk, xinc, x, y, z, x1, y1, x2, y2, xc, yc, xl, yl, rad, xci, yci : real;
cntr, toolno, i, j, tcnt, ci : integer;
gfile : text;
name : string;

begin
  writeln('Enter Bridgeport Program name ');
  readln(name);
tcnt := 0;
entr := 0;
rewrite(gfile, 'Gfile');
write(gfile, chr(18));
write(gfile, colon, name);
write('Beginning processing of the Gfile');
write('Please wait ......');

{GFILE PROCESSING IS TO BE ENTERED HERE}

for i := 1 to u do
begin
  writeln('DAT G = ', dat[i, 1]);
  if (dat[i, 1] = 0) or (dat[i, 1] = 1) then
  begin
    x := (dat[i, 2] - xzero) / 32;
    y := (yzero - dat[i, 3]) / 32;
    z := dat[i, 4] / 32;
    if (dat[i, 1] = 0) then
      begin
        cntr := cntr + 1;
        cntr := cntr + 1;
        write(gfile, 'N', cntr : 1, 'G1Z', z : 6 : 4, 'F', plunge : 1, ' ');
      end
    else
      begin
        cntr := cntr + 1;
      end
  end
end
begin
  cntr := cntr + 1;
  x := (dat[i, 2] - xzero) / 32;
  y := (yzero - dat[i, 3]) / 32;
  rad := dat[i, 4] / 32;
  z := dat[i, 5] / 32;
end if (dat[i, 1] = 175) or (dat[i, 1] = 176) or (dat[i, 1] = 177) then
begin
end
else
begin
end
else if (dat[i, 1] = -10) then
begin
  tcnt := tcnt + 1;
  cntr := cntr + 1;
  feed := tno[tcnt, 4];
  plunge := round(feed / 2);
end
else if ((dat[i, 1] = 170) or (dat[i, 1] = 171) or (dat[i, 1] = 172)) then
begin
  cntr := cntr + 1;
  x1 := (dat[i, 2] - xzero) / 32;
  y1 := (yzero - dat[i, 3]) / 32;
  x2 := (dat[i, 4] - xzero) / 32;
  y2 := (yzero - dat[i, 5]) / 32;
  x1 := x2 - x1;
  y1 := y1 - y2;
  xc := x1 + (x1 / 2);
  yc := y2 + (y1 / 2);
  rad := dat[i, 6] / 32;
  z := dat[i, 7] / 32;
end if (dat[i, 1] = 170) or (dat[i, 1] = 171) then
begin
end
else if (dat[i, 1] = 172) then
begin
end {end for 170 or 171 or 172}
end {End for 171, 172, or 170}
else if (dat[i, 1] = 81) or (dat[i, 1] = 83) then
begin
x1 := (dat[i, 2] - xzero) / 32;
y1 := (yzero - dat[i, 3]) / 32;
z := dat[i, 4] / 32;
cntr := cntr + 1;
writeln(gfile, 'N', cntr : 1, 'G0X', x1 : 6 : 4, 'Y', y1 : 6 : 4, 'Z0.05');
cntr := cntr + 1;
if (dat[i, 1] = 81) then
begin
end
else
begin
end
end else if (dat[i, 1] = 40) or (dat[i, 1] = 41) or (dat[i, 1] = 42) then
begin
x := (dat[i, 2] - xzero) / 32;
y := (yzero - dat[i, 3]) / 32;
cntr := cntr + 1;
if (dat[i, 1] = 41) or (dat[i, 1] = 42) then
begin
i := i + 2;
{ei is the turn around the corner coordinate}
writeln(gfile, 'N', cntr : 1, 'G0G', dat[i, 1] : 1, 'X', x : 6 : 4, 'Y', y : 6 : 4, 'Z0.05');
end
else if (dat[i, 1] = 40) then
begin
xci := (dat[ci, 2] - xzero) / 32;
yci := (yzero - dat[ci, 3]) / 32;
writeln(gfile, 'N', cntr : 1, 'G40G0');
end;
end; {END Big If}
end; {END FOR}
cntr := cntr + 1;
writeln(gfile, 'N', cntr : 1, 'G0X0.0Y0.0Z1.0M30');
writeln(gfile, chr(4), chr(14));
writeln('GFILE WRITTEN TO DISK SUCCESSFULLY');
end; {post - GFILE GENERATOR}
end.
unit frontview;

interface
uses
  macCamglobals, MiscellaneousProcs, savedraw, moreprocs;

procedure hangaround;
procedure pressakey (zdimension : real);
procedure done;
procedure frview (snap : boolean;
  snapsize, gridsize, feed, xzero, yzero, u, v, plunge : integer;
  dat : dtarray;
  tnlo : toolinfo);

implementation

procedure hangaround;
begin
  setcursor(pencil);
  repeat
    until button;
  repeat
    until not button;
end;

procedure pressakey;
begin
  textsize(12);
  moveto(20, 225);
  writedraw(' Z = ', zdimension : 6 : 4);
  moveto(300, 225);
  textsize(10);
  textface([bold]);
  writedraw('Click Button to Continue');
  textface([]);
  textsize(12);
  hangaround;
  penpat(white);
  paintrect(216, 18, 228, 450);
  penpat(black);
end;

procedure done;
begin
  moveto(300, 225);
  textsize(10);
  textface([bold]);
  writedraw('Done! Click Button to Return');
  sysbeep(8);
  sysbeep(8);
  textface([]);
  textsize(12);
  hangaround;
procedure frview;
const
alwaysminx := 100;
alwaysmaxy := 145;
alwayszeroz := 155; {This is always ALWAYSMAXY+10}
var
i, yold, maxx, maxy, minx, miny, rhole, xtrans, ytrans : integer;
xlength, ylength, zlength, smaxx, sminx, smaxy, sminy, smaxz, sminz : integer;
xhole, yhole, radius, pnsze, offset, previousz, previoustopx, previoustopy : integer;
xscale, yscale, scale : real;
recpat : pattern;
rec, drawhole : rect;
firstval, step : boolean;
begin
setcursor(pencil);
{find the Scale factor}
firstval := true;
maxx := 0;
maxy := 0;
minx := 0;
miny := 0;
maxz := 0;
minz := 0;
for i := 1 to u do
begin
(Consider Lines for Scale factor)

if (dat[i, 1] = 0) or (dat[i, 1] = 1) then
begin
if firstval then
begin
firstval := false;
minx := dat[i, 2];
miny := dat[i, 3];
minz := dat[i, 4];
end;
if dat[i, 2] > maxx then
maxx := dat[i, 2]
else if dat[i, 2] < minx then
minx := dat[i, 2];
if dat[i, 3] > maxy then
maxy := dat[i, 3]
else if dat[i, 3] < miny then
miny := dat[i, 3];
end;
If $\text{dat}[i, 4] > \text{maxz}$ then
\[ \text{maxz} := \text{dat}[i, 4]; \]
else if $\text{dat}[i, 4] < \text{minz}$ then
\[ \text{minz} := \text{dat}[i, 4]; \]
end;

{consider rectangles for scale factor}

If $(\text{dat}[i, 1] = 170)$ or $(\text{dat}[i, 1] = 171)$ or $(\text{dat}[i, 1] = 172)$ then
  begin
    If $\text{firstval}$ then
      begin
        firstval := false;
        minx := $\text{dat}[i, 2]$;
        miny := $\text{dat}[i, 3]$;
        minz := $\text{dat}[i, 7]$;
      end;
    If $(\text{dat}[i, 2] < \text{minx})$ then
      minx := $\text{dat}[i, 2]$;
    else if $(\text{dat}[i, 4] > \text{maxx})$ then
      maxx := $\text{dat}[i, 4]$;
    end;
    If $(\text{dat}[i, 3] < \text{miny})$ then
      miny := $\text{dat}[i, 3]$;
    else if $(\text{dat}[i, 5] > \text{maxy})$ then
      maxy := $\text{dat}[i, 5]$;
    end;
    If $\text{dat}[i, 7] > \text{maxz}$ then
      maxz := $\text{dat}[i, 7]$;
    else if $\text{dat}[i, 7] < \text{minz}$ then
      minz := $\text{dat}[i, 7]$;
  end;

{Consider Holes for Scale factor}

If $((\text{dat}[i, 1] = 81)$ or $(\text{dat}[i, 1] = 83))$ then
  begin
    rhole := round($\text{dat}[i, 5] / 10000 * 32 / 2$);
    rec.top := $\text{dat}[i, 3] - \text{rhole}$;
    rec.left := $\text{dat}[i, 2] - \text{rhole}$;
    rec.right := $\text{dat}[i, 2] + \text{rhole}$;
    rec.bottom := $\text{dat}[i, 3] + \text{rhole}$;
    If $\text{firstval}$ then
      begin
        firstval := false;
        minx := $\text{rec.left}$;
        miny := $\text{rec.top}$;
        minz := $\text{dat}[i, 4]$;
      end;
    If $(\text{rec.left} < \text{minx})$ then
      minx := $\text{rec.left}$;
    else if $(\text{rec.right} > \text{maxx})$ then
      maxx := $\text{rec.right}$;
    end;
miny := rec.top
else if (rec.bottom > maxy) then
     maxy := rec.bottom;
     If dat[i, 4] > maxz then
         maxz := dat[i, 4]
     else if dat[i, 4] < minz then
         minz := dat[i, 4];
end;

{Scale Check for Circle Mills}

If ((dat[i, 1] = 175) or (dat[i, 1] = 176) or (dat[i, 1] = 177)) then
begin
     rhole := dat[i, 4];
     rec.top := dat[i, 3] - rhole;
     rec.left := dat[i, 2] - rhole;
     rec.right := dat[i, 2] + rhole;
     rec.bottom := dat[i, 3] + rhole;
If firstval then
begin
     firstval := false;
     minx := rec.left;
     miny := rec.top;
     minz := dat[i, 5];
end;
If (rec.left < minx) then
     minx := rec.left
else if (rec.right > maxx) then
     maxx := rec.right;
If (rec.top < miny) then
     miny := rec.top
else if (rec.bottom > maxy) then
     maxy := rec.bottom;
If dat[i, 5] > maxz then
     maxz := dat[i, 5]
else if dat[i, 5] < minz then
     minz := dat[i, 5];
end;
{End the Hunt for Max Min Values}

writeln('Maxx = ', maxx, ' Minx = ', minx);
writeln('Maxy = ', maxy, ' Miny = ', miny);
writeln('Maxz = ', maxz, ' Minx = ', minz);
xlength := maxx - minx;
ylength := maxy - miny;
zlength := maxz - minz;
xscale := (390 - alwaysminx) / xlength;
yscale := (alwaysmaxy - 18) / ylength;
If (xscale > yscale) then
scale := yscale
else
scale := xscale;
writein('Scale = ', scale : 6 : 4);
writein('Xzero = ', xzero, 'Yzero = ', yzero);
Smaxx := round(scale * maxx);
Smaxy := round(scale * maxy);
Smaxz := round(scale * maxz);
Sminx := round(scale * minx);
Sminy := round(scale * miny);
Sminz := round(scale * minz);
writein(Smaxx, sminx, smaxy, smtny, smaxz, sminz);
xtrans := alwaysminx - sminx;
ytrans := alwaysmaxy - smaxy;

(Draw the scaled image)
textsize(10);
moveto(405, 30);
textface((bold));
drawstring('STEP -STEP?');
textface([]);
yesno(step);
xhole := 1;
yhole := 1;
clearall;
penpat(ltgray);
moveto(alwayszeroz);
lineto(400, alwayszeroz);
penpat(black);
pensize(1, 1);
for i := 1 to u do
begin
writeln(dat[i, 1]);
if dat[i, 1] = 0 then
begin
previousz := round(dat[i, 2] * scale + xtrans);
previousstopx := round(dat[i, 2] * scale + xtrans);
previousstopy := round(dat[i, 3] * scale + ytrans);
end
else if dat[i , 1] = 1 then
begin
if (dat[i , 7] = 1) then
penpat(gray)
exte.
else
penpat(black);
pensize(1, 1);
moveto(previousstopx, previousstopy);
lineto(round(dat[i, 2] * scale + xtrans), round(dat[i, 3] * scale + ytrans));
previousstopx := round(dat[i, 2] * scale + xtrans);
previousstopy := round(dat[i, 3] * scale + ytrans);
moveto(previousz, alwayszeroz - dat(i, 4));
lineto(round(dat(i, 2) * scale + xtrans), alwayszeroz - dat(i, 4));
previousz := round(dat(i, 2) * scale + xtrans);
If step then
  pressakey(dat[i, 4] / 32);
  penpat(black);
end
else if (dat[i, 1] = 172) or (dat[i, 1] = 171) or (dat[i, 1] = 170) then
begin
  rec.left := round(dat[i, 2] * scale + xtrans);
  rec.top := round(dat[i, 3] * scale + ytrans);
  rec.right := round(dat[i, 4] * scale + xtrans);
  rec.bottom := round(dat[i, 5] * scale + ytrans);
  radius := round(dat[i, 6] * scale);
  moveto(rec.left, alwayszeroz + dat[i, 7]);
  lineto(rec.right, alwayszeroz + dat[i, 7]);
  lineto(rec.right, alwayszeroz + dat[i, 7]);
  If dat[i, 1] = 172 then
    begin
      pnsize := 2;
      recpat := dkgray;
      ofset := 3;
      recdraw(rec, radius, ofset, pnsize, recpat);
      ofset := 0;
      recpat := black;
      pnsize := 1;
      recdraw(rec, radius, ofset, pnsize, recpat);
    end
else if dat[i, 1] = 171 then
  begin
    pnsize := 1;
    recpat := gray;
    ofset := 3;
    recdraw(rec, radius, ofset, pnsize, recpat);
    ofset := 0;
    recpat := black;
    pnsize := 1;
    recdraw(rec, radius, ofset, pnsize, recpat);
  end
else if dat[i, 1] = 170 then
  begin
    ofset := 0;
    recpat := black;
    pnsize := 1;
    recdraw(rec, radius, ofset, pnsize, recpat);
    pnsize := 1;
    recpat := gray;
    ofset := -3;
    recdraw(rec, radius, ofset, pnsize, recpat);
  end;
If step then
  pressakey(-dat[i, 7] / 32);
  penpat(black);
end
else if (dat[i, 1] = 81) or (dat[i, 1] = 83) then
begin
  xhole := round(dat[i, 2] * scale + xtrans);
  yhole := round(dat[i, 3] * scale + ytrans);
  rhole := round(((dat[i, 5] / 10000 * 32 / 2) * scale);
  drawhole.top := (yhole - rhole);
  drawhole.left := (xhole - rhole);
  drawhole.right := (xhole + rhole);
  drawhole.bottom := (yhole + rhole);
  penpat(black);
  pensize(1, 1);
  frameoval(drawhole);
  penpat(gray);
  moveto(xhole, yhole - 2);
  lineto(xhole, yhole + 2);
  moveto(xhole - 2, yhole);
  lineto(xhole + 2, yhole);
  moveto(drawhole.left, alwayszeroz);
  lineto(drawhole.left, alwayszeroz + dat[i, 4]);
  lineto(xhole, alwayszeroz + dat[i, 4] + 3);
  lineto(drawhole.right, alwayszeroz + dat[i, 4]);
  lineto(drawhole.right, alwayszeroz);
  If step then
    pressakey(-dat[i, 4] / 32);
end
else if (dat[i, 1] = 177) or (dat[i, 1] = 176) or (dat[i, 1] = 175) then
begin
  offset := 3;
  xhole := round(dat[i, 2] * scale + xtrans);
  yhole := round(dat[i, 3] * scale + ytrans);
  radius := round(dat[i, 4] * scale);
  rec.left := xhole - radius;
  rec.right := xhole + radius;
  rec.top := yhole - radius;
  rec.bottom := yhole + radius;
  penpat(black);
  frameoval(rec);
if dat[i, 1] = 177 then
begin
  penpat(dkgray);
  pensize(2, 2);
  rec.left := xhole - radius + offset;
  rec.right := xhole + radius - offset;
  rec.top := yhole - radius + offset;
  rec.bottom := yhole + radius - offset;
  frameoval(rec);
end
else if dat[i, 1] = 176 then
begin
  pensize(1, 1);
  penpat(gray);
  pensize(1, 1);
rec.left := xhole - radius + offset;
rec.right := xhole + radius - offset;
rec.top := yhole - radius + offset;
rec.bottom := yhole + radius - offset;
frameoval(rec);
end
else if dat[i, 1] = 175 then
begin
  dat[u, 1] := 175;
  penpat(gray);
  pensize(1, 1);
  rec.left := xhole - radius - offset;
  rec.right := xhole + radius + offset;
  rec.top := yhole - radius - offset;
  rec.bottom := yhole + radius + offset;
  frameoval(rec);
end; {This inner if}
moveto(rec.left, alwayszeroz + dat[i, 5]);
lineto(rec.right, alwayszeroz + dat[i, 5]);
if step then
  pressakey(-dat[i, 5] / 32);
end;
end;
done;
done;
end;
end.
program MacCam19;
uses
MacCamglobals, MiscellaneousProcs, MoreProcs, savedraw, frontview, bridgeport;
{ DINESH DHAMIJA }

procedure showxy (x, y : real);
begin
  textsize(8);
  moveto(415, 229);
  penpat(white);
  paintrect(222, 415, 230, 475);
  synch;
  penpat(black):
  writedraw(x : 6 : 3, "", y : 6 : 3, ""):
  synch;
end;(showxy)

procedure scaleselect (var pixelval : integer);
const
  disp = 32;
var
  sfp, i, x, y : integer;
  xcor, ycor : real;
  pixelsel : boolean;
begin
  setcursor(cross);
  drawscale;
  pixelsel := false;
  while not pixelsel do
    begin
      repeat
        until Button;
      repeat
        getmouse(x, y);
        xcor := (0);
        ycor := (y - 35) / 32;
        showxy(xcor, ycor);
        until not button;
      if ((x >= 469) and (x <= 480)) and ((y >= 35) and (y <= 128)) then
        begin
          pixelval := (y - 35); {35 pixel displacement}
          pixelsel := true;
        end
      else
        begin
          sysbeep(8);
        end;
    end;
end; {scaleselect}
procedure toolchange (var dat : dtarray;
    var tino : tooinfo;
    var u, v, tool, spndl, feed, plunge : integer;
    var tooldia : real;
    cutting, chipload : integer);

var
    i, x, y, tooldiapix, noteeth : integer;
    toolsel : boolean;

begin
    moveto(435, 20);
    setcursor(hand);
    textface([bold]);
    drawstring('Select');
    moveto(435, 35);
    drawstring('Tool #');
    moveto(450, 40);
    line(0, 192);
    line(10, 0);
    line(O, -192);
    line(-10, 0);
    textsize(6);
    textface([]);
    for i := 1 to 24 do
        begin
            moveto(450, 40 + (i * 8));
            line(10, 0);
            moveto(460, 36 + (i * 8));
            line(6, 0);
            moveto(460, 38 + (i * 8));
            writeln(i);
        end;
    toolsel := false;

while not toolsel do
begin
    repeat
        until Button;
    repeat
        getmouse(x, y);
        until not button;
    if (x > 450) and (x < 460) and (y > 40) and (y < 232) then
begin
    writeln(x, y);
    tool := trunc((y - 40) / 8);
    tool := tool + 1;
    writeln(tool);
    moveto(449, 40 + (tool * 8));
    pensize(3, 3);
    line(10, 0);
    line(0, -8);
    line(-10, 0);
    line(0, 8);
end;
pensize(1, 1);
for i := 1 to 1000 do
  begin
    textsiz8(12);
    penpat(white);
    paintrect(5, 420, 240, 490);
    penpat(black);
  end;

textsize(8);
textface([bold]);
moveto(420, 25);
drawstring('TOOL DIA. ');
textsize(12);
textface([]);
scaleselect(tooldiapix);
tooldia := tooldiapix / 32;
writeln('Tool diameter set to ', tooldia : 6 : 4);
writeln('Enter the number of teeth on the cutter ');
readln(noteeth);
erasedialog;
toolsel := true;
end;

{tool dia setting}
textsize(8);
textface([bold]);
moveto(420, 25);
drawstring('TOOL DIA. ');
textsize(12);
textface([]);
scaleselect(tooldiapix);
tooldia := tooldiapix / 32;
writeln('Tool diameter set to ', tooldia : 6 : 4);
writeln('Enter the number of teeth on the cutter ');
readln(noteeth);
erasedialog;
toolsel := true;
end;

else
  begin
    sysbeep(8);
  end;
end;

u := u + 1;
dat[u, 1] := -10;
sndl := round(4 * cutting / tooldia);
if sndl > 4200 then
  sndl := 4200;
feed := round(sndl * noteeth * (chipload / 1000));
plunge := round(feed / 2);

tlno[v, 1] := tool;

tlno[v, 2] := round(tooldia * 10000);
tlno[v, 3] := sndl;
tlno[v, 4] := feed;
textsize(12);
v := v + 1;  {V is updated after assignment for tlno ONLY}
setcursor(cross);
end;  {Tool Change}

procedure gridset (var gridsize : integer);
var
  i, stp, x, y : integer;
  grid : boolean;
begin
  grid := false;
stp := 4;
moveto(100, 100);
line(64, 0);
moveto(100, 100);
line(0, 6);
for i := 1 to 16 do
  begin
    moveto(100 + stp, 100);
    if (i / 4) = round(i / 4) then
      line(0, 6)
    else
      line(0, 3);
    stp := stp + 4;
  end;
setcursor(cross);
moveto(77, 75);
textface([bold]);
write('CHOOSE GRID SIZE');
textface([]);
pensize(3, 3);
frameroundrect(50, 55, 130, 220, 25, 25);
pensize(1, 1);
frameroundrect(55, 60, 125, 215, 20, 20);
moveto(97, 118);
draw('O·');
moveto(127, 118);
draw('l·');
moveto(160, 118);
draw('2·');
while not grid do
  begin
    repeat
      until Button;
    getmouse(x, y);
    repeat
      until not button;
  end;
if ((x >= 100 and x <= 164) and (y >= 100 and y <= 106)) then
  begin
    gridsize := (x - 100);
    writeln(gridsize);
    moveto(50, 50);
    penpat(white);
    paintrect(0, 0, 234, 234);
    penpat(black);
    grid := true;
  end
else
  begin
    sysbeep(8);
  end;
end;(gridsetup)
procedure windowinit;
var
drawcoord, textcoord : rect;
begin

with textcoord do
begin
  textcoord.top := 288;
  textcoord.left := 2;
  textcoord.bottom := 342;
  textcoord.right := 512;
end;
with drawcoord do
begin
  drawcoord.top := 40;
  drawcoord.left := 2;
  drawcoord.bottom := 290;
  drawcoord.right := 512;
end;
hideall;
settextrect(textcoord);
showtext;
setdrawingrect(drawcoord);
showdrawing;
end; {windoinit}

procedure zset (zpixel : integer);
const
disp = -32:
var
  stp, i : integer;
begin
  stp := 4;
moveto(470, 10);
line(0, 192);
moveto(470, 10);
line(9, 0);
for i := 1 to 48 do
begin
  moveto(470, 10 + stp);
  if (i / 4) = round(i / 4) then
    line(9, 0)
  else
    line(6, 0)
  else
    line(3, 0);
  stp := stp + 4;
end;
textsize(8);
moveto(482, 14);
writtendraw('+2-');
moveto(482, 14 + disp);
writedraw('+1·');
moveto(482, 14 + 2 * disp);
writedraw('0·');
moveto(482, 14 + 3 * disp);
writedraw('-1·');
moveto(482, 14 + 4 * disp);
writedraw('-2·');
moveto(482, 14 + 5 * disp);
writedraw('-3·');
moveto(482, 14 + 6 * disp);
writedraw('-4·');
moveto(455, zpixel * (-1) + 74);
line(10, 0);
line(-4, -4);
move(4, 4);
line(-4, 4);
end; (z set)

procedure toolset (var cutting, chipload : integer);
type
  matarray = array[1..10] of string;
tooldata = array[1..10, 1..2] of integer;
lookuptable = array[0..20] of integer;
var
  matfile : text;
  matdatafile : text;
  material : matarray;
tool : tooldata;
lookup : lookuptable;
totalmat, i, ydisp, x, y, ym, xm, yround, ycheck, look : integer;
select : boolean;
beginn
  look := 1;
  for i := 0 to 20 do
    begin
      if (trunc(i / 2)) = (i / 2) then
        begin
          lookup[i] := look;
          look := look + 1;
        end;
      end;
      reset(matfile, 'matfile');
      reset(matdatafile, 'matdatafile');
      readln(matdatafile, totalmat);
      for i := 1 to totalmat do
        begin
          readln(matfile, material[i]);
          readln(matdatafile, tool[i, 1], tool[i, 2]);
        end;
x := 20;
y := 60;
select := false;
setcursor(hand);
textface([bold]);
textsize(12);
moveto(x + 40, 30);
drawstring('MATERIAL');
moveto(x + 160, 30);
drawstring('CUTTING SPEED(SFM)');
moveto(x + 320, 30);
drawstring('CHIP LOAD');
textface([]);
penpat(black);
pensize(1, 1);
for i := 1 to totalmat do
begin
  drawbox(x, y – 10, 10);
moveto(x + 40, y);
drawstring(material[i]);
moveto(x + 170, y);
writeln(tool[i, 1] : 6);
moveto(x + 330, y);
writeln(tool[i, 2] / 1000 : 6 : 4);
y := y + 20;
end;
drawbox(x, y – 10, 10);
moveto(x + 40, y);
drawstring('Pick your own');
while not select do
begin
  repeat
    until Button;
  getmouse(xm, ym);
  if (xm > 20) and (xm < 30) and (ym >= 50) then
  begin
    ycheck := ym – 50;
    yround := trunc(ycheck / 10);
    if yround <= (2 * totalmat – 2) then
    begin
      if ((trunc(yround / 2)) = (yround / 2)) then
      begin
        i := lookup[yround];
        select := true;
      end
      else
      begin
        sysbeep(8);
      end
    end
    else
    begin
      if (xm > 20) and (xm < 30) and (ym > (y – 10)) and (ym < y) then
      begin
        if (totalmat <= 7) then
        else
        begin
          If (totalmat <= 7) then
begin
  close(matfile);
  close(matdatafile);
  totalmat := totalmat + 1;
  writeln('Enter New Material Name');
  readln(material[totalmat]);
  writeln('Enter the CUTTING Speed (SFM)');
  readln(tool[totalmat, 1]);
  writeln('Enter the Chip Load (thousandths)');
  readln(tool[totalmat, 2]);
  rewrite(matfile, 'matfile');
  rewrite(matdatafile, 'matdatafile');
  writeln(matdatafile, totalmat);
  select := true;
  for i := 1 to totalmat do
    begin
      writeln(matfile, material[i]);
      writeln(matdatafile, tool[i, 1], tool[i, 2]);
    end;
  close(matfile);
  close(matdatafile);
  reset(matfile, 'matfile');
  reset(matdatafile, 'matdatafile');
  readln(matdatafile, totalmat);
  for i := 1 to totalmat do
    begin
      readln(matfile, material[i]);
      readln(matdatafile, tool[i, 1], tool[i, 2]);
    end;
  i := totalmat;
end
else
  begin
    sysbeep(8);
    sysbeep(8);
    writeln('MAXIMUM ACTIVE MATERIALS IS 8');
    writeln('Purge some materials first');
  end;
end
else
  begin
    sysbeep(8);
  end;
end
repeat
  until not button;
end;
y := 40 + (i * 20);
pensize(4, 4);
drawbox(x, y - 10, 10);
pensize(1, 1);
cutting := tool[i, 1];
chipload := tool[i, 2];
for i := 1 to 400 do
  begin
    penpat(white);
paintrect(0, 0, 342, 512);
    penpat(black);
    setcursor(cross);
  end;{toolset}

procedure mtrack (snap : boolean;
                 snapsize, xzero, yzero : integer;
                 var x, y : integer);
begin
repeat until Button;
  repeat
    getmouse(x, y);
    if snap then
      begin
        x := snapsize • trunc((x + (snapsize / 2)) / snapsize);
        y := snapsize • round(y / snapsize);
      end;
    if (x <= 47) then
      setcursor(hand)
    else
      setcursor(cross);
xcor := (x - xzero) / 32; {32 pixels make an inch}
ycor := (yzero - y) / 32;
showxy(xcor, ycor);
until not button;
end;

procedure frdecision (var pocket, inside : boolean);
var
  xf, yf : integer;
  choice : boolean;
begin
  choice := false;
drawdialog;
  setcursor(hand);
moveto(425, 80);
drawstring('FRAME');
moveto(421, 150);
drawstring('POCKET');
penpat(black);
while not choice do
  begin
    repeat
      until Button;
  end;
end;
repeat
    getmouse(xf, yf);
until not button;
if (xf > 405) and (xf < 485) and (yf > 125) and (yf < 165) then
begin
    pocket := true;
    inside := true;
    choice := true;
end
else if (xf > 405) and (xf < 485) and (yf > 55) and (yf < 95) then
begin
    pocket := false;
    choice := true;
end;
if not choice then
    sysbeep(8);
end;
if not pocket then
begin
    choice := false;
    penpat(white);
paintrect(70, 422, 90, 480);
paintrect(136, 417, 153, 480);
moveto(423, 80);
drawstring('INSIDE');
moveto(418, 150);
drawstring('OUTSIDE');
while not choice do
begin
    repeat
    until Button;

repeat
    getmouse(xf, yf);
until not button;
if (xf > 405) and (xf < 485) and (yf > 125) and (yf < 165) then
begin
    inside := false;
    choice := true;
end
else if (xf > 405) and (xf < 485) and (yf > 55) and (yf < 95) then
begin
    inside := true;
    choice := true;
end;
if not choice then
    sysbeep(8);
end;
end
else
inside := true;
[end if]
erasedialog;
setcursor(cross);
end;

procedure framemill (snap : boolean;
napsize, xzero, yzero, zpixel : integer;
var u : integer;
var dat : dtarray);

var
x, y, i, radius, offset, psize, depth : integer;
xcor, ycor, inchradius : real;
rec : rect;
pocket, inside : boolean;
recpat : pattern;

begin

textsize(12);
moveto(405, 30);
textface([bold]);
drawstring('FRAME MILL');
textface([]);
frdecision(pocket, inside);
moveto(420, 25);
textsize(8);
textface([bold]);
setcursor(cross);
drawstring('PICK RADIUS');
textsize(12);
textface([]);
scaleselect(radius);
erasedialog;
moveto(420, 25);
textsize(8);
textface([bold]);
setcursor(cross);
drawstring('PICK DEPTH');
textsize(12);
textface([]);
scaleselect(depth);
erasedialog;
u := u + 1;
write('INDICATE TOP LEFT');
mtrack(snap, snapsize, xzero, yzero, x, y);
rec.left := x;
dat[u, 2] := x;
rec.top := y;
dat[u, 3] := y;
write('INDICATE BOTTOM RIGHT');
mtrack(snap, snapsize, xzero, yzero, x, y);
rec.right := x;
dat(u, 4) := x;
rec.bottom := y;
dat(u, 5) := y;
dat(u, 6) := radius;
dat(u, 7) := depth;
If pocket then
  begin
    dat(u, 1) := 172;
  end
else if inside then
  begin
    dat(u, 1) := 171;
  end
else
  dat(u, 1) := 170;
{end if}
writeln(dat[u, 1], dat[u, 2], dat[u, 3], dat[u, 4], dat[u, 5], dat[u, 6], dat[u, 7]);
If pocket then
  begin
    pnsize := 2;
    recpat := dkgray;
ofset := 3;
    recdraw(rec, radius, ofset, pnsize, recpat);
ofset := 0;
    recpat := black;
pnsize := 1;
    recdraw(rec, radius, ofset, pnsize, recpat);
  end
else if inside then
  begin
    pnsize := 1;
    recpat := gray;
ofset := 3;
    recdraw(rec, radius, ofset, pnsize, recpat);
ofset := 0;
    recpat := black;
pnsize := 1;
    recdraw(rec, radius, ofset, pnsize, recpat);
  end
else
  begin
    ofset := 0;
    recpat := black;
pnsize := 1;
    recdraw(rec, radius, ofset, pnsize, recpat);
pnsize := 1;
    recpat := gray;
ofset := -3;
    recdraw(rec, radius, ofset, pnsize, recpat);
  end;
{end if}
end; {frame mill}
procedure drillhole (snap: boolean;
    snapsize : integer;
    var u, xzero, yzero : integer;
    var dat : dtarray;
    var tooldia : real;
    cutting, chipload : integer);

    var
        xf, yf, xhole, yhole, rhole, dhole : integer;
        choice : boolean;
        drawhole : rect;

begin
    drawdialog;
    textsize(12);
    moveto(405, 20);
    textface([bold]);
    drawstring('DRILL HOLE');
    textface([]);
    textsize(8);
    moveto(426, 80);
    drawstring('CHANGE');
    moveto(421, 150);
    drawstring('PLACE');
    choice := false;
    while not choice do
        begin
            repeat
                getmouse(xf, yf);
            until not button;
            if (xf > 405) and (xf < 485) and (yf > 125) and (yf < 165) then
                begin
                    choice := true;
                end
            else if (xf > 405) and (xf < 485) and (yf > 55) and (yf < 95) then
                begin
                    erasedialog;
                    toolchange(dat, tlno, u, v, tool, spndl, feed, plunge, tooldia, cutting, chipload);
                    choice := true;
                end;
            if not choice then
                sysbeep(8);
        end;
    erasedialog;
    writeln('INDICATE THE DEPTH OF THE HOLE');
    moveto(430, 25);
    textsize(8);
    textface([bold]);
drawstring('PICK DEPTH');
setcursor(cross);
textsize(12);
textface([]);
scaleselect(dhole);
erasedialog;
writeln('Hole Diameter = ', tooldia : 6 : 4);
writeln('Hole depth is = ', dhole / 32 : 6 : 4);
writeln('INDICATE HOLE CENTER');
mttrack(snap, snapsize, xzero, yzero, xhole, yhole);
rhole := round(tooldia * 32 / 2);
drawhole.top := yhole - rhole;
drawhole.left := xhole - rhole;
drawhole.right := xhole + rhole;
drawhole.bottom := yhole + rhole;
frameoval(drawhole);
penpat(gray);
moveto(xhole, yhole - 2);
line(0, 5);
moveto(xhole - 2, yhole);
line(5, 0);
u := u + 1;
if (dhole / 32) < 0.5 then
  dat[u, 1] := 81
else
  dat[u, 1] := 83;
  dat[u, 2] := xhole;
  dat[u, 3] := yhole;
  dat[u, 4] := dhole;
  dat[u, 5] := round(tooldia * 10000);
writeln('X = ', xhole, ' y = ', yhole, ' Depth = ', dhole);
end; {DrillHole}

procedure circlemill (snap : boolean; 
snapsize, xzero, yzero, zpixel : integer;
var u : integer;
var dat : dtarray);
begin
  ofset := 3;
textsize(12);
moveto(405, 30);
textface([bold]);
drawstring('CIRCLE MILL');
textface([]);
frdecision(pocket, inside);
moveto(420, 25);
textsize(8);
textface([bold]);
drawstring("PICK RADIUS");
textsize(12);
textface([]);
scaleselect(radius);
erasedialog;
moveto(420, 25);
textsize(8);
textface([bold]);
drawstring("PICK DEPTH");
textsize(12);
textface([]);
scaleselect(depth);
erasedialog;
u := u + 1;
writeln("INDICATE CENTER OF CIRCLE");
mtrack(snap, snapsize, xzero, yzero, x, y);
rec.left := x - radius;
rec.right := x + radius;
rec.top := y - radius;
rec.bottom := y + radius;
dat(u, 2] := x;
dat(u, 3] := y;
dat(u, 4] := radius;
dat(u, 5] := depth;
writeln(dat(u, 1], dat(u, 2], dat(u, 3], dat(u, 4], dat(u, 5]);
penpat(black);
pensize(1, 1);
frameoval(rec);
If pocket then 
  begin 
    dat(u, 1] := 177;
    penpat(dkgray);
pensize(2, 2);
    rec.left := x - radius + offset;
    rec.right := x + radius - offset;
    rec.top := y - radius + offset;
    rec.bottom := y + radius - offset;
    frameoval(rec);
pensize(1, 1);
  end
else If inside then 
  begin 
    dat(u, 1] := 176;
    penpat(gray);
pensize(1, 1);
    rec.left := x - radius + offset;
    rec.right := x + radius - offset;
    rec.top := y - radius + offset;
    rec.bottom := y + radius - offset;
    frameoval(rec);
  end
else
begin
    dat[u, 1] := 175;
    penpat(gray);
    pensize(1, 1);
    rec.left := x - radius - offset;
    rec.right := x + radius + offset;
    rec.top := y - radius - offset;
    rec.bottom := y + radius + offset;
    frameoval(rec);
end;
[end if]
end; {circle mill}

procedure cuttercomp (snapsize, xzero, yzero : integer;
var u, xon, yon : integer;
var snap : boolean;
var dat : dtarray);

var
    x, y, xf, yf : integer;
    choice, result, oops : boolean;
begin
    textsize(12);
    moveto(405, 30);
    textface([bold]);
    drawstring('TOOL');
    textface([]);
    oops := false;
    choice := false;
    drawdialog;
    setcursor(hand);
    moveto(426, 80);
    drawstring('LEFT');
    moveto(421, 150);
    drawstring('RIGHT');
    penpat(black);
    while not choice do
begin
    repeat
        until Button;
    repeat
        getmouse(xf, yf);
        until not button;
    If (xf > 405) and (xf < 485) and (yf > 125) and (yf < 165) then
begin
    result := false;
    choice := true;
end
else if (xf > 405) and (xf < 485) and (yf > 55) and (yf < 95) then
begin
    result := true;
    choice := true;
end
else if (x >= 1) and (x <= 21) and (y >= -133) and (y <= 153) then
  begin
    oops := true;
    choice := true;
  end;
  if not choice then
    sysbeep(8);
end;
erase_dialog;
if not oops then
  begin
    sysbeep(8);
    writeln('Indicate Cutter Compensation turn on point');
mtrack(snap, snapsize, xzero, yzero, x, y);
  u := u + 1;
  if result = false then
    dat[u, 1] := 42
  else
    dat[u, 1] := 41;
  xon := x;
yon := y;
dat[u, 2] := x;
dat[u, 3] := y;
end;
setcursor(cross);
end;

procedure compoff (var u, xon, yon : integer;
  var dat : dtarray):
begin
  penpat(black);
  u := u + 1;
  dat[u, 1] := 40;
  dat[u, 2] := xon;
  dat[u, 3] := yon;
end;

procedure drawline (xold, yold, x, y integer;
  comp : boolean);
begin
  if comp then
    penpat(gray)
  else
    penpat(black);
moveto(xold, yold);
lineto(x, y);
end;(drawline)

(Line Draw Begins)
procedure line (snap : boolean;
  var comp : boolean;
  var snapsize, gridsize, x, y, zpixel, xzero, yzero : integer;
  var u, v : integer;
var dat : datarray);

var
  firstfl, xold, yold, xbegin, ybegin, i, xon, yon : integer;
  xcor, ycor : real;

begin
  pensize(2, 2);
  moveto(5, 18);
  lineto(18, 5);
  firstfl := 0;
  textsize(10);
  moveto(400, 30);
  xold := 0;
  yold := 0;
  textface([bold]);
  drawstring('CUTTER COMP??');
  textsize(12);
  textface([]);
  yesno(comp);
  if comp then
    begin
      cuttercomp(snapsize, xzero, yzero, u, xon, yon, snap, dat);
    end
  else
    comp := false;

  [LINE - MAIN EVENTLOOP]
  setcursor(cross);
  begin
    pensize(1, 1);
    while not (((x >= 1) and (x <= 21)) and ((y >= 133) and (y <= 153))) do
      begin
        repeat
          until Button;
          repeat
            getmouse(x, y);
            if snap and (x >= 48) then
              begin
                x := snapsize * trunc(x + (snapsize / 2)) / snapsize;
                y := snapsize * round(y / snapsize);
              end;
            if (x <= 47) then
              setcursor(hand)
            else
              setcursor(cross);
            xcor := (x - xzero) / 32; [32 pixels make an inch]
            ycor := (yzero - y) / 32;
            showxy(xcor, ycor);
            if not ((x >= 0) and (x <= 45) and (y >= -50) and (y <= 250)) then
              begin
                if not ((xold = 0) and (yold = 0)) then
                  begin

                end
              end
            end
          end
        end
      end
    end
  end
}
moveto(xold, yold);
lineto(x, y);
penmode(patxor);
for i := 1 to 200 do
begin
end;
moveto(xold, yold);
lineto(x, y);
penmode(patcopy);
end;
end;
until not button;

If not ((x >= -1) and (x <= 43)) and ((y >= 0) and (y <= 153)) then
begin
If firstfl = 0 then
begin
xbegin := x;
ybegin := y;
xold := x;
yold := y;
u := u + 1;
dat[u, 1] := 0;
dat[u, 2] := x;
dat[u, 3] := y;
dat[u, 4] := zpixel;
If comp then
{writeln(u, dat[u, 1], dat[u, 2], dat[u, 3], dat[u, 4])}
drawline(xold, yold, x, y, comp);
firstfl := 1;
end
else
begin
u := u + 1;
dat[u, 1] := 1;
dat[u, 2] := x;
dat[u, 3] := y;
dat[u, 4] := zpixel;
If comp then
dat[u, 7] := 1;
drawline(xold, yold, x, y, comp);
xold := x;
yold := y;
end;
end;
end;
end;
end;
If comp and (xbegin <> x) and (ybegin <> y) then
begin
drawline(xold, yold, xbegin, ybegin, comp);
u := u + 1;
dat[u, 1] := 1;
dat[u, 2] := xbegin;
dat[u, 3] := ybegin;
dat[u, 4] := zpixel;
If comp then
  dat[u, 7] := 1;
end;
plotgrid(gridsize);
penpat(white);
paintrect(2, 2, 20, 20);
penpat(black);
pensize(1, 1);
moveto(5, 18);
lineto(18, 5);
end;

procedure setsnap (var snapsize : integer;
  var snap : boolean);
begin
  snap := true;
moveto(430, 25);
textsize(8);
textface([bold]);
drawstring('SNAP SIZE');
setcursor(cross);
textsize(12);
textface([]);
scaleselect(snapsize);
textsize(12);
textface([]);
erasedialog;
If snapsize = 0 then
  snap := false;
end;

procedure zdepth (var zpixel : integer);
var
  x, y : integer;
  z : real;
  zval : boolean;
begin
  pensize(2, 2);
drawzicon;
pensize(1, 1);
zval := false;
zset(zpixel);

  while not zval do
  begin
    repeat
      until Button;
  end;
end;
repeat
    getmouse(x, y);
    xcor := (0);
    ycor := (74 - y) / 32;
    showxy(xcor, ycor);
    penpat(gray);
    moveto(455, y);
    lineto(465, y);
    lineto(461, y - 4);
    moveto(465, y);
    lineto(461, y + 4);
    penpat(white);
    moveto(455, y);
    lineto(465, y);
    lineto(461, y - 4);
    moveto(465, y);
    lineto(461, y + 4);
until not button;
penpat(black);
if ((x >= 469) and (x <= 480) and (y >= 10) and (y <= 203)) then
    begin
        zpixel := (74 - y) / 32;
        z := zpixel / 32;
        writeln(‘Z depth set to ‘, z, ‘: ‘, 6 : 4);
        erasedialog;
        zval := true;
    end
else
    begin
        sysbeep(8);
    end;
end;
penpat(white);
paintrect(113, 2, 130, 20);
penpat(black);
drawzicon;
end; {Z Depth Setting}

begin {MainProcedure}
    windowinit;
inithand;
initpencil;
inicross;
writeln(‘NOTE: Pick a Snapsiz9 of 0 for no Snap’);
setnap(snapsize, snap);
writeln(snap, snapsize);
zpixel := 0;
toolset(cutting, chipload);
u := 0;
v := 1;
x := 1;
y := 1;
gridsize := 0;
toolchange(dat, tino, u, v, tool, spndl, feed, plunge, tooldia, cutting, chipload);
gridset(gridsize);
setup(x, y, gridsize);
writein("INDICATE PART ORIGIN");
repeat
until Button;
repeat
getmouse(xzero, yzero);
if snap then
begin
  xzero := snapsize * trunc((xzero + (snapsize / 2)) / snapsize);
  yzero := snapsize * round(yzero / snapsize);
end;
until not button;

{MAIN PROCEDURE EVENTLOOP}
setcursor(cross);
repeat
begin
  repeat
    getmouse(x, y);
    if snap then
      begin
        x := snapsize * trunc((x + (snapsize / 2)) / snapsize);
        y := snapsize * round(y / snapsize);
      end;
      if (x <= 47) then
        setcursor(hand)
      else
        setcursor(cross);
      xcor := (x - xzero) / 32;
      ycor := (yzero - y) / 32;
      showxy(xcor, ycor);
    until Button;
    getmouse(x, y);
    if ((x >= 1) and (x <= 21)) and ((y >= 1) and (y <= 21)) then
      begin
        line(snap, comp, snapsize, gridsize, x, y, zpixel, xzero, yzero, u, v, dat);
        if comp then
          compoff(u, xon, yon, dat);
      end
    else if ((x >= 1) and (x <= 21)) and ((y >= 111) and (y <= 131)) then
      begin
        zdepth(zpixel);
      end
    else if ((x >= 1) and (x <= 21)) and ((y >= 89) and (y <= 109)) then
      begin
        post(zpixel, xzero, yzero, u, v, feed, plunge, dat, tino);
      end
    else if ((x >= 23) and (x <= 44)) and ((y >= 45) and (y <= 65)) then
      begin
        toolchange(dat, tino, u, v, tool, spndl, feed, plunge, tooldia, cutting, chipload);
      end
  end
end
else if (x >= 1) and (x <= 21) and ((y >= 45) and (y <= 65)) then
begin
  framemill(snap, snapsize, xzero, yzero, zpixel, u, dat);
end
else if (x >= 23) and (x <= 43) and ((y >= 23) and (y <= 43)) then
begin
  drillhole(snap, snapsize, u, xzero, yzero, dat, tooldia, cutting, chipload);
end
else if (x >= 1) and (x <= 21) and ((y >= 23) and (y <= 43)) then
begin
  username := false;
  saveall(username, snap, snapsize, gridsize, feed, xzero, yzero, u, v, tlno, dat);
  username := false;
  loaddrawing(username, snap, snapsize, gridsize, feed, xzero, yzero, u, v, tlno, dat);
end
else if (x >= 23) and (x <= 43) and ((y >= 1) and (y <= 21)) then
begin
  circlemill(snap, snapsize, xzero, yzero, zpixel, u, dat);
end
else if (x >= 1) and (x <= 21) and ((y >= 67) and (y <= 87)) then
begin
  username := true;
  saveall(username, snap, snapsize, gridsize, feed, xzero, yzero, u, v, tlno, dat);
end
else if (x >= 23) and (x <= 43) and ((y >= 67) and (y <= 87)) then
begin
  username := true;
  loaddrawing(username, snap, snapsize, gridsize, feed, xzero, yzero, u, v, tlno, dat);
end
else if (x >= 23) and (x <= 43) and ((y >= 133) and (y <= 153)) then
begin
  setsnap(snapsize, snap);
end
else if (x >= 1) and (x <= 21) and ((y >= 155) and (y <= 175)) then
begin
  username := false;
  saveall(username, snap, snapsize, gridsize, feed, xzero, yzero, u, v, tlno, dat);
  frview(snap, snapsize, gridsize, feed, xzero, yzero, u, v, plunge, dat, tina);
  username := false;
  loaddrawing(username, snap, snapsize, gridsize, feed, xzero, yzero, u, v, tlno, dat);
end
else if (x >= 23) and (x <= 43) and ((y >= 111) and (y <= 131)) then
begin
  textsize(10);
  moveto(400, 30);
  textface(\[bold]) ;
  drawstring('NEW DRAWING??');
  textsize(12);
  textface(\[]) ;
  yesno(result);
  If result then
  begin
    clearall;
  end
 остальному:

writeln('NOTE: Pick a Snapsize of 0 for no Snap');
set snap(snapsize, snap);
z pixel := 0;
tool set (cut ting, chip load);
u := 0;
v := 1;
x := 1;
y := 1;
g rids i ze := 0;
tool change (dat, t ln o, u, v, tool, sp ndl, feed, plunge, tool dia, cutting, chip load);
grid set (grid size);
setup (x, y, grid size);
writeln('INDICATE PART ORIGIN');
repeat
until Button;
repeat
getmouse(xzero, yzero);
if snap then
begin
xzero := snapsize \ trunc((xzero + (snapsize / 2)) / snapsize);
yzero := snapsize \ round(yzero / snapsize);
end;
until not button;
end;
repeat
until not button;
writeln(x, y);
end:
until ((x >= 23) and (x <= 43)) and ((y >= 89) and (y <= 109));
textface([ outline]);
textsize(18);
move to(29, 107);
draw char('E');
for i := 1 to 300 do
begin
end;
p enpat (white);
paint rect(91, 26, 109, 40);
text face([ ]); 
textsize (18);
move to (29, 107);
draw char('E');
end. {Main Procedure}
MacCam1.0

A CAD/CAM PACKAGE FOR
THE
MACINTOSH
AND
BRIDGEPORT

USER’S MANUAL

By
Dinesh Dhamija
SYSTEM INFORMATION

This software has been written in Lightspeed Pascal. It is presented as a compiled application and does not require the presence of the compiler. The minimum system requirements are a Macintosh computer with specifications as listed in appendix A and a Bridgeport Series I Milling machine, specified in appendix B.

DISK CONTENTS

The Mac-Cam system is contained on one floppy disk. A System and the finder are installed on the disk. There are three files that Mac Cam is composed of.

1) The Mac Cam Application
2) ‘MatFile’ - A list of materials available in the system
3) ‘MatDataFile’ - A file containing parameters for the corresponding material in ‘MatFile’

The Mac Cam application is the first file. This contains the entire code for the program. Two support files are used. They are ‘Matfile’ and ‘Matdatafile’. These files contain the material and spindle speed and feed specifications for the program. These must always be resident on the same disk as the application. For hard disk users, copy the application along with these two files onto the same directory.

Mac Cam uses two reserved file names. These files are generated from within the program. The first file is named ‘temp’. This file is created during a redraw or just before the front view routine is called, to store the current state of the drawing on the disk. This also serves a dual purpose of backups. If the user has chosen to redraw his screen, then the file ‘temp’ exists on the disk. In the event of a power outage, the user can re-start the system and choose the Load File option. The file name is given as ‘temp’ and the file is retrieved. This option may also be thought of as a ‘Revert to Previous’ implementation.

The second filename that the program uses is ‘Gfile’. This is the file that contains the G codes, once they are generated. This file is re-written, every time the G option is picked. If the user wants to save the G file that is generated, then he must rename the file at the operating system level. It was considered redundant to allow the saving of the Gfiles under various names, since the user can save the geometry, and the generation of the Gfile from the geometry takes no more that a minute at the most.

GETTING STARTED

Make a back up of the original Mac Cam disk. Store the original disk in a safe place and
use the back up copy for all the following instructions.

Insert the Mac Cam disk and turn on the computer system. Double Click on the Mac Cam application and the program is operational. Starting Mac Cam is as simple as that.

OPERATION

On starting Mac Cam, the system begins with prompting the user to select the material to be machined.

<table>
<thead>
<tr>
<th>MATERIAL</th>
<th>CUTTING SPEED (sfm)</th>
<th>CHIP LOAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine wax</td>
<td>400</td>
<td>.004</td>
</tr>
<tr>
<td>Aluminum</td>
<td>300</td>
<td>.004</td>
</tr>
<tr>
<td>Steel</td>
<td>100</td>
<td>.003</td>
</tr>
<tr>
<td>Pick your own</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The related parameters that are being set with the above options are the feed rate, the plunge rate (half the feed rate), and the spindle speed. These options are displayed on the screen, as well so the user may pick a material with the closest desired parameters if the specific material to be machined is unavailable. Cutting speed and chip load are explained in the main documentation.

The user may also choose to add a material that is not listed to the existing list of materials. This option is designed to make the system a learning system. At the present moment, the system does not allow more than 8 materials at a time, but this is not a problem since the user can edit ‘MatFile’ and ‘MatDatafile’ as per the instructions provided in the appendix C to add custom material files.

After the material selection, the user is prompted to select the tool number. This may be any value from 1 to 24, the maximum value that the Bridgeport can support. If the user is uncertain as to the correct tool number, tool number 1 is a good choice for a new part. It is wise to use up the tool numbers in sequential order, as and when required. It is to be noted though that a tool may be redefined and the program takes care of the G codes to reflect this change on the Bridgeport.
This is followed by a prompt to enter the tool diameter. This tool diameter will correspond to the tool number selected above. This diameter selected will be written into the tool chart on the Bridgeport when the program is downloaded. Each subsequent tool change will correspondingly require the user to enter a tool diameter.

At this point, the user must note that he takes entire responsibility for qualifying each tool he selects on the Bridgeport tool chart. A brief description of the qualification process is described below.
Each tool has associated with it, two parameters, the diameter and the tool length offset. The tool length offset is the distance from the tip of the tool to zero z depth on the part when the tool is mounted on the spindle and the spindle is retracted to the home position. It is obvious that this length is different for different tools as the tools may be of different lengths. Thus, if a tool requiring an offset of 2 inches is given an offset of 1 inch, it will crash into the part, it would be an inch too long. The program cannot take care of the tool length offset as this entirely depends on the position of the knee. There is no way that this position can be detected. Thus, each tool selected has to have it's offset set before machining. Please refer to the Bridgeport manual for detailed instructions on the setting of the tool length offset.

Tool definitions are followed by a prompt to select the grid size. The size of the grid may be anything from an eighth of an inch to two inches. The grid size must be specified. It was decided to force the user to use a grid as the kinds of drawings in this system will be geometric and will benefit greatly from a grid. Snap is optional, however, and the user can choose to ignore the grid during drawing.

At this point the system displays the main drawing window and is ready to begin. All the options for the menu are explained independently below and two examples are appended to this manual to give the user a feel for the system.
This option is on the top left of the menu area. It is used to generate straight lines and polygons on the system.

At the outset, the user is prompted if cutter compensation should be turned on. For further instructions on cutter compensation, see the section on cutter compensation. All cutter compensated paths are denoted by drawing the lines in gray on the screen. At this point the computer waits for the user to click the first point on the screen. The line draw routine enables chain connected lines to be drawn. This means that the user can keep clicking several points on the screen and the computer draws line segments between the last two points that have been clicked. This enables the user to generate any kind of polygon. If the user wants to exit the line draw routine, he clicks on the little cross icon on the bottom of the menu. This returns the user to the top level menu. Disjointed lines and separate polygons, or independent line segments may be constructed by repeatedly entering and exiting the line draw routine.

The Circle Mill icon is the small circle on the top line, right hand side of the menu. This option is used if a circular path is to be milled or a circular pocket is to machined in the part.
As the user clicks on this icon, he is asked if he wishes to machine a pocket or a frame. If the frame is opted for, then the next prompt is if the frame is an inside frame or an outside frame. This question is due to the fact that the circle mill routines use the Bridgeport's canned cycles. These canned cycles are automatically tool path compensated. If an inside frame is required, then the tool is compensated inward, that is the outer dimension of the circle will be the one that is as specified. If outside frame is specified, then the inside of the tool path will have the right dimension. In the case of the pocket mill, the cutter repeatedly traces out circular paths, stepping the tool over by a predetermined amount, removing the material to make the pocket.

Irrespective of the type of circular milling desired, the prompts from this point on are similar. The user is asked to select the size of the circle to be machined. This is done using the mouse over an icon that appears in the dialog box. The user is then asked to enter the total depth of the circular channel or the pocket, also entered via an icon in the dialog box. The program automatically assigns a step depth of an eighth of an inch so that if the total depth of the pocket or channel is large, the tool does not attempt to remove the material in a single pass.

Once these parameters are selected, the circle is displayed on the screen. The black circle denotes the actual dimension of the circle that is being machined, the gray circle of one pixel width on the inside, or the outside of the black circle indicates the type of compensation selected. If the inner circle is dark gray and two pixels wide, then this is the graphical representation of a pocket.

**REDRAW**

This option is on the second menu line, on the left hand side. This is used to redraw the screen as the user wishes, and also to serve as the temporary backup. Redraw may be necessary after exiting the line routine, because it's rubber band mode tends to erase any geometry that is previously drawn. This is because the rubber band mode is achieved by continuously erasing and redrawing the screen. A screen update just takes a copy of the data file and regenerates the geometry on the screen.

**DRILL HOLES**

This is represented by the circle and cross hairs icon on the second row. The hole drilling
routine is used both for regular and deep holes. For holes greater than 0.5" in depth, the software automatically selects the deep hole drilling mode. This mode starts drilling of the hole to 1/4" and retracts the quill for chip removal. From this point on, it drills the hole, pecking 1/8" at a time until the full depth specified by the user is attained.

On selecting this option, the diameter of the current tool is displayed and the user is first prompted if he wishes to change the tool. This option is given here since the user might have been using an end mill and would now like to change over to a drilling cutter, or if the cutter is a drill then he may wish to change the diameter. If a tool change is desired, then the tool change routine is called, just as explained in that section. The next prompt is for the depth of the hole. Any depth greater than 0.5" is automatically assigned the peck drill cycle by the program. The next prompt is for the location of the hole. This is done using the mouse. If snap was desired, then it should be turned on before selecting this option. Multiple holes are placed by repeatedly calling this routine.

FRAME MILL

This is the square icon that is on the third row on the left of the menu. This is used to mill rectangular frames, inside or outside and rectangular pockets, with or without fillet radii.

On selection of this option, the user is prompted if a frame or a pocket is required. If a frame is required, then the next prompt queries if it is an inside or an outside frame. From this point on the prompts for all options are similar.

The fillet radius is prompted for. If no fillet is desired, then a radius of 0" is selected. The depth of the pocket or the frame is the next prompt. This is the total depth desired. The user is then prompted to indicate the top left corner of the rectangle and the bottom right corner of the rectangle. At this the rectangle is drawn on the screen. A gray rectangle inside the black one, one pixel thick denotes an inside frame. A gray rectangle outside the black one, one pixel thick, denotes an outside frame. A dark gray rectangle inside the black one, two pixels thick denotes a pocket.

TOOL CHANGE

This is the ‘T’ icon on the third row of the menu. This icon is used to select a different tool while describing the geometry. There are 24 different tools that the Bridgeport can
support. The tool number may be reused if desired since the program updates the tool chart as it runs. It is however the user's responsibility to remember which tool was assigned to which number. At a later stage, this information may only be retrieved, based on the tool diameter. This may not always be an accurate indication since there may be more than one tool with the same diameter.

SAVE

The 'S' icon on the fourth row of the menu represents the Save routine. This is used to save geometry to disk. This geometry may later be retrieved through the Load option. On selecting this option, the user is prompted to enter a filename. The file is then saved to disk in an ASCII format, unique to this application.

LOAD

The load option is the converse of the save option. Any files saved previously may be retrieved by the load option. If the user has a current drawing on the screen, then he is asked if he would like to save his work before loading a file. If the load option is selected with no entities on the screen, then the "save current drawing" query is suppressed.

G CODE GENERATION

This option is used to generate the G codes for the Bridgeport from the geometry existent on the screen. This option must be selected after the user is satisfied with the drawing.

On selecting this option, the user is asked for a Bridgeport program name. This name must be numeric, and no greater than 4 characters in length in conformance to the Bridgeport file naming format. This name is then automatically appended on to the G codes generated. The G codes are generated in the order in which the user drew the object. Thus, if a specific machining order is preferred, then the user should draw the geometry in the desired order of preference. The G code file resulting from this option is saved under the name 'Gfile', including the control characters necessary for communications embedded in the file. Thus it is ready for downloading to the Bridgeport.
The ‘E’ icon on the fifth row of the menu is the exit icon. This option is used to exit the application. Make sure that your work is saved before you exit the application.

This is the option on the sixth row left of the menu. It is used for changing the current z depth that the geometry is being defined in. It is used by all routines, and some of the routines, particularly the bridgeport canned cycle routines, call this procedure automatically from within them. For the line drawing routine, however, the user takes responsibility for invoking this routine prior to defining the line path or the polygon.

This is the ‘N’ icon on the sixth row of the menu. If the user does not like the geometry on the screen, but wishes to re start the drawing, the new option is selected. This option is as good as exiting and restarting the application without actually doing so.

This option is related to the line draw routine. If the user wishes to exit from the line draw routine, he clicks on this icon. This icon is also used in conjunction with the line draw routine to draw separate polygons, disjointed lines and the like. See the line draw routine explanation for more details.

This is the dotted icon on the Seventh row of the menu structure. It is used to enable the user to snap on to specific locations on the screen. The snap may or may not be the same
as the grid. This feature is very helpful for drawing regular dimension objects, since it is very difficult to locate the cursor on an exact location on the screen.

On selecting this option, the user is asked to pick the snap size on the scale displayed in the dialog area. Snap may be reset at any time to any other value, by re-selecting this option. If snap is to be turned off, the snap size is selected to zero. Note that certain snap sizes will seem to be malfunctioning, but this is because of the way the user has selected the zero point or the part origin. Simply select a smaller snap size so that the zero reference and the snap size are in harmony again.

**FRONT VIEW**

This feature enables the user to visualize his creation a little better. It draws the front view of the tool path as the top view is being generated. The entire drawing is scaled and centered before displaying the front view and the user need not worry about doing anything special to his part generation to accommodate this option.

On selecting this option, the user is prompted if he wishes to step through the generation. It is normally a good idea to step through the generation since in this mode, the user sees the entity drawn in the top and front view simultaneously. The Z depths are also digitally indicated, using this option for additional confirmation. This option may be invoked as often as the user likes, even after each entity is placed in the geometry, so that the user gets a feel for the geometry as it is being generated.

**COMMUNICATIONS**

Communication to the Bridgeport is achieved using Red Ryder*, the communications program. The parameters to be set on both, the Bridgeport and the computer are as follows

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAUD RATE</td>
<td>600,1200,4800,9600</td>
</tr>
<tr>
<td>DATA LENGTH</td>
<td>7 Bits</td>
</tr>
<tr>
<td>PARITY</td>
<td>Even</td>
</tr>
<tr>
<td>STOP BIT</td>
<td>One</td>
</tr>
<tr>
<td>FLOW CONTROL</td>
<td>XON/XOFF</td>
</tr>
<tr>
<td>DUPLEX</td>
<td>FULL</td>
</tr>
</tbody>
</table>

*© Freesoft Company. To obtain a copy of Red Ryder, Contact Freesoft @ 150 Hickory Drive, Beaver Falls, PA 15010
This is provided in a file called 'bpsendfile' on the distribution disk. To set the parameters, just start up Red Ryder and pick the load settings from disk option. Use the filename 'bpsendfile' and the Macintosh is automatically set to the above parameters. On the Bridgeport side, all the above parameters are hard wired, except for the baud rate. Depress the options button on the Bridgeport until BAUD is displayed on the screen of the machine. Then punch in the new baud rate. It is preferable to set both ports A and B to the same baud rate.

The cable for communication should be connected to the communications port of the Macintosh. The Bridgeport Macintosh cabling diagram is included in the appendix with the Bridgeport specifications.

To send a file from the macintosh to the bridgeport, follow the directions below.

ON THE BRIDGEPORT

All references to keys in this section refer to the Bridgeport’s key panel unless otherwise specified. Make sure the cable is connected correctly. Depress the

LOAD
EDIT
CLEAR

key on the Bridgeport control panel until the screen on the bridgeport shows LOAD. Then depress the 0 key, to indicate that the load is from a remote terminal. Depress the 0 key again to signify that the LOAD is from remote. Do NOT enter a program name but hit the EXECUTE button until the prompt SETUP LOADING: appears on the top of the Bridgeport screen. At this point, the Bridgeport is waiting to receive a file.

ON THE MACINTOSH

Once the Bridgeport is set up as above, select ‘Send Text File’ option from the Red Ryder menu. Pick the filename of the file to be sent, and click the select button. Wait for a few seconds and the file will be transferred to the Bridgeport. The transfer is complete when the Bridgeport reports the size of it’s buffer and displays the beginning of the program on the Bridgeport screen.
FROM THE BRIDGEPORT TO THE MACINTOSH

Downloading the Bridgeport buffer from the Bridgeport to the Macintosh for backup purposes is also very simple. On the Bridgeport, depress the LOAD key until load appears on the control screen. Depress the 0 key to indicate a remote load and depress the 1 key to indicate that the load is to remote this time.

On the Macintosh, select the capture incoming file option and give a filename when prompted. When the Bridgeport has finished downloading, click the end capture option from the Red Ryder menu and the Bridgeport buffer is saved under the specified name.

USING THE EDITOR

Small changes to the uploaded program may be made through the Bridgeport’s editor port, designated port A. Set up the Macintosh with RedRyder and load the file ‘bpsendfile’ to set the communications parameters. Depress the EDIT key on the Bridgeport control panel and the Macintosh screen should show

```
*BOSS 9 EDITOR
```

This indicates that the editor is ready. Refer to the Bridgeport manual for a complete description of the editor and its functions.

IMPORTANT DETAILS

It is the user’s responsibility to make sure that the following parameters on the Bridgeport are set before machining a part.

1) Set the clear point at a convenient place to allow smooth tool changes.
2) Each tool that has been selected, has to be qualified and its tool length offset is to be set. *THIS STEP IS VERY IMPORTANT* and incorrect setting may cause a crash on the Bridgeport, leading to severe machine damage and a disintegrated tool.

3) The part origin selected on the drawing must be made to coincide with the part origin on the machine. The X,Y coordinates at the origin must be set to zero.

4) *NEVER OPERATE THE MACHINE IF YOU ARE UNSURE OF ANY SETTINGS.*

**DISCLAIMER**

The author has taken every care in the development of this software, and has tested it to the best of his ability. No warranties are made, however, express or implied, and the user of the software assumes all responsibility for the use of this software. This author and Ohio University are not responsible for any damage, loss of profit or any other inconvenience caused by the use of this product. Use of this product signifies acceptance of these conditions.
APPENDIX A

APPLE MACINTOSH COMPUTER SPECIFICATIONS

The computer used for this project has the following specifications:

- Macintosh, Macintosh 512, plus, SE or Macintosh II Computer
- One Printer Port
- One Modem/peripheral Communication Port
- One 800 K Double sided Internal Disk Drive
- One 800 K Double sided External Disk Drive
- ImageWriter or laserwriter Printer.

The compilers used in the development of the application is as follows:

- Macintosh Pascal Version 2.0 (For initial Development)
  (C) Apple Computer Inc, Cupertino, CA 95014.

- LightSpeed Pascal (For the final application)
  (C) 1986 Think Technologies Inc., Lexinton, MA 02173

The communications software used for the DNC link is

- Red Ryder Version 10.0
  (C) FreeSoft Company, Beaver Falls, PA 15010
BRIDGEPORT MACHINE SPECIFICATIONS

- Controlled 3 axes (x, y and z), rotary axis (optional)

- Simultaneous controllable axes
  - 3 axes of the four available (linear interpolation)
  - 2 axes in any plane (circular interpolation - XY, XZ, YZ)
  - X, Y axis circular, Z linear (Helical or spiral interpolation)

- Least input increment
  - Inch system : 0.0001
  - Metric system : 0.001 mm
  - Degrees : 0.001

- Least output increment
  - Inch System : 0.0001
  - Metric System : 0.00254 mm

- Maximum Programmable dimensions
  - Inch system : +/- 838.8607
  - Metric System : +/- 8388.607
  - Degrees A : +/- 720.000
  - Degrees C : +/- 8388.607

- Part program code, EIA RS-358-A, 7 bit, Even parity

- Part Program format : Variable block, Word address format

- Feed rate : Programmable directly between 0.1 and 250.0 inches per minute (ipm)

- Automatic Acceleration / de-acceleration

- Controller : R2E4, Acronym for Rigid Ram, 2 horsepower spindle motor, E type control, 3 axes milling machine

- Operating System : BOSS 9 (Bridgeport Operating System Software)
APPENDIX C

MATERIAL LIST MODIFICATION

This appendix is designed to allow the user to modify the material list that is included as a part of Mac Cam. The material information is stored in two files, 'MatFile' and 'MatDataFile'. MatFile has a list of the material names and MatDataFile has a corresponding list of parameters. The parameters stored in MatDataFile are the cutting speed of the material being machined in surface feet per minute and the chip load.

The very first entry in MatDataFile is the number of materials in the current list. The application is supplied with data for wax, carbon steel and aluminum. Thus this value will be 3. Each material in MatFile has a corresponding line in MatDataFile giving the cutting speed and chip load in thousandths.

For example, if the chip load of Aluminum is to be changed from 0.004 to 0.005, locate the position of aluminum in MatFile. If Aluminum is the second from the top, then the second line from the top in MatDatafile, after the number of entries variable, contains the relevant parameters. This line will have the cutting speed in surface feet per minute of aluminum followed by a few spaces and the number 4, representing the chip load of aluminum as 4 thousandths. Using any standard ASCII editor, delete the four and replace it with the number 5, representing 5 thousandths. Save MatDataFile and the changes are complete. The following is a specification of the file structures.

MATFILE

material 1
material 2
material 3

MATDATAFILE

number of materials in list
Material 1:   Data1  Data2
Material 2:   Data1  Data2
Material 3:   Data1  Data2
Sample files are as follows:

**MATFILE**

Aluminum  
Steel  
Wax

**MATDATAFILE**

3  
400  4  
100  3  
400  4

It is possible for the user to store multiple material files under different names for different groups of materials. Any specific material list may then be activated by copying the relevant data files to the working Mac Cam disk and renaming the files 'MatFile' and 'MatDataFile' respectively.
program parabola;

(Working Procedure Post, non Graphic Version)
(DINESH DHAMIIJA 05/25/88 - Program worked on this day)
(Program generates any parabola given the focus and the radius)
(Also generates the Gcodes for the BRIDGEPORT Milling Machine)

var
dfile, gfile : text;
focus, wide, xscale,yscale, scale, x, yf : real;
y, step : integer;
response : char;

procedure post (var dfile : text);

var
tooldia, maxblank, xinc, x, y, z : real;
cntr, toolno, spndl, feed, plunge, progunm : integer;
gfile : text;

begin
cntr := 1;
reset(dfile);
rewrite(gfile, 'Gfile');
write('Enter the program name (4 digits max) ');
readln(progunm);
write('Enter the tool number '); readln(toolno);
write('Enter the tool diameter(in inches) '); readln(tooldia);
write('Enter the spindle speed (rpm) '); readln(spndl);
write('Enter the feed rate (inches/min) '); readln(feed);
write('Enter the maximum blank size (inches) '); readln(maxblank);
maxblank := maxblank / 2;
plunge := round(feed / 2);
x := 0.0;
write(gfile, chr(18));
cntr := cntr + 1;
write(gfile, 'N', cntr : 1, 'G0X', x : 6 : 4, 'Y0.0Z', 0.05 : 6 : 4);
readln(dfile, x, z); {discarding the first data points - zero anyways}
write('Beginning processing of the Gfile');
write('Please wait .......');
while not eof(dfile) do
begin
readln(dfile, x, z);
z := (-1) * z;
cntr := cntr + 1;
repeat
begin
  cntr := cntr + 1;
  if cntr = 4 then
    writeln(gfile, 'N', cntr : 1, 'G2X', x : 6 : 4, 'Y0.010.J0.F', feed : 1, '.')
  else
    writeln(gfile, 'N', cntr : 1, 'G2X', x : 6 : 4, 'F', feed : 1, '.');
  x := x + tooldia / 2;
  cntr := cntr + 1;
  writeln(gfile, 'N', cntr : 1, 'G1X', x : 6 : 4, 'F', feed : 1, '.');
end;
until x >= maxblink;
end;
cntr := cntr + 1;
writeln(gfile, 'N', cntr : 1, 'G0X0.0Y0.0Z1.0M30');
writeln(gfile, chr(4), chr(14));
writeln('GFILE WRITTEN TO DISK SUCCESSFULLY');
end;

procedure drawsegment (xv, yv : integer);
begin
  lineto(xv, yv);
end; {drawsegment}

procedure Drawparabola (x, scale, focus : real);
var
  xv, yv : integer;
begin
  xv := round(scale * x);
  yv := round(scale * ((x * x) / (4.0 * focus)));
  drawsegment(xv, yv);
end; {drawparabola}

{MAIN PROGRAM BEGINS HERE}
begin
  rewrite(dfile, 'dfile');
pensize(2, 2);
write('Enter the Focus (inches ) '); readln(focus);
write('Enter the width (inches ) '); readln(wide);
write('How many steps should I take ?(the more the merrier ) '); readln(step);
xscale := 200 / (wide / 2);
yscale := 200 / focus;
if xscale < yscale then
  scale := xscale
else
  scale := yscale

scale := yscale;
writeln('Scale Factor for the graphics = ', scale : 3 : 4);
x := 0.0;
moveto(0, 0);
while x <= (wide / 2) do begin
  yf := ((x * x) / (4.0 * focus));
  writeln(dfile, x : 8 : 4, yf : 8 : 4);
  drawparabola(x, scale, focus);
  x := x + (wide / (2 * step));
end;
pensize(1, 1);
y := round(scale * focus);
lineto(0, y);
writeln('MACHINE this or RE-design (MIA)');
readln(response);
if (response = 'M') or (response = 'm') then post(dfile)
else begin
  writeln('Ok! I"m quitting');
  writeln('Re-run the program with different parameters');
end;
end.
PARABOLA EXAMPLE: FOCAL LENGTH 2 " , WIDTH 6"

:2222
N1 G0 G90 T1/ D0.2500 S1200 M26
N2 G0 X0.0000 Y0.0200 Z0.0500
N3 G1 X 0.2000 Z -0.0050 F10.
N4 G2 X0.2000 Y0.0100 J0. J0. F20.
N5 G1 X0.3250 F20.
N6 G2 X0.3250 F20.
N7 G1 X0.4500 F20.
N8 G2 X0.4500 F20.
N9 G1 X0.5750 F20.
N10 G2 X0.5750 F20.
N11 G1 X0.7000 F20.
N12 G2 X0.7000 F20.
N13 G1 X0.8250 F20.
N14 G2 X0.8250 F20.
N15 G1 X0.9500 F20.
N16 G2 X0.9500 F20.
N17 G1 X1.0750 F20.
N18 G2 X1.0750 F20.
N21 G1 X1.3250 F20.
N22 G2 X1.3250 F20.
N23 G1 X1.4500 F20.
N24 G2 X1.4500 F20.
N25 G1 X1.5750 F20.
N26 G2 X1.5750 F20.
N27 G1 X1.7000 F20.
N28 G2 X1.7000 F20.
N29 G1 X1.8250 F20.
N30 G2 X1.8250 F20.
N31 G1 X1.9500 F20.
N32 G2 X1.9500 F20.
N33 G1 X2.0750 F20.
N34 G2 X2.0750 F20.
N37 G1 X2.3250 F20.
N38 G2 X2.3250 F20.
N39 G1 X2.4500 F20.
N40 G2 X2.4500 F20.
N41 G1 X2.5750 F20.
N42 G2 X2.5750 F20.
N43 G1 X2.7000 F20.
N44 G2 X2.7000 F20.
N45 G1 X2.8250 F20.
N46 G2 X2.8250 F20.
N47 G1 X2.9500 F20.
N48 G2 X2.9500 F20.
N49 G1 X3.0750 F20.
N50 G1 X 0.4000 Z -0.0200 F10.
N51 G2 X0.4000 F20.
N52 G1 X0.5250 F20.
N53 G2 X0.5250 F20.
N54 G1 X0.6500 F20.
N55 G2 X0.6500 F20.
N56 G1 X0.7750 F20.
N57 G2 X0.7750 F20.
N58 G1 X0.9000 F20.
N59 G2 X0.9000 F20.
N60 G1 X1.0250 F20.
N61 G2 X1.0250 F20.
N62 G1 X1.1500 F20.
N63 G2 X1.1500 F20.
N64 G1 X1.2750 F20.
N65 G2 X1.2750 F20.
N66 G1 X1.4000 F20.
N67 G2 X1.4000 F20.
N68 G1 X1.5250 F20.
N69 G2 X1.5250 F20.
N70 G1 X1.6500 F20.
N71 G2 X1.6500 F20.
N72 G1 X1.7750 F20.
N73 G2 X1.7750 F20.
N74 G1 X1.9000 F20.
N75 G2 X1.9000 F20.
N76 G1 X2.0250 F20.
N77 G2 X2.0250 F20.
N78 G1 X2.1500 F20.
N79 G2 X2.1500 F20.
N80 G1 X2.2750 F20.
N81 G2 X2.2750 F20.
N82 G1 X2.4000 F20.
N83 G2 X2.4000 F20.
N84 G1 X2.5250 F20.
N85 G2 X2.5250 F20.
N86 G1 X2.6500 F20.
N87 G2 X2.6500 F20.
N88 G1 X2.7750 F20.
N89 G2 X2.7750 F20.
N90 G1 X2.9000 F20.
N91 G2 X2.9000 F20.
N92 G1 X3.0250 F20.
N93 G1 X 0.6000 Z -0.0450 F10.
N94 G2 X0.6000 F20.
N95 G1 X0.7250 F20.
N96 G2 X0.7250 F20.
N97 G1 X0.8500 F20.
N98 G2 X0.8500 F20.
N99 G1 X0.9750 F20.
N100 G2 X0.9750 F20.
N101 G1 X1.1000 F20.
N102 G2 X1.1000 F20.
N103 G1 X1.2250 F20.
N104 G2 X1.2250 F20.
N105 G1 X1.3500 F20.
N107 G1 X1.4750 F20.
N109 G1 X1.6000 F20.
N110 G2 X1.6000 F20.
N111 G1 X1.7250 F20.
N112 G2 X1.7250 F20.
G1 X1.8500 F20.
G2 X1.8500 F20.
G1 X1.9750 F20.
G2 X1.9750 F20.
G1 X2.1000 F20.
G2 X2.1000 F20.
G1 X2.2250 F20.
G2 X2.2250 F20.
G1 X2.3500 F20.
G2 X2.3500 F20.
G1 X2.4750 F20.
G2 X2.4750 F20.
G1 X2.6000 F20.
G2 X2.6000 F20.
G1 X2.7250 F20.
G2 X2.7250 F20.
G1 X2.8500 F20.
G2 X2.8500 F20.
G1 X2.9750 F20.
G2 X2.9750 F20.
G1 X3.1000 F20.
G2 X3.1000 F20.
G1 X0.8000 Z -0.0800 F10.
G2 X0.8000 F20.
G1 X0.9250 F20.
G2 X0.9250 F20.
G1 X1.0500 F20.
G2 X1.0500 F20.
G1 X1.1750 F20.
G2 X1.1750 F20.
G1 X1.3000 F20.
G2 X1.3000 F20.
G1 X1.4250 F20.
G2 X1.4250 F20.
G1 X1.5500 F20.
G2 X1.5500 F20.
G1 X1.6750 F20.
G2 X1.6750 F20.
G1 X1.8000 F20.
G2 X1.8000 F20.
G1 X1.9250 F20.
G2 X1.9250 F20.
G1 X2.0500 F20.
G2 X2.0500 F20.
G1 X2.1750 F20.
G2 X2.1750 F20.
G1 X2.3000 F20.
G2 X2.3000 F20.
G1 X2.4250 F20.
G2 X2.4250 F20.
G1 X2.5500 F20.
G2 X2.5500 F20.
G1 X2.6750 F20.
G2 X2.6750 F20.
G1 X2.8000 F20.
G2 X2.8000 F20.
G1 X2.9250 F20.
G2 X2.9250 F20.
G1 X3.0500 F20.
N171 G1X 1.0000Z -0.1250F10.
N172 G2X1.0000F20.
N173 G1X1.1250F20.
N175 G1X1.2500F20.
N176 G2X1.2500F20.
N177 G1X1.3750F20.
N178 G2X1.3750F20.
N179 G1X1.5000F20.
N180 G2X1.5000F20.
N181 G1X1.6250F20.
N182 G2X1.6250F20.
N183 G1X1.7500F20.
N184 G2X1.7500F20.
N185 G1X1.8750F20.
N186 G2X1.8750F20.
N187 G1X2.0000F20.
N188 G2X2.0000F20.
N189 G1X2.1250F20.
N190 G2X2.1250F20.
N191 G1X2.2500F20.
N192 G2X2.2500F20.
N193 G1X2.3750F20.
N194 G2X2.3750F20.
N195 G1X2.5000F20.
N196 G2X2.5000F20.
N197 G1X2.6250F20.
N198 G2X2.6250F20.
N199 G1X2.7500F20.
N200 G2X2.7500F20.
N201 G1X2.8750F20.
N202 G2X2.8750F20.
N203 G1X3.0000F20.
N204 G1X 1.2000Z -0.1800F10.
N206 G1X1.3250F20.
N207 G2X1.3250F20.
N208 G1X1.4500F20.
N209 G2X1.4500F20.
N210 G1X1.5750F20.
N211 G2X1.5750F20.
N212 G1X1.7000F20.
N213 G2X1.7000F20.
N214 G1X1.8250F20.
N216 G1X1.9500F20.
N218 G1X2.0750F20.
N219 G2X2.0750F20.
N222 G1X2.3250F20.
N223 G2X2.3250F20.
N224 G1X2.4500F20.
N225 G2X2.4500F20.
N226 G1X2.5750F20.
N227 G2X2.5750F20.
N228 G1X2.7000F20.
N229G2X2.7000F20.
N230G1X2.8250F20.
N231G2X2.8250F20.
N232G1X2.9500F20.
N233G2X2.9500F20.
N234G1X3.0750F20.
N235G1X 1.4000Z -0.2450F10.
N236G2X1.4000F20.
N237G1X1.5250F20.
N238G2X1.5250F20.
N239G1X1.6500F20.
N240G2X1.6500F20.
N241G1X1.7750F20.
N242G2X1.7750F20.
N243G1X1.9000F20.
N244G2X1.9000F20.
N245G1X2.0250F20.
N246G2X2.0250F20.
N247G1X2.1500F20.
N248G2X2.1500F20.
N249G1X2.2750F20.
N250G2X2.2750F20.
N251G1X2.4000F20.
N252G2X2.4000F20.
N253G1X2.5250F20.
N254G2X2.5250F20.
N255G1X2.6500F20.
N256G2X2.6500F20.
N257G1X2.7750F20.
N258G2X2.7750F20.
N259G1X2.9000F20.
N260G2X2.9000F20.
N261G1X3.0250F20.
N262G1X 1.6000Z -0.3200F10.
N263G2X1.6000F20.
N264G1X1.7250F20.
N265G2X1.7250F20.
N266G1X1.8500F20.
N267G2X1.8500F20.
N268G1X1.9750F20.
N269G2X1.9750F20.
N270G1X2.1000F20.
N271G2X2.1000F20.
N272G1X2.2250F20.
N273G2X2.2250F20.
N274G1X2.3500F20.
N275G2X2.3500F20.
N276G1X2.4750F20.
N277G2X2.4750F20.
N278G1X2.6000F20.
N279G2X2.6000F20.
N280G1X2.7250F20.
N281G2X2.7250F20.
N282G1X2.8500F20.
N283G2X2.8500F20.
N284G1X2.9750F20.
N285G2X2.9750F20.
N286G1X3.1000F20.
N345 G2 X2.6500 F20.
N346 G1 X2.7750 F20.
N347 G2 X2.7750 F20.
N348 G1 X2.9000 F20.
N349 G2 X2.9000 F20.
N350 G1 X3.0250 F20.
N351 G1 X 2.6000 Z -0.8450 F10.
N352 G2 X2.6000 F20.
N353 G1 X2.7250 F20.
N354 G2 X2.7250 F20.
N355 G1 X2.8500 F20.
N356 G2 X2.8500 F20.
N357 G1 X2.9750 F20.
N358 G2 X2.9750 F20.
N359 G1 X3.1000 F20.
N360 G1 X 2.8000 Z -0.9800 F10.
N361 G2 X2.8000 F20.
N362 G1 X2.9250 F20.
N363 G2 X2.9250 F20.
N364 G1 X3.0500 F20.
N365 G0 X0.0 Y0.0 Z1.0 M30